A LOGICAL FOUNDATION FOR ENVIRONMENT CLASSIFIERS

TAKESHI TSUKADA [a] AND ATSUSHI IGARASHI [b]

[a] Graduate School of Information Science, Tohoku University
  e-mail address: tsukada@kb.ecei.tohoku.ac.jp

[b] Graduate School of Informatics, Kyoto University
  e-mail address: igarashi@kuis.kyoto-u.ac.jp

ABSTRACT. Taha and Nielsen have developed a multi-stage calculus λ^α with a sound type system using the notion of *environment classifiers*. They are special identifiers, with which code fragments and variable declarations are annotated, and their scoping mechanism is used to ensure statically that certain code fragments are closed and safely runnable.

In this paper, we investigate the Curry-Howard isomorphism for environment classifiers by developing a typed λ-calculus $\lambda^\triangleright$. It corresponds to multi-modal logic that allows quantification by transition variables—a counterpart of classifiers—which range over (possibly empty) sequences of labeled transitions between possible worlds. This interpretation will reduce the "run" construct—which has a special typing rule in λ^α—and embedding of closed code into other code fragments of different stages—which would be only realized by the cross-stage persistence operator in λ^α—to merely a special case of classifier application. $\lambda^\triangleright$ enjoys not only basic properties including subject reduction, confluence, and strong normalization but also an important property as a multi-stage calculus: time-ordered normalization of full reduction.

Then, we develop a big-step evaluation semantics for an ML-like language based on $\lambda^\triangleright$ with its type system and prove that the evaluation of a well-typed $\lambda^\triangleright$ program is properly staged. We also identify a fragment of the language, where erasure evaluation is possible.

Finally, we show that the proof system augmented with a classical axiom is sound and complete with respect to a Kripke semantics of the logic.

1. INTRODUCTION

A number of programming languages and systems that support manipulation of programs as data [1, 2, 3, 4, 5] have been developed in the last two decades. A popular language abstraction in these languages consists of the Lisp-like *quasiquotation* mechanism to create and compose code fragments and a function to run them like `eval` in Lisp. For those languages and systems, a number of type systems for so-called "multi-stage" calculi have been studied [5, 6, 7, 8, 9, 10, 11] to guarantee safety of generated programs *even before the generating program runs*.

Among them, some seminal work on the principled design of type systems for multi-stage calculi is due to Davies [7] and Davies and Pfenning [12, 8]. They discovered the Curry-Howard isomorphism between modal/temporal logics and multi-stage calculi by identifying (1) modal operators in modal logic with type constructors for code fragments treated as data and, in the case of temporal logic, (2) the notion of time with computation stages. For example, the calculus $\lambda^{\bigcirc}$ [7], which can be thought as a reformulation of Glück and Jørgensen's calculus for multi-level generating extensions [6] by using explicit quasiquote and unquote in the language, corresponds to a fragment of linear-time temporal logic (LTL) with the temporal operator "next" (written $\bigcirc$) [13]. Here, linearly ordered time corresponds to the level of nesting of quasiquotations, and a modal formula $\bigcirc A$ to the type of *code* of type A. It, however, does not treat `eval`; in fact, the code type in $\lambda^{\bigcirc}$ represents code values, whose bodies are open, that is, may have free variables, so simply adding `eval` to the calculus does not work—execution may fail by referencing free variables in the code. The calculus developed by Davies and Pfenning [12, 8], on the other hand, corresponds to (intuitionistic) modal logic S4 (only with the necessity operator $\square$), in which a formula $\square A$ is considered the type of *closed code values* of type A. It supports safe `eval` since every code is closed, but inability to deal with open code hampers generation of efficient code. The following work by Taha and others [5, 14, 15, 9, 16] sought various forms of combinations of the two systems above to develop expressive type systems for multi-stage calculi.

Finally, Taha and Nielsen [9] developed a multi-stage calculus λ^{α}, which was later modified to make type inference possible [16] and implemented as a basis of MetaOCaml. The calculus λ^{α} has a strong type system while supporting open code, **run**, (which corresponds to `eval`), and a mechanism called cross-stage persistence (CSP), which allows a value to be embedded in a code fragment evaluated later. They introduced the notion of *environment classifiers* (or, simply, classifiers), which are special identifiers with which code fragments and variable declarations are annotated, to the type system. A key idea is to reduce the closedness checking of a code fragment (which is useful to guarantee the safety of **run**) to the freshness checking of a classifier. Unfortunately, however, correspondence to a logic is not clear for λ^{α} any longer, resulting in somewhat ad-hoc typing rules and complicated operational semantics, which would be difficult to adapt to different settings.

In this paper, we investigate a Curry-Howard isomorphism for environment classifiers by developing a typed λ-calculus $\lambda^{\triangleright}$. As a computational calculus, $\lambda^{\triangleright}$ is equipped with quasiquotation (annotated with environment classifiers) and abstraction over environment classifiers just like λ^{α}, with application of a classifier abstraction to a *possibly empty sequence of* environment classifiers, which makes $\lambda^{\triangleright}$ different from λ^{α}. Intuitively, (the type system of) $\lambda^{\triangleright}$ can be considered a proof system of a multi-modal logic to reason about deterministic labeled transition systems. Here, modal operators are indexed with transition labels, and so the logic is multi-modal. One notable feature of the logic is that it has quantification that allows one to express "for any state transitions," where a state transition is a possibly empty sequence of labels. This quantifier corresponds to types for classifier abstractions, used to ensure freshness of classifiers, which correspond to transition labels (and variables ranging over their sequences).

A pleasant effect of this logical interpretation—in particular, interpreting environment classifiers as variables ranging over sequences of transition labels—is that it will reduce the **run** construct—which has a peculiar typing rule in λ^{α}—and embedding of closed code into other code fragments of different stages—which would be only realized by the CSP operator in λ^{α}—to merely a special case of classifier application.

Our technical contributions can be summarized as follows:

- Identification of a modal logic that corresponds to (a computational calculus with) environment classifiers;
- Development of a new typed λ-calculus $\lambda^\triangleright$, naturally emerged from the correspondence, with its syntax, operational (small-step reduction and big-step evaluation) semantics, and type system;
- Proofs of basic properties, which a multi-stage calculus is expected to enjoy; and
- Proofs of soundness and completeness of the proof system (augmented with a classical axiom) with respect to a Kripke semantics of the logic.

Our calculus $\lambda^\triangleright$ not only enjoys the basic properties such as subject reduction, confluence, and strong normalization but also time-ordered normalization [7, 10], which says (full) reduction to a normal form can always be performed according to the order of stages. We extend $\lambda^\triangleright$ with base types and recursion, define a big-step evaluation semantics as a basis of a multi-stage programming language such as MetaOCaml, and prove the evaluation of a well-typed program is safe and staged, i.e., if a program of a code type evaluates to a result, it is a code value whose body is a well-typed program, again. We also develop erasure semantics, where information on classifiers is (mostly) discarded, and identify a subset of the language, where the original and erasure semantics agree, by an alternative type system. It turns out that the subset is rather similar to λ^i [16], whose type system is used in the current implementation of MetaOCaml.

One missing feature in $\lambda^\triangleright$ is CSP for all types of values but we do not think it is a big problem. First, CSP for primitive types such as integers is easy to add as a primitive; CSP for function types is also possible as long as they do not deal with open code, which, we believe, is usually the case. Second, as mentioned above, embedding closed code into code fragments of later stages is supported by a different means. It does not seem very easy to add CSP for open code to $\lambda^\triangleright$, but we think it is rarely needed. For more detail, see Section 6.3.

We can obtain a natural deduction proof system of a new logic that corresponds to the calculus $\lambda^\triangleright$ just by removing terms from typing rules, as usual. It is also easy to see that terms and reduction in the calculus correspond to proofs and proof normalization in the logic, respectively.

Of course, we should answer an important question: "What does this logic really mean?" We will elaborate the intuitive meaning of formulae in Section 2 and proof rules can be understood according to this informal interpretation but, to answer this question more precisely, one has to give a semantics and prove the proof system is sound and complete with respect to the semantics. However, the logic is intuitionistic and it is not straightforward to give (Kripke) semantics [17]. So, instead of Kripke semantics of the logic directly corresponding to $\lambda^\triangleright$, we give that of a classical version of the proof system, which has a proof rule for double negation elimination and prove that the proof system is sound and complete with it in Section 5. Even though the semantics does not really correspond to $\lambda^\triangleright$, it justifies our informal interpretation.

Organization of the Paper. In Section 2, we review λ^α and informally describe how the features of its type system correspond to those of a logic. In Section 3, we define the multi-stage calculus $\lambda^\triangleright$ and prove basic properties including subject reduction, strong normalization, confluence, and time-ordered normalization. In Section 4, we define MiniML$^\triangleright$,

an extension of $\lambda^{\triangleright}$ with base types and recursion, with its big-step semantics and prove that the big-step semantics implements staged execution. We also investigate erasure semantics of a subset of MiniML$^{\triangleright}$ here. In Section 5, we formally define (a classical version of) the logic and prove soundness and completeness of the proof system (augmented with a classical rule) with respect to a Kripke semantics. Lastly, we discuss related work and conclude.

2. Interpreting Environment Classifiers in a Modal Logic

In this section, we informally describe how environment classifiers can be interpreted in a modal logic. We start with reviewing Davies' $\lambda^{\bigcirc}$ [7] to get an intuition of how notions in a modal logic correspond to those in a multi-stage calculus. Then, along with reviewing main ideas of environment classifiers, we describe our logic informally and how our calculus $\lambda^{\triangleright}$ is different from λ^{α} by Taha and Nielsen [9].

2.1. $\lambda^{\bigcirc}$: Multi-Stage Calculus Based on LTL.
Davies has developed the typed multi-stage calculus $\lambda^{\bigcirc}$, which corresponds to a fragment of intuitionistic LTL by the Curry-Howard isomorphism. It can be considered the λ-calculus with a Lisp-like quasiquotation mechanism. We first review linear-time temporal logic and the correspondence between the logic and the calculus.

In LTL, the truth of propositions may depend on discrete and linearly ordered time, i.e., a given time has a unique time that follows it. Some of the standard temporal operators are $\bigcirc$ (to mean "next"), $\square$ (to mean "always"), and U (to mean "until"). The Kripke semantics of (classical) LTL can be given by taking the set of natural numbers as possible worlds;[1] then, for example, the semantics of $\bigcirc$ is given by: $n \Vdash \bigcirc\tau$ if and only if $n+1 \Vdash \tau$, where $n \Vdash \tau$ is the satisfaction relation, which means "τ is true in world—or, at time—n."

In addition to the usual Curry-Howard correspondence between propositions and types and between proofs and terms, Davies has pointed out additional correspondences between time and computation stages (i.e., levels of nested quotations) and between the temporal operator $\bigcirc$ and the type constructor meaning "the type of code of". So, for example, $\bigcirc\tau_1 \to \bigcirc\tau_2$, which means "if τ_1 holds at next time, then τ_2 holds at next time," is considered the type of functions that take a piece of code of type τ_1 and return another piece of code of type τ_2. According to this intuition, he has developed $\lambda^{\bigcirc}$, corresponding to the fragment of LTL only with $\bigcirc$.

$\lambda^{\bigcirc}$ has two new term constructors **next** M and **prev** M, which correspond to the introduction and elimination rules of $\bigcirc$, respectively. The type judgment of $\lambda^{\bigcirc}$ is of the form $\Gamma \vdash^n M : \tau$, where Γ is a context, M is a term, τ is a type (a proposition of LTL, only with $\bigcirc$) and n is a natural number indicating a stage. A context, which corresponds to assumptions, is a mapping from variables to pairs of a type and a natural number, since the truth of a proposition depends on time. The key typing rules are those for **next** and **prev**:

$$\frac{\Gamma \vdash^{n+1} M : \tau}{\Gamma \vdash^n \textbf{next } M : \bigcirc\tau} \qquad \frac{\Gamma \vdash^n M : \bigcirc\tau}{\Gamma \vdash^{n+1} \textbf{prev } M : \tau}.$$

---

[1]Note that this is equivalent to another, perhaps more standard presentation as a sublogic of CTL* [13].

The former means that, if M is of type τ at stage $n + 1$, then, at stage n, **next** M is code of type τ; the latter is its converse. Computationally, **next** and **prev** can be considered quasiquote and unquote, respectively. So, in addition to the standard β-reduction, $\lambda^\bigcirc$ has the reduction rule **prev** (**next** M) $\longrightarrow M$, which cancels **next** by **prev**.

The code types in $\lambda^\bigcirc$ are often called open code types, since the quoted code may contain free variables, so naively adding the construct to "run" quoted code does not work, since it may cause unbound variable errors.

Although the logic is considered intuitionistic, Davies has only shown that the proof system augmented with double negation elimination is equivalent to a standard axiomatic formulation [13], which is known to be sound and complete with the Kripke semantics described above. Kojima and Igarashi [18, 19] have studied the semantics of intuitionistic LTL and shown that the proof system obtained from $\lambda^\bigcirc$ is sound and complete with the given semantics. Even though the Kripke semantics discussed here does not really correspond to the logic obtained from the calculus, it certainly helps understand intuition behind the logic and we will continue to use Kripke semantics in what follows for an explanatory purpose.

2.2. **Multi-Modal Logic for Environment Classifiers.** Taha and Nielsen [9] have introduced environment classifiers to develop λ^α, which has quasiquotation, **run**, and CSP with a strong type system. We explain how λ^α can be derived from $\lambda^\bigcirc$.[2] Environment classifiers are a special kind of identifiers with which code types and quoting are annotated: for each classifier α, there are a type constructor $\langle\tau\rangle^\alpha$ for code and a term constructor $\langle M\rangle^\alpha$ to quote M. Then, a stage is naturally expressed by a sequence of classifiers, and a type judgment is of the form $\Gamma \vdash^A M : \tau$, where natural numbers in a $\lambda^\bigcirc$ type judgment are replaced with sequences A of classifiers. So, the typing rules of quoting and unquoting (written $\tilde{} M$) in λ^α are given as follows:

$$\frac{\Gamma \vdash^{A\alpha} M : \tau}{\Gamma \vdash^A \langle M\rangle^\alpha : \langle\tau\rangle^\alpha} \qquad \frac{\Gamma \vdash^A M : \langle\tau\rangle^\alpha}{\Gamma \vdash^{A\alpha} \tilde{} M : \tau}.$$

Obviously, this is a generalization of $\lambda^\bigcirc$: if only one classifier is allowed, then the calculus is essentially $\lambda^\bigcirc$.

The corresponding logic would also be a generalization of LTL, in which there are several "dimensions" of linearly ordered time. A Kripke frame for the logic is given by a transition system in which each transition relation is a map. More formally, a frame is a triple $(S, L, \{\xrightarrow{\alpha} \mid \alpha \in L\})$ where S is the (non-empty) set of states, L is the set of labels, and $\xrightarrow{\alpha} \in S \to S$ for each $\alpha \in L$. Then, the semantics of $\langle\tau\rangle^\alpha$ is given by: $s \Vdash \langle\tau\rangle^\alpha$ if and only if $s' \Vdash \tau$ for any s' such that $s \xrightarrow{\alpha} s'$, where s and s' are states.

The calculus λ^α has also a scoping mechanism for classifiers and it plays a central role to guarantee safety of **run**. The term $(\alpha)M$, which binds α in M, declares that α is used locally in M and such a local classifier can be instantiated with another classifier by term $M[\beta]$. We show typing rules for them with one for **run** below:

$$\frac{\Gamma \vdash^A M : \tau \qquad \alpha \notin \mathrm{FV}(\Gamma, A)}{\Gamma \vdash^A (\alpha)M : (\alpha)\tau} \qquad \frac{\Gamma \vdash^A M : (\alpha)\tau}{\Gamma \vdash^A M[\beta] : \tau[\alpha := \beta]} \qquad \frac{\Gamma \vdash^A M : (\alpha)\langle\tau\rangle^\alpha}{\Gamma \vdash^A \mathbf{run}\ M : (\alpha)\tau}.$$

---

[2]Unlike the original presentation, classifiers do not appear explicitly in contexts here. The typing rules shown are accordingly adapted.

The rule for $(\alpha)M$ requires that α does not occur in the context—the term M has no free variable[3] labeled α—and gives a type of the form $(\alpha)\tau$, which Taha and Nielsen called α-*closed type*, which characterizes a relaxed notion of closedness. For example, the term $\langle \lambda x : b.x \rangle^\alpha$ is a closed term, so this term is α-closed and the judgment $\emptyset \vdash^\varepsilon (\alpha)\langle \lambda x : b.x \rangle^\alpha : (\alpha)\langle b \to b \rangle^\alpha$ is valid. The term $\langle x \rangle^\alpha$, however, is not α-closed because this term has free variable x in the stage α, but β-closed (if $\beta \neq \alpha$) because there is no free variable in the stage containing the classifier β. The rule for **run** M says that an α-closed code fragment annotated with α can be run. Note that $\langle \cdot \rangle^\alpha$ (but not $(\alpha)\cdot$) is removed in the type of **run** M. Taha and Nielsen have shown that α-closedness is sufficient to guarantee safety of **run**.

When this system is to be interpreted as logic, it is fairly clear that $(\alpha)\tau$ is a kind of universal quantifier, as Taha and Nielsen have also pointed out [9]. Then, the question is "What does a classifier range over?", which has not really been answered so far. Another interesting question is "How can the typing rule for **run** be read logically?"

One plausible answer to the first question is that "classifiers range over the set of transition labels". This interpretation matches the rule for $M[\beta]$ and it seems that the typing rules without **run** (with a classical axiom) are sound and complete with the Kripke semantics that defines $s \Vdash (\alpha)\tau$ by $s \Vdash \tau[\alpha := \beta]$ for all $\beta \in L$. However, it is then difficult to explain the rule for **run**.

The key idea to solve this problem is to have classifiers range over the set of finite (and possibly empty) *sequences* of transition labels and to allow a classifier abstraction $(\alpha)M$ to be applied to also sequences of classifiers. Then, **run** will be unified to a special case of application of a classifier abstraction to the *empty* sequence. More concretely, we change the term $M[\beta]$ to $M[B]$, where B is a possibly empty sequence of classifiers (the left rule below). When B is empty and τ is $\langle \tau_0 \rangle^\alpha$ (assuming τ_0 do not include α), the rule (as shown as the right rule below) can be thought as the typing rule of (another version of) **run**, since α-closed code of τ_0 becomes simply τ_0 (without $(\alpha)\cdot$ as in the original λ^α).

$$\frac{\Gamma \vdash^A M : (\alpha)\tau}{\Gamma \vdash^A M[B] : \tau[\alpha := B]} \qquad \frac{\Gamma \vdash^A M : (\alpha)\langle \tau_0 \rangle^\alpha}{\Gamma \vdash^A M[\varepsilon] : \tau_0}$$

Another benefit of this change is that CSP for closed code (or embedding of persistent code [10]) can be easily expressed. For example, if x is of the type $(\alpha)\langle \mathbf{int} \rangle^\alpha$, then it can be used as code computing an integer at *different* stages as in, say, $\langle \cdots (\tilde{~}x[\alpha]) + 3 \cdots \langle \cdots 4 + (\tilde{~}\tilde{~}x[\alpha\beta]) \cdots \rangle^\beta \cdots \rangle^\alpha$. So, once a programmer obtains closed code, she can use it at any later stage. While our calculus $\lambda^\triangleright$ does not have a primitive of CSP for all types, we can express CSP in many cases. In Section 6.3, we discuss this subject in more detail.

Correspondingly, the semantics is now given by $v, \rho; s \Vdash \tau$ where v is a valuation for propositional variables and ρ is a mapping from classifiers to sequences of transition labels. Then, $v, \rho; s \Vdash \langle \tau \rangle^\alpha$ is defined by $v, \rho; s' \Vdash \tau$ where s' is reachable from s through the sequence $\rho(\alpha)$ of transitions and $v, \rho; s \Vdash (\alpha)\tau$ by: $v, \rho[A/\alpha]; s \Vdash \tau$ for any sequence A of labels ($\rho[A/\alpha]$ updates the value of α to be A). In Section 5, we give a formal definition of the Kripke semantics and show that the proof system, based in the ideas above, with double negation elimination is sound and complete with respect to it.

---

[3]It is important to distinguish labels of free variables from free occurrences of classifiers. For example, the term $\langle \lambda x : b.x \rangle^\alpha$ has free occurrence of the classifier α, but no free variable labeled by α because there is no free variable at all.

3. The Calculus $\lambda^{\triangleright}$

In this section, we define the calculus $\lambda^{\triangleright}$, based on the ideas described in the previous section: we first define its syntax, type system, and small-step full reduction semantics and states some basic properties; then we prove the time-ordered normalization property. Finally, we give an example of programming in $\lambda^{\triangleright}$. We intentionally make notations for type and term constructors different from λ^{α} because their precise meanings are different; it is also to avoid confusion when we compare the two calculi.

3.1. **Syntax.** Let Σ be a countably infinite set of *transition variables*, ranged over by α and β. A *transition*, denoted by A and B, is a finite sequence of transition variables; we write ε for the empty sequence and AB for the concatenation of the two transitions. We write Σ^* for the set of transitions. A transition is often called a *stage*. We write $\mathrm{FTV}(A)$ for the set of transition variables in A, defined by $\mathrm{FTV}(\alpha_1 \alpha_2 \ldots \alpha_n) = \{\alpha_i \mid 1 \leq i \leq n\}$.

Let PV be the set of *base types* (corresponding to propositional variables), ranged over by b. The set Φ of *types*, ranged over by τ and σ, is defined by the following grammar:

$$\textit{Types} \qquad \tau ::= b \mid \tau \to \tau \mid \triangleright_\alpha \tau \mid \forall \alpha.\tau \quad .$$

A type is a base type, a function type, a code type, which corresponds to $\langle \cdot \rangle^\alpha$ of λ^α, or an α-closed type, which corresponds to $(\alpha)\tau$. The transition variable α of $\forall \alpha.\tau$ is bound in τ. In what follows, we assume tacit renaming of bound variables in types. The type constructor $\triangleright_\alpha$ connects tighter than $\to$ and $\to$ tighter than $\forall$: for example, $\triangleright_\alpha \tau \to \sigma$ means $(\triangleright_\alpha \tau) \to \sigma$ and $\forall \alpha.\tau \to \sigma$ means $\forall \alpha.(\tau \to \sigma)$. We write $\mathrm{FTV}(\tau)$ for the set of free transition variables, which is defined in a straightforward manner.

Let Υ be a countably infinite set of *variables*, ranged over by x and y. The set of *terms*, ranged over by M and N, is defined by the following grammar:

$$\textit{Terms} \qquad M ::= x \mid M\,M \mid \lambda x : \tau.M \mid \blacktriangleright_\alpha M \mid \blacktriangleleft_\alpha M \mid \Lambda \alpha.M \mid M\,A \quad .$$

In addition to the standard λ-terms, there are four more terms, which correspond to $\langle M \rangle^\alpha$, $\tilde{}M$, $(\alpha)M$, and $M[\beta]$ of λ^α (respectively, in the order presented). Note that, unlike $\tilde{}M$ in λ^α, the term $\blacktriangleleft_\alpha M$ for unquote is also annotated. This annotation is needed because a single transition variable can be instantiated with a sequence, in which case a single unquote has to be duplicated accordingly. The variable x in $\lambda x : \tau.M$ and the transition variable α in $\Lambda \alpha.M$ are bound in M. Bound variables are tacitly renamed to avoid variable capture in substitution. We write $\mathrm{FTV}(M)$ for the set of free transition variables, which is defined in a straightforward manner, e.g., $\mathrm{FTV}(\blacktriangleright_\alpha M) = \mathrm{FTV}(\blacktriangleleft_\alpha M) = \mathrm{FTV}(M) \cup \{\alpha\}$, $\mathrm{FTV}(\Lambda \alpha.M) = \mathrm{FTV}(M) - \{\alpha\}$ and $\mathrm{FTV}(M\,A) = \mathrm{FTV}(M) \cup \mathrm{FTV}(A)$.

3.2. **Type System.** As mentioned above, a type judgment and variable declarations in a context are annotated with stages. A *context* Γ is a finite set $\{x_1 : \tau_1 @ A_1, \ldots, x_n : \tau_n @ A_n\}$, where x_i are distinct variables. We often omit braces $\{\}$. We write $\mathrm{FTV}(\Gamma)$ for the set of free transition variables in Γ, defined by: $\mathrm{FTV}(\{x_i : \tau_i @ A_i \mid 1 \leq i \leq n\}) = \bigcup_{i=1}^n (\mathrm{FTV}(\tau_i) \cup \mathrm{FTV}(A_i))$.

A *type judgment* is of the form $\Gamma \vdash^A M : \tau$, read "term M is given type τ under context Γ at stage A." Figure 1 presents the typing rules to derive type judgments. The notation $\tau[\alpha := B]$, used in the rule (INS), is capture-avoiding substitution of transition B for α in τ. When α in $\triangleright_\alpha$ is replaced by a transition, we identify $\triangleright_\varepsilon \tau$ with τ and $\triangleright_{AB}\tau$ with $\triangleright_A \triangleright_B \tau$. For example, $(\triangleright_\alpha \forall \alpha. \triangleright_\alpha b)[\alpha := \varepsilon] = \forall \alpha. \triangleright_\alpha b$ and $(\forall \alpha. \triangleright_\beta b)[\beta := \alpha\alpha] = \forall \alpha'. \triangleright_\alpha \triangleright_\alpha b$.

$$\frac{}{\Gamma, x : \tau @ A \vdash^A x : \tau} \; (\text{Var}) \qquad\qquad \frac{\Gamma \vdash^{A\alpha} M : \tau}{\Gamma \vdash^A \blacktriangleright_\alpha M : \triangleright_\alpha \tau} \; (\blacktriangleright)$$

$$\frac{\Gamma, x : \tau @ A \vdash^A M : \sigma}{\Gamma \vdash^A \lambda x : \tau.M : \tau \to \sigma} \; (\text{Abs}) \qquad\qquad \frac{\Gamma \vdash^A M : \triangleright_\alpha \tau}{\Gamma \vdash^{A\alpha} \blacktriangleleft_\alpha M : \tau} \; (\blacktriangleleft)$$

$$\frac{\Gamma \vdash^A M : \tau \to \sigma \qquad \Gamma \vdash^A N : \tau}{\Gamma \vdash^A M\,N : \sigma} \; (\text{App}) \qquad \frac{\Gamma \vdash^A M : \tau \qquad \alpha \notin \text{FTV}(\Gamma) \cup \text{FTV}(A)}{\Gamma \vdash^A \Lambda\alpha.M : \forall\alpha.\tau} \; (\text{Gen})$$

$$\frac{\Gamma \vdash^A M : \forall\alpha.\tau}{\Gamma \vdash^A M\,B : \tau[\alpha := B]} \; (\text{Ins})$$

Figure 1: Typing rules.

The first three rules on the left are mostly standard except for stage annotations. The conditions on stage annotations are similar to those in most multi-stage calculi: The rule (Var) means that variables can appear only at the stage in which those variables are declared. and the rule (Abs) requires the stage of the parameter to be the same as that of the body and, correpondingly, the rule (App) requires M and N are typed at the same stage. The next two rules ($\blacktriangleright$) and ($\blacktriangleleft$) are for quoting and unquoting and already explained in the previous section. The last two rules (Gen) and (Ins) are for generalization and instantiation of a transition variable, respectively. They resemble the introduction and elimination rules of $\forall x.A(x)$ in first-order predicate logic: the side condition of the rule (Gen) ensures that the choice of α is independent of the context. Computationally, this side condition expresses α-closedness of M, that means M has no free variable which has annotation α in its type or its stage. This is a weaker form of closedness, which means M has no free variable at all.

3.3. **Reduction.** We will introduce full reduction $M \longrightarrow N$, read "M reduces to N in one step," and prove basic properties including subject reduction, confluence and strong normalization.

Before giving the definition of reduction, we define substitution. Since the calculus has binders for term variables and transition variables, we need two kinds of substitutions for both kinds of variables. Substitution $M[x := N]$ for a term variable is the standard capture-avoiding one, and its definition is omitted here. Substitution $M[\alpha := A]$ of A for α is defined similarly to $\tau[\alpha := A]$. We show representative cases below:

$$\begin{aligned}
(\lambda x : \tau.M)[\alpha := A] &= \lambda x : (\tau[\alpha := A]).(M[\alpha := A]) \\
(M\,B)[\alpha := A] &= (M[\alpha := A])\,(B[\alpha := A]) \\
(\blacktriangleright_\beta M)[\alpha := A] &= \blacktriangleright_{\beta[\alpha := A]}(M[\alpha := A]) \\
(\blacktriangleleft_\beta M)[\alpha := A] &= \blacktriangleleft_{\beta[\alpha := A]}(M[\alpha := A]),
\end{aligned}$$

where $\blacktriangleright_{\alpha_1 \ldots \alpha_n} M$ is an abbreviation for $\blacktriangleright_{\alpha_1} \cdots \blacktriangleright_{\alpha_n} M$, and $\blacktriangleleft_{\alpha_1 \ldots \alpha_n} M$ for $\blacktriangleleft_{\alpha_n} \cdots \blacktriangleleft_{\alpha_1} M$. In particular, $(\blacktriangleright_\alpha M)[\alpha := \varepsilon] = (\blacktriangleleft_\alpha M)[\alpha := \varepsilon] = M[\alpha := \varepsilon]$. Note that, when a transition variable in $\blacktriangleleft$ is replaced, the order of transition variables is reversed, because this is the inverse operation of $\blacktriangleright$. This is similar to the inversion operation in group theory: $(a_1 a_2 \ldots a_n)^{-1} = a_n^{-1} a_{n-1}^{-1} \ldots a_1^{-1}$.

The *reduction relation* $M \longrightarrow N$ is the least relation closed under the following three computation rules

$$(\lambda x : \tau.M) \, N \longrightarrow M[x := N] \qquad \blacktriangleleft_\alpha(\blacktriangleright_\alpha M) \longrightarrow M \qquad (\Lambda\alpha.M) \, A \longrightarrow M[\alpha := A]$$

and congruence rules, which are omitted here. In addition to the standard β-reduction, there are two rules: the second one, which is already explained previously, cancels quote by unquote and the last one, instantiation of a transition variable, is similar to polymorphic function application in System F. Note that the reduction is full—reduction occurs under any context in any stage. This reduction relation can be thought as (non-deterministic) proof normalization, which should preserve types, be confluent and strongly normalizing. Then, we will define another reduction relation as a triple $M \xrightarrow{T} N$, with T standing for the stage of reduction in Section 3.5, as done in $\lambda^\bigcirc$ [7] and $\lambda^{\bigcirc\square}$ [10], to prove time-ordered normalization.

3.4. **Basic Properties.** We will prove three basic properties, namely, subject reduction, strong normalization and confluence.

The key lemma is, as usual, Substitution Lemma, which says substitution preserves typing. We will prove such a property for each kind of substitution. We define substitution $\Gamma[\alpha := A]$ for contexts as follows:

$$\{x_i : \tau_i@A_i\}[\alpha := A] = \{x_i : \tau_i[\alpha := A]@A_i[\alpha := A]\}$$

Lemma 3.1 (Substitution Lemma)**.**
(1) If $\Gamma, x : \sigma@B \vdash^A M : \tau$ and $\Gamma \vdash^B N : \sigma$, then $\Gamma \vdash^A M[x := N] : \tau$.
(2) If $\Gamma \vdash^A M : \tau$, then $\Gamma[\alpha := B] \vdash^{A[\alpha:=B]} M[\alpha := B] : \tau[\alpha := B]$.

Proof. Easy induction on the typing rules. We only show main cases.

The proof of (1):
• Case $M = x$: It is the case that $\tau = \sigma$ and $A = B$. So, what we have to show is $\Gamma \vdash^B N : \tau$, which is already assumed.

• Case $M = M_1 M_2$: By the typing rules, we know that $\Gamma, x : \sigma@B \vdash^A M_1 : \tau_0 \to \tau$ and $\Gamma, x : \sigma@B \vdash^A M_2 : \tau_0$ for some τ_0. By the induction hypothesis, $\Gamma \vdash^A (M_1[x := N]) : \tau_0 \to \tau$ and $\Gamma \vdash^A (M_2[x := N]) : \tau_0$. By the rule (APP), we have $\Gamma \vdash^A (M_1 M_2)[x := N] : \tau$.

• Case $M = \blacktriangleright_\alpha M_0$: By the typing rules, we know that $\tau = \triangleright_\alpha\tau_0$ and $\Gamma, x : \sigma@B \vdash^{A\alpha} M_0 : \tau_0$. By the induction hypothesis, $\Gamma \vdash^{A\alpha} M_0[x := N] : \tau_0$. So, we obtain $\Gamma \vdash^A (\blacktriangleright_\alpha M_0)[x := N] : \triangleright_\alpha\tau_0$.

The proof of (2):
• Case $M = x$: By the typing rules, $x : \tau@A \in \Gamma$. So, $x : (\tau[\alpha := B]) : (A[\alpha := B]) \in \Gamma[\alpha := B]$. Therefore, $\Gamma[\alpha := B] \vdash^{A[\alpha:=B]} x : \tau[\alpha := B]$.

• Case $M = \blacktriangleright_\beta M_0$: By the typing rules, we have $\tau = \triangleright_\beta\tau_0$ and $\Gamma \vdash^{A\beta} M_0 : \tau_0$. By the induction hypothesis, $\Gamma[\alpha := B] \vdash^{(A[\alpha:=B])(\beta[\alpha:=B])} M_0[\alpha := B] : \tau_0[\alpha := B]$. By applying the rule ($\blacktriangleright$) as many times as required, we obtain $\Gamma[\alpha := B] \vdash^{A[\alpha:=B]} \blacktriangleright_{\beta[\alpha:=B]}(M_0[\alpha := B]) : \triangleright_{\beta[\alpha:=B]}(\tau_0[\alpha := B])$. By the definition of substitution, this is equal to what we have to show.

- Case $M = \Lambda\beta.M_0$: By the typing rules, we have $\tau = \forall\beta.\tau_0$ and $\Gamma \vdash^A M_0 : \tau_0$. Moreover we can assume without loss of generality that $\beta \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(A) \cup \mathrm{FTV}(B) \cup \{\alpha\}$. By the induction hypothesis, we have $\Gamma[\alpha := B] \vdash^{A[\alpha:=B]} M_0[\alpha := B] : \tau_0[\alpha := B]$. Because $\beta \notin \mathrm{FTV}(\Gamma[\alpha := B]) \cup \mathrm{FTV}(A[\alpha := B])$, we obtain $\Gamma[\alpha := B] \vdash^{A[\alpha:=B]} (\Lambda\beta.(M_0[\alpha := B])) : \forall\beta.(\tau_0[\alpha := B])$. This judgement is equal to what we have to show. $\qquad\square$

Theorem 3.2 (Subject Reduction). *If $\Gamma \vdash^A M : \tau$ and $M \longrightarrow M'$, then $\Gamma \vdash^A M' : \tau$.*

Proof. By straightforward induction on the derivation of $M \longrightarrow M'$, using Substitution Lemma (Lemma 3.1). We only show three base cases and omit induction steps.

- Case $M = (\lambda x : \tau_0.M_0) M_1 \longrightarrow M_0[x := M_1] = M'$: Because $\Gamma \vdash^A (\lambda x : \tau_0.M_0) M_1 : \tau$, we have $\Gamma, x : \tau_0 @ A \vdash^A M_0 : \tau$ and $\Gamma \vdash^A M_1 : \tau_0$. By Substitution Lemma (1), we obtain $\Gamma \vdash^A M_0[x := M_1] : \tau$.

- Case $M = (\Lambda\alpha.M_0) B \longrightarrow M_0[\alpha := B] = M'$: Because $\Gamma \vdash^A (\Lambda\alpha.M_0) B : \tau$, we have $\alpha \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(A)$ and $\Gamma \vdash^A M_0 : \tau_0$ such that $\tau_0[\alpha := B] = \tau$. By the second statement of Substitution Lemma (Lemma 3.1), we obtain $\Gamma[\alpha := B] \vdash^{A[\alpha:=B]} M_0[\alpha := B] : \tau_0[\alpha := B]$. Because $\alpha \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(A)$, $\Gamma[\alpha := B] = \Gamma$ and $A[\alpha := B] = A$. Therefore, $\Gamma \vdash^A M_0[\alpha := B] : \tau$.

- Case $M = \blacktriangleleft_\alpha \blacktriangleright_\alpha M_0 \longrightarrow M_0 = M'$: By replacing M in the assumption, we obtain $\Gamma \vdash^A \blacktriangleleft_\alpha \blacktriangleright_\alpha M_0 : \tau$. So we have $\Gamma \vdash^A M_0 : \tau$ as required. $\qquad\square$

Theorem 3.3 (Strong Normalization). *Let M be a typable term. There is no infinite reduction sequence $M \longrightarrow N_1 \longrightarrow N_2 \longrightarrow \cdots$.*

Proof. We construct a term $\natural(M)$ of the simply typed λ-calculus ($\lambda^\rightarrow$) as follows:

$$
\begin{aligned}
\natural(x) &= x \\
\natural(\lambda x : \tau.N) &= \lambda x.\natural(N) \\
\natural(N_1 N_2) &= \natural(N_1)\,\natural(N_2) \\
\natural(\blacktriangleright_\alpha N) &= \natural(N) \\
\natural(\blacktriangleleft_\alpha N) &= \natural(N) \\
\natural(\Lambda\alpha.N) &= \natural(N) \\
\natural(N\,A) &= \natural(N)
\end{aligned}
$$

We can easily prove the following propositions by induction of the structure of M:

(1) If M is typable in $\lambda^\triangleright$, then $\natural(M)$ is typable in $\lambda^\rightarrow$.
(2) Suppose $M \longrightarrow M'$. If this is a β-reduction step, then $\natural(M) \longrightarrow \natural(M')$ in $\lambda^\rightarrow$. Otherwise, $\natural(M) = \natural(M')$.

Now, assume there exists an infinite reduction sequence from a typable term M. It is clear that there are infinitely many β-reduction. By (2), there exists an infinite reduction sequence from $\natural(M)$, which is typable by (1). This contradicts the strong normalization property of $\lambda^\rightarrow$. $\qquad\square$

The last property we will show is confluence. We prove this by using parallel reduction and complete development [20]. We define the *parallel reduction* relation $M \Longrightarrow N$ as in Figure 2. Notice that, the rule (P-$\blacktriangleleft\blacktriangleright$) allows more than one nested pairs of quoting and unquoting to be cancelled in one step: For example, $\blacktriangleleft_{\alpha_1} \blacktriangleleft_{\alpha_2} \ldots \blacktriangleleft_{\alpha_n} \blacktriangleright_{\alpha_n} \ldots \blacktriangleright_{\alpha_2} \blacktriangleright_{\alpha_1} x \Longrightarrow$

$$\frac{}{x \Longrightarrow x} \ (\text{P-Var}) \qquad\qquad \frac{M \Longrightarrow N}{M \, A \Longrightarrow N \, A} \ (\text{P-TApp})$$

$$\frac{M \Longrightarrow N}{\lambda x : \tau.M \Longrightarrow \lambda x : \tau.N} \ (\text{P-Lam}) \qquad \frac{M \Longrightarrow N}{(\Lambda \alpha.M) \, A \Longrightarrow N[\alpha := A]} \ (\text{P-Ins})$$

$$\frac{M_1 \Longrightarrow N_1 \quad M_2 \Longrightarrow N_2}{M_1 \, M_2 \Longrightarrow N_1 \, N_2} \ (\text{P-App}) \qquad \frac{M \Longrightarrow N}{\blacktriangleright_\alpha M \Longrightarrow \blacktriangleright_\alpha N} \ (\text{P-}\blacktriangleright)$$

$$\frac{M_1 \Longrightarrow N_1 \quad M_2 \Longrightarrow N_2}{(\lambda x : \tau.M_2) \, M_2 \Longrightarrow N_1[x := N_2]} \ (\text{P-Beta}) \qquad \frac{M \Longrightarrow N}{\blacktriangleleft_\alpha M \Longrightarrow \blacktriangleleft_\alpha N} \ (\text{P-}\blacktriangleleft)$$

$$\frac{M \Longrightarrow N}{\Lambda \alpha.M \Longrightarrow \Lambda \alpha.N} \ (\text{P-Gen}) \qquad \frac{M \Longrightarrow N}{\blacktriangleleft_A \blacktriangleright_A M \Longrightarrow N} \ (\text{P-}\blacktriangleleft\,\blacktriangleright)$$

Figure 2: Rules for Parallel Reduction.

x. It is not very standard in the sense that parallel reduction usually does not allow "hidden" redices (that is, redices that appear only after some other reduction steps) to be contracted in one step. We require this definition because a transition variable α can be replaced with a sequence A of transition variables during reduction. If A in (P-$\blacktriangleleft\,\blacktriangleright$) were α, Lemma 3.5 (2) below would not hold any longer.

The following lemma relates the reduction relation and the parallel reduction relation.

Lemma 3.4. $(\longrightarrow) \subseteq (\Longrightarrow) \subseteq (\longrightarrow^*)$

Proof. $(\longrightarrow) \subseteq (\Longrightarrow)$ can be shown by induction on the derivation $M \longrightarrow N$. We can prove $(\Longrightarrow) \subseteq (\longrightarrow^*)$ by induction on the structure of $M \Longrightarrow N$. □

Thanks to this lemma, we know that confluence of $\longrightarrow$ is equivalent to confluence of $\Longrightarrow$. We prove confluence of $\Longrightarrow$ by showing that $\Longrightarrow$ enjoys the diamond property. The following properties of parallel reduction are useful.

Lemma 3.5.
(1) If $M_1 \Longrightarrow N_1$ and $M_2 \Longrightarrow N_2$, then $M_1[x := M_2] \Longrightarrow N_1[x := N_2]$.
(2) If $M \Longrightarrow N$, then $M[\alpha := A] \Longrightarrow N[\alpha := A]$.

Proof. Easy induction on the structure of the derivation $M_1 \Longrightarrow N_1$ and $M \Longrightarrow N$, respectively. □

Now, we define the notion of complete development and show its key property. The *complete development* $M^\star$ of M is defined by induction as in Figure 3.

Lemma 3.6. If $M \Longrightarrow N$, then $N \Longrightarrow M^\star$.

Proof. By induction on the derivation of $M \Longrightarrow N$ with case analysis on M. We show only interesting cases.
• Case (P-Lam): We have $M = \lambda x : \tau.M_0$ and $N = \lambda x : \tau.N_0$ with $M_0 \Longrightarrow N_0$. By the induction hypothesis, we have $N_0 \Longrightarrow M_0^\star$. So, by applying (P-Lam) rule, we obtain $\lambda x : \tau.N_0 \Longrightarrow \lambda x : \tau.M_0^\star$, as required.

$$
\begin{aligned}
x^\star &= x \\
(\lambda x : \tau.M)^\star &= \lambda x : \tau.M^\star \\
((\lambda x : \tau.M)\,N)^\star &= M^\star[x := N^\star] \\
(M\,N)^\star &= M^\star\,N^\star & \text{(if } M \neq \lambda x : \tau.M') \\
(\blacktriangleright_\alpha M)^\star &= \blacktriangleright_\alpha M^\star \\
(\blacktriangleleft_A \blacktriangleright_A M)^\star &= M^\star \\
(\blacktriangleleft_\alpha M)^\star &= \blacktriangleleft_\alpha M^\star & \text{(if } M \neq \blacktriangleleft_A \blacktriangleright_{A\alpha} M') \\
(\Lambda\alpha.M)^\star &= \Lambda\alpha.M^\star \\
((\Lambda\alpha.M)\,A)^\star &= M^\star[\alpha := A] \\
(M\,A)^\star &= M^\star\,A & \text{(if } M \neq \Lambda\alpha.M')
\end{aligned}
$$

Figure 3: Definition of Complete Development.

- Case (P-TApp): We have $M = M_0\,A$ and $N = N_0\,A$ with $M_0 \implies N_0$. There are two subcases.

Assume $M_0 \neq \Lambda\alpha.M_1$. By the induction hypothesis, we have $N_0 \implies M_0^\star$. By applying (P-TApp) rule, we obtain $N_0\,A \implies M_0^\star\,A$.

Assume $M_0 = \Lambda\alpha.M_1$ for some M_1. By the induction hypothesis, we have $N_0 \implies M_0^\star = \Lambda\alpha.M_1^\star$. By the definition of parallel reduction, we have that $N_0 = \Lambda\alpha.N_1$ for some N_1 and

$$N_1 \implies M_1^\star.$$

So, by applying (P-TIns) rule, we obtain $(\Lambda\alpha.N_1)\,A \implies M_1^\star[\alpha := A] = M^\star$.

- Case (P-Ins): We have $M = (\Lambda\alpha.M_0)\,A$ and $N = N_0[\alpha := A]$ with $M_0 \implies N_0$. By the induction hypothesis, we have $N_0 \implies M_0^\star$. By applying Lemma 3.5, we obtain $N_0[\alpha := A] \implies M_0^\star[\alpha := A]$ □

It is easy to show diamond property of $\implies$ by using Lemma 3.6.

Lemma 3.7. *If $M \implies N_1$ and $M \implies N_2$, then there exists N which satisfies $N_1 \implies N$ and $N_2 \implies N$.*

Proof. Choose $M^\star$ as N and use the previous lemma. □

Theorem 3.8 (Confluence). *If $M \longrightarrow^* N_1$ and $M \longrightarrow^* N_2$, then there exists N such that $N_1 \longrightarrow^* N$ and $N_2 \longrightarrow^* N$.*

Proof. By Lemma 3.4, we have

$$(\longrightarrow) \subseteq (\implies) \subseteq (\longrightarrow^*).$$

So we obtain

$$(\longrightarrow^*) \subseteq (\implies^*) \subseteq (\longrightarrow^*),$$

which implies $(\longrightarrow^*) = (\implies^*)$. Therefore what we should show is confluence of $\implies$, which is an easy consequence of Lemma 3.7. □

3.5. **Annotated Reduction and Time-Ordered Normalization.** We introduce the notion of stages into reduction and prove the property called *time-ordered normalization* [7, 10]. Intuitively, it says that normalization can be done in the increasing order of stages and does not need to 'go back' to earlier stages. In other words, once all redices at some stage are contracted, subsequent reductions never yield a new redex at that stage. To state time-ordered normalization formally, we first introduce the notion of *paths* from one stage to another and a new reduction relation, annotated with paths to represent the stage at which reduction occurs.

A path represents how the stage of a subterm is reached from the stage of a given term. For example, if $\Gamma \vdash^\alpha M$ and $\Gamma' \vdash^{\alpha\beta} N$ for a subterm N of M, then we say the path from (the stage of) M to (that of) N is β. The stage of a subterm may not be able to be expressed by a transition (a sequence of transition variables), however: For example, consider the path from $\blacktriangleleft_\alpha M$ to M. We introduce *formal inverses* α^{-1} to deal with such cases: the path from the stage of $\blacktriangleleft_\alpha M$ to that of M is represented by α^{-1}. Similarly, the path from $\blacktriangleright_\alpha \blacktriangleleft_\beta M$ to M will be $\alpha\beta^{-1}$.

Formally, the set of paths, ranged over by T and U, is the free group generated by the set of transition variables Σ. In other words, a path is a finite sequence $\xi_1\xi_2\ldots\xi_n$, where $\xi_i = \alpha$ or α^{-1}, such that it includes no subsequence of the form $\alpha\alpha^{-1}$ or $\alpha^{-1}\alpha$ for any α. We define $(\alpha^{-1})^{-1} = \alpha$ and

$$(\xi_1 \cdots \xi_n) \cdot (\xi_{n+1} \ldots \xi_m) = \begin{cases} (\xi_1 \cdots \xi_{n-1}) \cdot (\xi_{n+2} \ldots \xi_m) & (\text{if } \xi_n = \xi_{n+1}^{-1}) \\ (\xi_1 \cdots \xi_{n-1}) \cdot (\xi_n\xi_{n+1} \ldots \xi_m) & (\text{if } \xi_n \neq \xi_{n+1}^{-1}). \end{cases}$$

The empty sequence ε is the unit element for the operation $T \cdot U$. We simply write TU for $T \cdot U$. We define $(\xi_1\xi_2\ldots\xi_n)^{-1} = \xi_n^{-1}\xi_{n-1}^{-1}\ldots\xi_1^{-1}$.

We say a path T is *positive* if T does not contain formal inverses, in other words, the canonical form of T is in Σ^*. We can naturally identify the positive paths with transitions and use metavariables A and B for positive paths. We write $T \leq U$ when there exists a positive path A which satisfies $TA = U$. Clearly, $\varepsilon \leq T$ if and only if T is positive.

The *annotated reduction* relation is a triple of the form $M \xrightarrow{T} N$, where M and N are terms and T is a path from the stage of M to that of its redex—more precisely, that of the constructor destructed by the reduction, since the stage of a redex and that of its constructor may be different as in $\blacktriangleright_\alpha$ in redex $\blacktriangleleft_\alpha \blacktriangleright_\alpha M$. The definition of the annotated reduction, presented in Figure 4, is mostly straightforward. For example, α^{-1} is given to $\blacktriangleleft_\alpha \blacktriangleright_\alpha M \longrightarrow M$ (the rule (AR-QUOTE)), because the path to the constructor $\blacktriangleright_\alpha$ is α^{-1}. As for the rule (AR-$\blacktriangleright$), the path from M to the constructor destructed by the reduction is T and the path from $\blacktriangleright_\alpha M$ to M is α, hence the path from $\blacktriangleright_\alpha M$ to the constructor is given by their concatenation αT. The rule (AR-$\blacktriangleleft$) is similar. The rule (AR-GEN) is the most interesting. First of all, α is bound here, so, we cannot propagate T in the premise to the conclusion to prevent α from escaping its scope. We have found that replacing α with ε, which is the earliest possible stage, is a reasonable choice, especially for time-ordered normalization.

Annotated reduction is closely related to reduction defined in the previous section. It is easy to see that $M \longrightarrow N$ if and only if there exists T such that $M \xrightarrow{T} N$. Furthermore, such T is unique.

The next theorem shows that any reduction occurs indeed at a positive stage.

Theorem 3.9. *If $\Gamma \vdash^A M : \tau$ and $M \xrightarrow{T} N$, then $\varepsilon \leq AT$.*

$$\frac{}{(\lambda x:\tau.M)\ N \xrightarrow{\varepsilon} M[x:=N]}\ \text{(AR-Beta)}$$

$$\frac{N \xrightarrow{T} N'}{M\ N \xrightarrow{T} M\ N'}\ \text{(AR-App2)}$$

$$\frac{}{(\Lambda\alpha.M)\ A \xrightarrow{\varepsilon} M[\alpha:=A]}\ \text{(AR-Trans)}$$

$$\frac{M \xrightarrow{T} M'}{\blacktriangleright_\alpha M \xrightarrow{\alpha T} \blacktriangleright_\alpha M'}\ \text{(AR-}\blacktriangleright\text{)}$$

$$\frac{}{\blacktriangleleft_\alpha \blacktriangleright_\alpha M \xrightarrow{\alpha^{-1}} M}\ \text{(AR-Quote)}$$

$$\frac{M \xrightarrow{T} M'}{\blacktriangleleft_\alpha M \xrightarrow{\alpha^{-1}T} \blacktriangleleft_\alpha M'}\ \text{(AR-}\blacktriangleleft\text{)}$$

$$\frac{M \xrightarrow{T} M'}{\lambda x:\tau.M \xrightarrow{T} \lambda x:\tau.M'}\ \text{(AR-Abs)}$$

$$\frac{M \xrightarrow{T} M'}{\Lambda\alpha.M \xrightarrow{T[\alpha:=\varepsilon]} \Lambda\alpha.M}\ \text{(AR-Gen)}$$

$$\frac{M \xrightarrow{T} M'}{M\ N \xrightarrow{T} M'\ N}\ \text{(AR-App1)}$$

$$\frac{M \xrightarrow{T} M'}{M\ A \xrightarrow{T} M'\ A}\ \text{(AR-Ins)}$$

Figure 4: Annotated Reduction.

Proof. Easy induction on the structure of $M \xrightarrow{T} N$. □

We say M is T-*normal* when there are no $U \leq T$ and N such that $M \xrightarrow{U} N$. Then, we can state time-ordered normalization as follows:

Theorem 3.10 (Time Ordered Normalization)**.** *Let M be a typable term. If M is T-normal and $M \longrightarrow^* N$, then N is T-normal.*

Proof. See Appendix A. □

As its corollary, we know that for any reduction to a normal form from a typable term M is "rearranged" according to an increasing order between stages. Moreover, this increasing order can be any total order that respects $\leq$, i.e., includes $\leq$ as a subset.

Corollary 3.11. *Let M be a typable term and $\preceq$ be a total order that respects $\leq$. Then, there is a reduction sequence $M \xrightarrow{T_1} N_1 \xrightarrow{T_2} \cdots \xrightarrow{T_n} N_n$, which satisfies $T_1 \preceq T_2 \preceq \cdots \preceq T_n$ and N_n is a normal form.* □

3.6. **Programming in $\lambda^\triangleright$.** We give an example of programming in $\lambda^\triangleright$. The example is the power function, which is a classical example in multi-stage calculi and partial evaluation. We augment $\lambda^\triangleright$ with integers, Booleans, arithmetic and comparison operators, **if-then-else**, a fixed point operator **fix**, and **let**. In the next section, we will formalize such a language (without **let**) as MiniML$^\triangleright$ and study its evaluation in more detail. For readability, we often omit type annotations and put terms under quotation in shaded boxes.

We start with the ordinary power function without staging.

$$\begin{aligned} &\textbf{let}\ \text{power}_0\colon\ \textbf{int} \to \textbf{int} \to \textbf{int} \\ &\qquad = \textbf{fix}\ f.\ \lambda n.\ \lambda x.\ \textbf{if}\ n=0\ \textbf{then}\ 1\ \textbf{else}\ x*(f\ (n-1)\ x) \end{aligned}$$

Our purpose is to get a code generator $\text{power}_\forall$ that takes the exponent n and returns (closed, hence runnable) code of $\lambda x.x * x * \ldots x * 1$, which computes x^n without recursion. Here, we follow the construction of code generators in the previous work [15, 14].

First, we construct a code manipulator $\mathtt{power}_1 : \mathbf{int} \to \triangleright_\alpha \mathbf{int} \to \triangleright_\alpha \mathbf{int}$, which takes an integer n and a piece of integer code and then outputs a piece of code which concatenates the input code by "$*$" n times. It can be obtained by changing type annotation and introducing quasiquotation.

$$\mathbf{let}\,\mathtt{power}_1 :\, \mathbf{int} \to \triangleright_\alpha \mathbf{int} \to \triangleright_\alpha \mathbf{int}$$
$$= \mathbf{fix}\, f.\; \lambda n.\; \lambda x \colon \triangleright_\alpha \mathbf{int}.$$
$$\mathbf{if}\; n = 0 \;\mathbf{then}\; (\blacktriangleright_\alpha 1)\; \mathbf{else}\; \blacktriangleright_\alpha ((\blacktriangleleft_\alpha x) * (\blacktriangleleft_\alpha f\,(n-1)\,x))$$

Then, from $\mathtt{power}_1$, we can construct a code generator $\mathtt{power}_\alpha$ of type $\mathbf{int} \to \triangleright_\alpha(\mathbf{int} \to \mathbf{int})$, which means it takes an integer and returns code of a function.

$$\mathbf{let}\,\mathtt{power}_\alpha :\, \mathbf{int} \to \triangleright_\alpha(\mathbf{int} \to \mathbf{int})$$
$$= \lambda n.\; \blacktriangleright_\alpha \lambda x \colon \mathbf{int}.\; \blacktriangleleft_\alpha (\mathtt{power}_1\, n\, (\blacktriangleright_\alpha x))$$

It indeed behaves as a code generator: for example, $\mathtt{power}_\alpha\, 3$ would evaluate to the code value $\blacktriangleright_\alpha \lambda x \colon \mathbf{int}\,.x * (x * (x * 1))$.

This construction is independent of the choice of the stage α. So, by abstracting α at appropriate places in $\mathtt{power}_1$ and $\mathtt{power}_\alpha$, we can obtain the desired code generator, whose return type is a closed code type $\forall \gamma. \triangleright_\gamma (\mathbf{int} \to \mathbf{int})$.

$$\mathbf{let}\,\mathtt{power}_2 :\, \forall\beta.\, \mathbf{int} \to \triangleright_\beta \mathbf{int} \to \triangleright_\beta \mathbf{int}$$
$$= \Lambda\beta.\, \mathbf{fix}\, f.\; \lambda n.\; \lambda x \colon \triangleright_\beta \mathbf{int}.$$
$$\mathbf{if}\; n = 0 \;\mathbf{then}\; (\blacktriangleright_\beta 1)\; \mathbf{else}\; \blacktriangleright_\beta ((\blacktriangleleft_\beta x) * (\blacktriangleleft_\beta f\,(n-1)\,x))$$
$$\mathbf{let}\,\mathtt{power}_\forall :\, \mathbf{int} \to \forall\gamma.\; \triangleright_\gamma (\mathbf{int} \to \mathbf{int})$$
$$= \lambda n.\, \Lambda\gamma.\, \blacktriangleright_\gamma \lambda x \colon \mathbf{int}.\; \blacktriangleleft_\gamma (\mathtt{power}_2\, \gamma\, n\, (\blacktriangleright_\gamma x))$$

The output from $\mathtt{power}_\forall$ is usable in any stage. For example, if we want code of a cube function at the later stage, say A, then we write $\mathtt{power}_\forall\, 3\, A$. In particular, when A is the empty sequence ε, $\mathtt{power}_\forall\, 3\, \varepsilon : \mathbf{int} \to \mathbf{int}$ evaluates to a function closure which computes $x * x * x * 1$ from the input x. The former corresponds to CSP (of closed code) and the latter to **run**.

4. MiniML$^\triangleright$

We extend $\lambda^\triangleright$ and define an ML-like functional language MiniML$^\triangleright$, which has, in addition to the features of $\lambda^\triangleright$, integers, arithmetic and comparison operations, Booleans, conditional expressions, and the (call-by-value) fixed-point combinator **fix**. We define the type system and big-step evaluation semantics for MiniML$^\triangleright$ and prove type soundness. In this semantics, bindings of transition variables have to be maintained at run time. So, we investigate a fragment of MiniML$^\triangleright$, in which programs can be executed by mostly forgetting information on transition variables. We give another type system, which identifies such a fragment, and erasure translation, which removes transitions from terms, and alternative evaluation semantics for erased terms. Then, we prove the erasure property, which says program executions before and after erasure agree.

4.1. **Syntax and Type System.** The syntax of types and terms of MiniML$^{\triangleright}$ is defined as follows, where n and bv are metavariables ranging over integers and Boolean constants **true** and **false**.

$$
\begin{array}{ll}
\textit{Types} & \tau ::= \mathbf{int} \mid \mathbf{bool} \mid \tau \to \tau \mid \triangleright_\alpha \tau \mid \forall \alpha.\tau \\
\textit{Terms} & M ::= x \mid n \mid bv \mid M = M \mid M + M \mid M - M \mid M * M \\
& \quad\mid \mathbf{if}\ M\ \mathbf{then}\ M\ \mathbf{else}\ M \mid \mathbf{fix}\ f : \tau \to \sigma.M \mid \lambda x : \tau.M \mid MM \\
& \quad\mid \blacktriangleright_\alpha M \mid \blacktriangleleft_\alpha M \mid \Lambda \alpha.M \mid M\ A
\end{array}
$$

The type system is given as a straightforward extension of that of $\lambda^{\triangleright}$. We show typing rules for the additional constructs.

$$
\frac{n \in \mathbb{Z}}{\Gamma \vdash^A n : \mathbf{int}}\ (\textsc{IntC}) \qquad \frac{bv \in \{\mathbf{true}, \mathbf{false}\}}{\Gamma \vdash^A bv : \mathbf{bool}}\ (\textsc{BoolC})
$$

$$
\frac{\Gamma \vdash^A M : \mathbf{int} \qquad \Gamma \vdash^A N : \mathbf{int}}{\Gamma \vdash^A M = N : \mathbf{bool}}\ (\textsc{Eq})
$$

$$
\frac{\Gamma \vdash^A M : \mathbf{int} \qquad \Gamma \vdash^A N : \mathbf{int} \qquad \Diamond \in \{+, -, *\}}{\Gamma \vdash^A M \Diamond N : \mathbf{int}}\ (\textsc{IntOp})
$$

$$
\frac{\Gamma \vdash^A M : \mathbf{bool} \qquad \Gamma \vdash^A N_1 : \tau \qquad \Gamma \vdash^A N_2 : \tau}{\Gamma \vdash^A \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2}\ (\textsc{If})
$$

$$
\frac{\Gamma, f : \tau \to \sigma @ A \vdash^A M : \tau \to \sigma}{\Gamma \vdash^A \mathbf{fix}\ f : \tau \to \sigma.M : \tau \to \sigma}\ (\textsc{Fix})
$$

We use the same notations for term substitution $M[x := N]$, and transition substitution $\tau[\alpha := A]$ and $M[\alpha := A]$ and other auxiliary notions, which can be similarly defined. It is easy to prove that MiniML$^{\triangleright}$ also enjoys Substitution Lemma.

Lemma 4.1 (Substitution Lemma)**.**
(1) If $\Gamma, x : \sigma @ B \vdash^A M : \tau$ and $\Gamma \vdash^B N : \sigma$, then $\Gamma \vdash^A M[x := N] : \tau$.
(2) If $\Gamma \vdash^A M : \tau$, then, $\Gamma[\alpha := B] \vdash^{A[\alpha:=B]} M[\alpha := B] : \tau[\alpha := B]$.

Proof. The proof is essentially the same as that of Substitution Lemma for $\lambda^{\triangleright}$ (Lemma 3.1). □

4.2. **Evaluation and Type Soundness.** Now, we give a big-step semantics and prove that the execution of a well-typed program is properly divided into stages. The judgment has the form $\vdash^A M \Downarrow R$, read "evaluating term M at stage A yields result R," where R is either **err**, which stands for a run-time error, or a value v, defined below. *Values* are given via a family of sets V^A indexed by transitions, that is, stages. The family V^A is defined by the following grammar:

$$
\begin{array}{ll}
v^\varepsilon \in V^\varepsilon & ::= n \mid \mathbf{true} \mid \mathbf{false} \mid \lambda x : \tau.M \mid \blacktriangleright_\alpha V^\alpha \mid \Lambda \alpha.V^\varepsilon \\
v^A \in V^A\ (A \neq \varepsilon) & ::= x \mid n \mid \mathbf{true} \mid \mathbf{false} \mid \lambda x : \tau.V^A \mid \mathbf{fix}\ f : \tau \to \sigma.V^A \\
& \quad \mid V^A V^A \mid \blacktriangleright_\alpha V^{A\alpha} \mid \Lambda \alpha.V^A \mid V^A B \\
& \quad \mid \blacktriangleleft_\alpha V^{A'} \quad (\text{if } A'\alpha = A \text{ and } A' \neq \varepsilon)
\end{array}
$$

The index A represents the current stage in which a value is typed. So, the index changes under quoting and unquoting. Note that a value at a higher stage (that is, under quotation)

include free variables, applications and instantiation since computation is suspended. For example, $x\,y \in V^\alpha$ and so $\blacktriangleright_\alpha x\,y \in V^\varepsilon$.

Figure 5 shows the evaluation rules. Notice that metavariables M or N for terms (not values) are used on the right side of $\Downarrow$, since it is not immediately clear that a result is really a value of a proper form (or **err**)—we will prove such a property as a theorem. The evaluation is left-to-right and call-by-value. The rules in Figure 5(1) are for ordinary evaluation. The rule for $\blacktriangleleft_\alpha M$ means that quote is canceled by unquote; since the resulting term M' belongs to the stage α (inside quotation), α is attached to the conclusion. As seen in the rule for $\Lambda\alpha.M$, Λ does *not* delay the evaluation of the body. The rule about instantiation of a transition abstraction is straightforward. The rules for stages later than ε, which are in Figure 5(2), are all similar: since the term to be evaluated is inside quotation, each term constructor is left as it is and only subterms of stage ε will be evaluated. We also need rules for handling erroneous terms, such as:

$$\frac{\vdash^\varepsilon M \Downarrow M' \qquad M' \notin \mathbb{Z} \qquad \Diamond \in \{+,-,*,=\}}{\vdash^\varepsilon M \Diamond N \Downarrow \mathbf{err}} \qquad \frac{\vdash^A M \Downarrow \mathbf{err} \qquad \Diamond \in \{+,-,*,=\}}{\vdash^A M \Diamond N \Downarrow \mathbf{err}}$$

They are shown in Appendix B.

We show a few properties of the big-step semantics. The first theorem says that evaluation is deterministic.

Theorem 4.2. *If $\vdash^A M \Downarrow R$ and $\vdash^A M \Downarrow R'$, then $R = R'$.*

Proof. By straightforward induction on the derivation of $\vdash^A M \Downarrow R$. $\qquad\square$

The second theorem below says that, unless the result is **err**, the result must be a value even though the rules do not say it is the case.

Theorem 4.3. *Suppose $\vdash^A M \Downarrow R$. Then, either $R = \mathbf{err}$ or $R \in V^A$.*

Proof. By easy induction on the derivation $\vdash^A M \Downarrow R$. $\qquad\square$

The last property is type soundness and its corollary that if a well-typed program of a code type yields a result, then the result is a quoted term, whose body is also typable at stage ε. Unlike a usual setting where only closed terms are considered programs, free variables at non-ε stages are considered symbols and do not cause unbound variable errors in MiniML$^\triangleright$, so we relax the notion of programs to include terms that contain such symbolic variables. We say that Γ is ε-*free* if it satisfies $A \neq \varepsilon$ for any $x : \tau@A \in \Gamma$; then, a program is a term which is typed under an ε-free environment. In the statement of Type Soundness Theorem, we also use the notation Γ^{-A}, defined by: $\Gamma^{-A} = \{x : \tau@B \mid x : \tau@AB \in \Gamma\}$.

Theorem 4.4 (Type Soundness)**.** *If Γ is ε-free and $\Gamma \vdash^\varepsilon M : \tau$ and $\vdash^\varepsilon M \Downarrow R$, then $R = v$ and $v \in V^\varepsilon$ for some v and $\Gamma \vdash^\varepsilon v : \tau$. In particular, if $\tau = \triangleright_\alpha\tau_0$, then $v = \blacktriangleright_\alpha N$ and $\Gamma^{-\alpha} \vdash^\varepsilon N : \tau_0$.*

Proof. See Section C.1. $\qquad\square$

$$\frac{\vdash^\varepsilon M \Downarrow m \qquad \vdash^\varepsilon N \Downarrow n \qquad m = n}{\vdash^\varepsilon M = N \Downarrow \mathbf{true}} \qquad \frac{\vdash^\varepsilon M \Downarrow m \qquad \vdash^\varepsilon N \Downarrow n \qquad m \neq n}{\vdash^\varepsilon M = N \Downarrow \mathbf{false}}$$

$$\frac{\vdash^\varepsilon M \Downarrow m \qquad \vdash^\varepsilon N \Downarrow n \qquad k = m \Diamond n \ (\text{where } \Diamond \in \{+, -, *\})}{\vdash^\varepsilon M \Diamond N \Downarrow k}$$

$$\frac{\vdash^\varepsilon M \Downarrow \mathbf{true} \qquad \vdash^\varepsilon N_1 \Downarrow N_1'}{\vdash^\varepsilon \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2 \Downarrow N_1'} \qquad \frac{\vdash^\varepsilon M \Downarrow \mathbf{false} \qquad \vdash^\varepsilon N_2 \Downarrow N_2'}{\vdash^\varepsilon \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2 \Downarrow N_2'}$$

$$\frac{}{\vdash^\varepsilon \lambda x : \tau.M \Downarrow \lambda x : \tau.M} \qquad \frac{\vdash^\varepsilon M \Downarrow \lambda x : \tau.M' \qquad \vdash^\varepsilon N \Downarrow N' \qquad \vdash^\varepsilon M'[x := N'] \Downarrow M''}{\vdash^\varepsilon M\ N \Downarrow M''}$$

$$\frac{\vdash^\alpha M \Downarrow M'}{\vdash^\varepsilon \blacktriangleright_\alpha M \Downarrow \blacktriangleright_\alpha M'} \qquad \frac{\vdash^\varepsilon M \Downarrow \blacktriangleright_\alpha M'}{\vdash^\alpha \blacktriangleleft_\alpha M \Downarrow M'} \qquad \frac{\vdash^\varepsilon M \Downarrow M'}{\vdash^\varepsilon \Lambda\alpha.M \Downarrow \Lambda\alpha.M'}$$

$$\frac{\vdash^\varepsilon M \Downarrow \Lambda\alpha.M' \qquad \vdash^\varepsilon M'[\alpha := B] \Downarrow M''}{\vdash^\varepsilon M\ B \Downarrow M''} \qquad \frac{c \in \mathbb{Z} \cup \{\mathbf{true}, \mathbf{false}\}}{\vdash^\varepsilon c \Downarrow c}$$

$$\frac{\vdash^\varepsilon M[f := \mathbf{fix}\ f : \tau \to \sigma.M] \Downarrow M'}{\vdash^\varepsilon \mathbf{fix}\ f : \tau \to \sigma.M \Downarrow M'}$$

(1) Rules for ordinary evaluation.

$$\frac{\vdash^A M \Downarrow M' \qquad \vdash^A N \Downarrow N' \qquad \Diamond \in \{+, -, *, =\}}{\vdash^A M \Diamond N \Downarrow M' \Diamond N'}$$

$$\frac{\vdash^A M \Downarrow M' \qquad \vdash^A N_1 \Downarrow N_1' \qquad \vdash^A N_2 \Downarrow N_2'}{\vdash^A \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2 \Downarrow \mathbf{if}\ M'\ \mathbf{then}\ N_1'\ \mathbf{else}\ N_2'}$$

$$\frac{}{\vdash^A x \Downarrow x} \qquad \frac{\vdash^A M \Downarrow M'}{\vdash^A \lambda x : \tau.M \Downarrow \lambda x : \tau.M'} \qquad \frac{\vdash^A M \Downarrow M' \qquad \vdash^A N \Downarrow N'}{\vdash^A M\ N \Downarrow M'\ N'}$$

$$\frac{\vdash^{A\alpha} M \Downarrow M'}{\vdash^A \blacktriangleright_\alpha M \Downarrow \blacktriangleright_\alpha M'} \qquad \frac{\vdash^A M \Downarrow M'}{\vdash^{A\alpha} \blacktriangleleft_\alpha M \Downarrow \blacktriangleleft_\alpha M'} \qquad \frac{\vdash^A M \Downarrow M'}{\vdash^A M\ B \Downarrow M'\ B}$$

$$\frac{\vdash^A M \Downarrow M'}{\vdash^A \Lambda\alpha.M \Downarrow \Lambda\alpha.M'} \qquad \frac{c \in \mathbb{Z} \cup \{\mathbf{true}, \mathbf{false}\}}{\vdash^A c \Downarrow c}$$

$$\frac{\vdash^A M \Downarrow M'}{\vdash^A \mathbf{fix}\ f : \tau \to \sigma.M \Downarrow \mathbf{fix}\ f : \tau \to \sigma.M'}$$

(2) Rules for evaluation inside quotation. Here, $A \neq \varepsilon$.

Figure 5: Big-Step Semantics of MiniML$^\triangleright$.

4.3. **Staged Transition Variables for Erasure Property.** The evaluation of MiniML$^\triangleright$ introduced above relies on the annotation of transition variables. For example, consider two terms

$$
\begin{aligned}
M_1 &= (\Lambda\alpha.\blacktriangleright_\alpha(\lambda x.1)(\blacktriangleright_\beta((\mathbf{fix}\ f.f)2)))\ \varepsilon \\
M_2 &= (\Lambda\alpha.\blacktriangleright_\alpha(\lambda x.1)(\blacktriangleright_\alpha((\mathbf{fix}\ f.f)2)))\ \varepsilon.
\end{aligned}
$$

The only difference is the annotation on $\blacktriangleright((\mathbf{fix}\ f.f)2)$, but $\vdash^\varepsilon M_1 \Downarrow 1$ whereas there is no term N such that $\vdash^\varepsilon M_2 \Downarrow N$. In other words, the evaluation of M_1 terminates but that of M_2 diverges. Therefore, we must record how transition variables are bound to transitions during evaluation.

From the implementation point of view, it is desirable that evaluation is insensitive to the annotation as much as possible to avoid overhead. In λ^α [9], environment classifiers can be regarded as completely static citizens so that the evaluation does not require them, although the authors do not explicitly state it. The property that the evaluation goes well even if we erase the annotations is called *erasure property*. The previous example shows that the erasure property does *not* hold for MiniML$^\triangleright$. Since the argument B, especially, its length, in an instantiation $M\ B$ is significant at run time, we cannot erase transitions completely. So, we consider a slightly weaker notion of erasure, which removes transition variables only from $\blacktriangleright, \blacktriangleleft$ and Λ and replaces the transition B in $M\ B$ with its length. The goal of this section is to find a practically meaningful subset of MiniML$^\triangleright$, which enjoys the erasure property under the translation sketched above.

The reasons why the erasure property is broken are (1) Λ-bound transition variables are used "too far" from the binder, as is the case in M_2 and (2) the "depth" of quoting $\blacktriangleright_\alpha$ can be changed by using instantiation with a transition, whose length is not 1. In the case of M_2, there is an occurrence of transition variable α far from the binder and α is instantiated by ε, whose length is 0. So, to ensure the erasure property, it is enough to prevent both (1) and (2) from holding at once, in other words, to guarantee that Λ-bound transition variables occur near the binder or to restrict instantiations to only transitions of length 1.

Based on this observation, we will introduce two instantiation rules. The first rule is for instantiation of transition variables used only near the binder. We can change the depth of quoting by using this rule, but this rule can be applied only in limited situations. The second rule is for instantiation of transition variables by transitions, whose lengths are 1. This rule can be applied to any $\forall$-types, but we cannot change the depth of quoting. We introduce a new term constructor $M\ [\alpha]$ to distinguish from the former.

The first instantiation rule requires some control on the occurrences of transition variables. We enforce one additional restriction, which requires that transition variables be also staged like term variables. This restriction rejects a type with nested occurrences of $\triangleright_\alpha$, such as $\forall\alpha.\triangleright_\alpha\triangleright_\alpha\tau$, whose term would have a distant use of $\blacktriangleright_\alpha$. This restriction is closely related to the distinction between open and closed code types in λ^i [16].

We define a new type system with staged transition variables. We need two changes to deal with the stages for transition variables. First, we introduce environments for transition variables. A *transition environment* is a set of the form $\{\alpha_1@A_1, \ldots, \alpha_n@A_n\}$, where α_i are distinct transition variables. An intuitive meaning of $\alpha@A$ is that the valid occurrence of α is always of the form $A\alpha$. The second change is the annotation for the universal quantifier. The new syntax for universal quantification is $\forall\alpha@A.\tau$, where A is the (positive) path from the current stage to the stage in which α is usable.

$$\frac{\Delta \vdash_s A \qquad b \in \{\mathbf{int}, \mathbf{bool}\}}{\Delta \vdash_s^A b} \text{ (ST-Base)} \qquad\qquad \frac{\Delta \vdash_s^{A\alpha} \tau}{\Delta \vdash_s^A \rhd_\alpha \tau} \text{ (ST-Code)}$$

$$\frac{\Delta \vdash_s^A \tau \qquad \Delta \vdash_s^A \sigma}{\Delta \vdash_s^A \tau \to \sigma} \text{ (ST-Imp)} \qquad\qquad \frac{\Delta, \alpha@AB \vdash_s^A \tau}{\Delta \vdash_s^A \forall\alpha@B.\tau} \text{ (ST-Univ)}$$

Figure 6: The definition of well-formed types under the transition environment Δ.

Next, we define well-formed transitions, transition environments, types, and type environments to ensure every use of a transition variable is valid. We say a transition $A = \alpha_1 \dots \alpha_n$ is well formed under a transition environment Δ if, for any $i \leq n$, $\alpha_i@\alpha_1 \dots \alpha_{i-1} \in \Delta$. We say Δ is well formed if, for any $\alpha@A \in \Delta$, A is well formed under Δ, i.e., all stages where transition variables are declared are well formed. This definition avoids the circular definition of transition variables, e.g., $\alpha@\beta, \beta@\alpha$. We write $\Delta \vdash_s A$ if A is well formed under Δ, and $\vdash_s \Delta$ if Δ is a well-formed transition environment.

The judgment of the form $\Delta \vdash_s^A \tau$ means "type τ is well formed at stage A under Δ", and defined by the rules in Figure 6. The base types $\mathbf{int}$ and $\mathbf{bool}$ are always well formed at any well-formed stage. The rules for $\tau \to \sigma$ and $\rhd_\alpha \tau$ resemble the typing rules (ABS) and (▶), respectively. The type $\forall\alpha@B.\tau$, which binds a new transition variable α, is well formed at A under Δ if τ is well formed under the transition variables environment extended by the new transition variable declaration $\alpha@AB$. Finally, we define well-formedness of type environment Γ under Δ, written $\Delta \vdash_s \Gamma$, by: Γ is well formed under Δ if and only if Δ is well formed and, for any $x : \tau@A \in \Gamma$, τ is well formed at A under Δ (i.e., $\Delta \vdash_s^A \tau$).

Figure 7 shows the typing rules that differ from the previous type system (except the addition of Δ). They have additional premises about well-formedness. The rule (S-Var) requires the well-formedness of environment $\Gamma, x : \tau@A$, which will require well-formedness of the type τ at A and the transition environment Δ. The rules (S-Num) and (S-Bool) require the well-formedness of the environment Γ and the stage A, which ensures the well-formedness of the base types. The typing rule (S-Gen) records the path from the current stage to the stage in which α is usable. This information is used by the rules (S-Ins1) and (S-Ins2). As mentioned above, there are two kinds of transition instantiation rules and corresponding term constructors. The first one (S-Ins1) is computationally meaningful, in other words it may change the depth of quoting, but can be used only in limited situations. The second one (S-Ins2) does not change the depth of quoting, so this is computationally meaningless and we can use anytime. Here, substitution for a transition variable α in $\forall\alpha@A.\tau$ (among other types) is defined as follows:

$$(\forall\alpha@A.\tau)[\beta := B] = \forall\alpha@(A[\beta := B]).(\tau[\beta := B])$$

It is easy to see that $\Gamma; \Delta \vdash_s^A M : \tau$ implies $\Gamma \vdash^A M : \tau$.

Now, we define the syntax for *erased terms*, terms after erasure and the erasure translation from $\lambda^\rhd$ terms to erased terms, and the big-step semantics for erased terms. The syntax of erased terms, ranged over by $M^\flat$, is as follows:

Erased Terms $\quad M^\flat ::= x \mid n \mid b \mid M^\flat = M^\flat \mid M^\flat + M^\flat \mid M^\flat - M^\flat \mid M^\flat * M^\flat$
$\qquad\qquad\qquad | \text{ if } M^\flat \text{ then } M^\flat \text{ else } M^\flat \mid \mathbf{fix} \ f.M^\flat \mid \lambda x.M^\flat \mid M^\flat M^\flat$
$\qquad\qquad\qquad | \ \blacktriangleright M^\flat \mid \ \blacktriangleleft M^\flat \mid \Lambda M^\flat \mid M^\flat [] \mid M^\flat n \ (n \geq 0)$

$$\frac{\Delta \vdash_s \Gamma, x : \tau@A}{\Gamma, x : \tau@A; \Delta \vdash_s^A x : \tau} \text{ (S-Var)}$$

$$\frac{\Delta \vdash_s \Gamma \quad \Delta \vdash_s A}{\Gamma; \Delta \vdash_s^A n : \mathbf{int}} \text{ (S-Num)}$$

$$\frac{\begin{array}{c} \Delta \vdash_s \Gamma \quad \Delta \vdash_s A \\ bv \in \{\mathbf{true}, \mathbf{false}\} \end{array}}{\Gamma; \Delta \vdash_s^A bv : \mathbf{bool}} \text{ (S-Bool)}$$

$$\frac{\begin{array}{c} \alpha \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(\Delta) \\ \Gamma; \Delta, \alpha@AB \vdash_s^A M : \tau \end{array}}{\Gamma; \Delta \vdash_s^A \Lambda\alpha.M : \forall\alpha@B.\tau} \text{ (S-Gen)}$$

$$\frac{\begin{array}{c} \Gamma; \Delta \vdash_s^A M : \forall\alpha@\varepsilon.\tau \\ \Delta \vdash_s AB \quad \tau = \rhd_\alpha\sigma \end{array}}{\Gamma; \Delta \vdash_s^A M\ B : \tau[\alpha := B]} \text{ (S-Ins1)}$$

$$\frac{\begin{array}{c} \Gamma; \Delta \vdash_s^A M : \forall\alpha@B.\tau \\ \beta@AB \in \Delta \end{array}}{\Gamma; \Delta \vdash_s^A M\ [\beta] : \tau[\alpha := \beta]} \text{ (S-Ins2)}$$

Figure 7: The typing rules which differ from the previous type system.

The *erasing function* $\flat(\cdot)$ from terms to erased erased terms is defined as follows:

$$
\begin{aligned}
\flat(c) &= c && (c \in \mathbb{Z} \cup \{\mathbf{true}, \mathbf{false}\}) \\
\flat(M \Diamond N) &= \flat(M) \Diamond \flat(N) && (\Diamond \in \{+, -, *, =\}) \\
\flat(\mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2) &= \mathbf{if}\ \flat(M)\ \mathbf{then}\ \flat(N_1)\ \mathbf{else}\ \flat(N_2) \\
\flat(\mathbf{fix}\ f : \tau \to \sigma.M) &= \mathbf{fix}\ f.\flat(M) \\
\flat(\lambda x : \tau.M) &= \lambda x.\flat(M) \\
\flat(M\ N) &= \flat(M)\,\flat(N) \\
\flat(\blacktriangleright_\alpha M) &= \blacktriangleright\flat(M) \\
\flat(\blacktriangleleft_\alpha M) &= \blacktriangleleft\flat(M) \\
\flat(\Lambda\alpha.M) &= \Lambda\flat(M) \\
\flat(M[\beta]) &= \flat(M)\,[] \\
\flat(M\ A) &= \flat(M)\,n && (n \text{ is the length of } A).
\end{aligned}
$$

So, $\flat(\blacktriangleright_A M) = \overbrace{\blacktriangleright \ldots \blacktriangleright}^{n}\flat(M)$ and $\flat(\blacktriangleleft_A M) = \overbrace{\blacktriangleleft \ldots \blacktriangleleft}^{n}\flat(M)$ where n is the length of A.

The *erasure semantics* is essentially the same as the ordinary evaluation semantics in Section 4.2, except for the two differences: one is that the stage A of $\vdash^A M \Downarrow N$ is replaced by the natural number n, which is the length of A; and the other is that the rule for $M\ n$ at the stage 0. In this case, M must be evaluated to $\Lambda \blacktriangleright M'$ and $\blacktriangleright$ at its head is duplicated by n times. We show only main rules below.

$$\frac{\vdash_e^0 M^\flat \Downarrow \blacktriangleright N^\flat}{\vdash_e^1 \blacktriangleleft M^\flat \Downarrow N^\flat} \qquad \frac{\vdash_e^{n+1} M^\flat \Downarrow N^\flat}{\vdash_e^n \blacktriangleright M^\flat \Downarrow \blacktriangleright N^\flat} \qquad \frac{\vdash_e^0 M^\flat \Downarrow \Lambda N^\flat}{\vdash_e^0 M^\flat\ [] \Downarrow N^\flat}$$

$$\frac{\vdash_e^0 M^\flat \Downarrow \Lambda \blacktriangleright M_0^\flat \quad \vdash_e^0 \overbrace{\blacktriangleright \blacktriangleright \cdots \blacktriangleright}^{n} M_0^\flat \Downarrow N^\flat}{\vdash_e^0 M^\flat\ n \Downarrow N^\flat}$$

Finally, we state the erasure property: the erasure semantics is equivalent to the semantics with transition variables for terms typed under the new type system.

Theorem 4.5 (Erasure Property). *Suppose Γ is ε-free and $\Gamma; \Delta \vdash_s^\varepsilon M : \tau$. Then,*
(1) if $\vdash^\varepsilon M \Downarrow N$, then $\vdash_e^0 \flat(M) \Downarrow \flat(N)$; and

(2) *if $\vdash_e^0 \flat(M) \Downarrow N^\flat$, then there is some N' such that $\vdash^\varepsilon M \Downarrow N'$ and $N^\flat = \flat(N')$.*

Proof. See Appendix C.2.   □

We believe that the calculus with this new type system does not lose much expressiveness for practical use; in fact, the example of power functions in Section 3.6 can be typed with the new type system.

5. Kripke Semantics for $\lambda^\triangleright$ and Logical Completeness

In this section, we formalize the Kripke semantics discussed in Section 2 and prove completeness of a classical version of the proof system obtained from $\lambda^\triangleright$ to justify the informal interpretation of types in $\lambda^\triangleright$ as formulae of a logic. We augment the set of propositions (namely types) with falsity and the proof rules with double negation elimination. It is left for future work to study the semantics of the intuitionistic version, of which recent work on Kripke semantics for intuitionistic LTL [18, 19] can be a basis.

First, we (re)define the set of propositions and the natural deduction proof system. Then, we proceed to the formal definition of the Kripke semantics and prove soundness and completeness of the proof system. Finally, we examine another rule for the double negation elimination.

5.1. **Natural Deduction.** The set $\Phi_\perp$, ranged over by ϕ and ψ, of *propositions* is given by the grammar for Φ extended with a new constant $\perp$.

The natural deduction system can be obtained by forgetting variables and terms in the typing rules. We add the following new rule, which is the ordinary double negation elimination rule, adapted for this setting:

$$\frac{\Gamma, (\phi \to \perp)@A \vdash^B \perp}{\Gamma \vdash^A \phi} \; (\perp\text{-E}) \quad .$$

5.2. **Kripke Semantics and Completeness.** As mentioned in Section 2, the Kripke semantics for this logic is based on a functional transition system $\mathcal{T} = (S, L, \{\xrightarrow{a} \mid a \in L\})$ where S is the (non-empty) countable set of states, L is the countable set of labels, and $\xrightarrow{a} \in S \to S$ for each label $a \in L$. We write $s \xrightarrow{a_1 \cdots a_n} s'$ if there exist $s_1, \ldots, s_{n-1}$ such that $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s'$. Actually, given $s, a_1, \ldots, a_n$, s' always exists in this setting because $\xrightarrow{a_i}$ is a total function for all $1 \le i \le n$.

To interpret a proposition, we need two *valuations*, one for propositional variables and the other for transition variables. The former is a total function $v \in S \times \text{PV} \to \{0, 1\}$; the latter is a total function $\rho \in \Sigma \to L^*$, where L^* is the set of all finite sequences of labels. Then, we define the *satisfaction relation* $\mathcal{T}, v, \rho; s \Vdash \phi$, where $s \in S$ is a state, as follows:

$$
\begin{array}{lll}
\mathcal{T}, v, \rho; s \Vdash p & \text{iff} & v(s, p) = 1 \\
\mathcal{T}, v, \rho; s \Vdash \perp & & \text{never occurs} \\
\mathcal{T}, v, \rho; s \Vdash \phi \to \psi & \text{iff} & \mathcal{T}, v, \rho; s \nVdash \phi \text{ or } \mathcal{T}, v, \rho; s \Vdash \psi \\
\mathcal{T}, v, \rho; s \Vdash \triangleright_\alpha \phi & \text{iff} & \mathcal{T}, v, \rho; s' \Vdash \phi \text{ where } s \xrightarrow{\rho(\alpha)} s' \\
\mathcal{T}, v, \rho; s \Vdash \forall \alpha. \phi & \text{iff} & \text{for all } A \in L^*, \; \mathcal{T}, v, \rho[A/\alpha]; s \Vdash \phi
\end{array}
$$

Here, $\rho[A/\alpha]$ is defined by: $\rho[A/\alpha](\alpha) = A$ and $\rho[A/\alpha](\beta) = \rho(\beta)$ (for $\beta \neq \alpha$). The satisfaction relation is extended pointwise to contexts Γ (possibly infinite sets of pairs of a proposition and a transition[4]) by:

$$\mathcal{T}, v, \rho; s \Vdash \Gamma \text{ iff } \mathcal{T}, v, \rho; s \Vdash \triangleright_A \phi \text{ for all } \phi@A \in \Gamma \ .$$

The local consequence relation $\Gamma \Vdash \phi$ is defined by:

$$\Gamma \Vdash \phi \text{ iff } \mathcal{T}, v, \rho; s \Vdash \Gamma \text{ implies } \mathcal{T}, v, \rho; s \Vdash \phi \text{ for any } \mathcal{T}, v, \rho, s \ .$$

Then, the natural deduction proof system is sound and complete with respect to the local consequence relation. The proof is similar to the one for first-order predicate logic: we use the standard techniques of Skolemization and Herbrand structure.

Theorem 5.1 (Soundness). *If $\Gamma \vdash^\varepsilon \phi$, then $\Gamma \Vdash \phi$.*

Proof. By induction on the derivation of $\Gamma \vdash^\varepsilon \phi$. $\qquad\square$

Theorem 5.2 (Completeness). *If $\Gamma \Vdash \phi$, then $\Gamma \vdash^\varepsilon \phi$.*

Proof. We give a proof sketch; more detailed proofs are found in Appendix D.

We assume $\Gamma \nvdash^\varepsilon \phi$. We construct a transition system $\mathcal{T}$, two valuations v and ρ and a state s such that $\mathcal{T}, v, \rho; s \Vdash \Gamma$ and $\mathcal{T}, v, \rho; s \nVdash \phi$. The construction is similar to the construction of counter models in first-order predicate logic.

First, we prove the following proposition.

$$\Gamma \nvdash^\varepsilon \bot \text{ implies there exists } \mathcal{T}, v, \rho \text{ and } s \text{ such that } \mathcal{T}, v, \rho; s \Vdash \Gamma.$$

We can reduce this proposition to the completeness of quantifier-free logic (the logic without the quantifier over transition variables) by using Skolemization and Herbrand Universes.

Then, we prove this theorem by contraposition. Because $\Gamma \nvdash^\varepsilon \phi$, we have $\Gamma, \phi \to \bot@\varepsilon \nvdash^\varepsilon \bot$. Therefore, there exist $\mathcal{T}$, v, ρ and s which satisfy $\mathcal{T}, v, \rho; s \Vdash \Gamma$ and $\mathcal{T}, v, \rho; s \Vdash \phi \to \bot$. Then, $\mathcal{T}, v, \rho; s \nVdash \phi$ because $\mathcal{T}, v, \rho; s \Vdash \phi \to \bot$. Therefore, $\Gamma \nVdash \phi$. $\qquad\square$

5.3. **Alternative Semantics.** We can give an alternative deduction rule for double negation elimination.

$$\frac{\Gamma, (\phi \to \bot)@A \vdash^A \bot}{\Gamma \vdash^A \phi} \ (\bot\text{-E})$$

The difference is in the stage of the premise. This rule requires that the stage of $\bot$ is equal to the stage of $\phi \to \bot$, but in the rule in Section 5.1 the stage of $\bot$ is arbitrary. This restriction makes the proof system weaker: for example, in this setting $\triangleright_\alpha$ is not self-dual, that means $\neg \triangleright_\alpha \neg \phi \leftrightarrow \triangleright_\alpha \phi$ is not provable (here $\neg \phi$ is an abbreviation of $\phi \to \bot$), while under the previous rules $\triangleright_\alpha$ is self-dual. The difference is equivalent to the axiom $\triangleright_A \bot \leftrightarrow \triangleright_B \bot$ (or a weaker form $\forall \alpha.(\triangleright_\alpha \bot \to \bot)$) in the sense that this system with this axiom is equivalent to the previous proof system. This axiom corresponds to the axiom $\bigcirc \neg A \leftrightarrow \neg \bigcirc A$ in linear-time temporal logic, due to Stirling [13].

---

[4]We allow Γ to be infinite for technical convenience. For an infinite context Γ, we write $\Gamma \vdash^A \phi$ if there exists a finite context $\Gamma' \subseteq \Gamma$ such that $\Gamma' \vdash^A \phi$. The following argument holds if we restrict Γ to be finite because the logic is compact, i.e., Γ is unsatisfiable if and only if there exists a finite subset $\Gamma' \subseteq \Gamma$ such that Γ' is also unsatisfiable. For more detail, see Appendix D.

$$
\begin{array}{rcl}
[\![b]\!] & = & b \\
[\![\tau \to \sigma]\!] & = & [\![\tau]\!] \to [\![\sigma]\!] \\
[\![\bigcirc\tau]\!] & = & \triangleright_\alpha [\![\tau]\!]
\end{array}
$$

$$
\begin{array}{rcl}
[\![x]\!] & = & x \\
[\![\lambda x : \tau.M]\!] & = & \lambda x : [\![\tau]\!].[\![M]\!] \\
[\![M\ N]\!] & = & [\![M]\!]\ [\![N]\!] \\
[\![\mathbf{next}\ M]\!] & = & \blacktriangleright_\alpha [\![M]\!] \\
[\![\mathbf{prev}\ M]\!] & = & \blacktriangleleft_\alpha [\![M]\!]
\end{array}
$$

$$
\begin{array}{rcl}
[\![\cdot]\!] & = & \cdot \\
[\![\Gamma, x : A@n]\!] & = & [\![\Gamma]\!], x : [\![A]\!]@\alpha^n
\end{array}
$$

Figure 8: Embedding from $\lambda^{\bigcirc}$ to $\lambda^{\triangleright}$. α is a fixed transition variable.

There is a corresponding semantics, with respect to which the new proof system is sound and complete. In fact, this is achieved by a minor change: we allow transition functions $\xrightarrow{a}$ to be partial. As a result, there can be no s' such that $s \xrightarrow{\rho(\alpha)} s'$. In this setting, the semantics for $\triangleright_\alpha \phi$ has to be modified. We define $s \overset{a_1 \cdots a_n}{\nrightarrow}$ as there is no s' which satisfies $s \xrightarrow{a_1 \ldots a_n} s'$, and $\mathcal{T}, v, \rho; s \Vdash \triangleright_\alpha \phi$ as follows:

$$
\mathcal{T}, v, \rho; s \Vdash \triangleright_\alpha \phi \quad \text{iff} \quad s \overset{\rho(\alpha)}{\nrightarrow} \text{ or } \mathcal{T}, v, \rho; s' \Vdash \phi \text{ where } s \xrightarrow{\rho(\alpha)} s'
$$

Completeness and soundness are proved similarly.

6. COMPARING WITH OTHER MULTI-STAGE CALCULI

In this section, we will compare $\lambda^{\triangleright}$ with closely related calculi $\lambda^{\bigcirc}$ [7], the Kripke-style modal λ-calculus [8], λ^α [9] and λ^i [16]. The first two calculi are based on Curry-Howard correspondence between multi-stage calculi and modal logics and our work can be considered a generalization of them. In fact, there are embeddings from these two calculi to $\lambda^{\triangleright}$. The calculi λ^α and λ^i are multi-stage calculi with environment classifiers. We discuss several differences among these two calculi and $\lambda^{\triangleright}$. Although it does not seem possible to give (straightforward) embeddings from them, due to the presence of CSP, we will show that an embedding from the CSP-free fragment of λ^i to $\lambda^{\triangleright}$ is possible.

6.1. **Comparing with $\lambda^{\bigcirc}$.** $\lambda^{\bigcirc}$ [7] is a multi-stage calculus corresponding to linear-time temporal logic (LTL). As already mentioned in Section 2, $\lambda^{\bigcirc}$ is obtained by using only one transition variable, say α. Then the modal operator $\bigcirc$ to mean "next" in LTL corresponds to the modal operator $\triangleright_\alpha$ and the stage n in LTL corresponds to α^n and so on. We define an embedding $[\![\cdot]\!]$ from $\lambda^{\bigcirc}$ into $\lambda^{\triangleright}$ in Figure 8. This embedding is essentially the same as that from $\lambda^{\bigcirc}$ calculus into λ^α, given by Taha and Nielsen [9]. The following two theorems show the correctness of the embedding.

$$
\begin{aligned}
[\![b]\!] &= b \\
[\![\tau \to \sigma]\!] &= [\![\tau]\!] \to [\![\sigma]\!] \\
[\![\Box\tau]\!] &= \forall\alpha. \rhd_\alpha [\![\tau]\!]
\end{aligned}
$$

$$
\begin{aligned}
[\![x]\!]^A &= x \\
[\![\lambda x : \tau.M]\!]^A &= \lambda x : [\![\tau]\!].[\![M]\!]^A \\
[\![M\ N]\!]^A &= [\![M]\!]^A\ [\![N]\!]^A \\
[\![\mathbf{box}\ M]\!]^A &= \Lambda\alpha. \blacktriangleright_\alpha [\![M]\!]^{A\alpha} \quad (\text{where } \alpha \notin \mathrm{FTV}(A)) \\
[\![\mathbf{unbox}_n\ M]\!]^A &= \blacktriangleleft_B [\![M]\!]^{A'}\ B \\
&\qquad (\text{where } A = A'B \text{ and the length of } B \text{ is } n) \\
[\![x_1 : \tau_1, \ldots x_n : \tau_n]\!]^A &= x_1 : [\![\tau_1]\!]@A, \ldots, x_n : [\![\tau_n]\!]@A
\end{aligned}
$$

$$
[\![\Gamma_0; \cdots ; \Gamma_n]\!]^{\alpha_1 \cdots \alpha_n} = [\![\Gamma_0]\!]^\varepsilon, \ldots, [\![\Gamma_i]\!]^{\alpha_1 \cdots \alpha_i}, \ldots, [\![\Gamma_n]\!]^{\alpha_1 \cdots \alpha_n}
$$

Figure 9: Embedding from the Kripke-style modal λ-calculus to $\lambda^\rhd$.

Theorem 6.1. *If* $\Gamma \vdash^n M : \tau$ *in* $\lambda^\bigcirc$, *then* $[\![\Gamma]\!] \vdash^{\alpha^n} [\![M]\!] : [\![\tau]\!]$.

Proof. By induction on the type derivation. $\qquad\square$

Theorem 6.2. *Let* M, N *be* $\lambda^\bigcirc$ *terms. Then,* $M \longrightarrow N$ *iff* $[\![M]\!] \longrightarrow [\![N]\!]$.

Proof. By induction on the structure of M. $\qquad\square$

Moreover, by giving a suitable definition of reduction annotated with time for $\lambda^\bigcirc$, we can easily show that the embedding preserves the stage of reduction.

We can construct a reverse mapping, a type- and reduction-preserving embedding from the quantifier-free fragment of $\lambda^\rhd$ to $\lambda^\bigcirc$, by simply forgetting annotations of transition variables. Moreover, the quantifier-free fragment of $\lambda^\rhd$ with only one transition variable is isomorphic to $\lambda^\bigcirc$ in the sense that there is a bijection that preserves typability and reduction.

6.2. **Comparing with Calculus based on S4.** Davies and Pfenning [8, 12] develop calculi that correspond to intuitionistic modal logic S4 (only with $\Box$). The type $\Box\tau$ represents closed code values, which thus can be run or embedded in code of any later stages, as is possible in $\lambda^\rhd$. We compare $\lambda^\rhd$ with one of their calculi (what they call the Kripke-style modal λ-calculus in Section 4 of [8]), in which, there are **box** and **unbox**$_n$ for quoting and unquoting, respectively (see Pfenning and Davies [8] for details). An embedding $[\![\cdot]\!]$ from this calculus into $\lambda^\rhd$ is given in Figure 9.

The following theorems state the correctness of the embedding.

Theorem 6.3. *Consider a sequence A of distinct transition variables of length n. Then* $\Gamma_0; \Gamma_1; \ldots; \Gamma_n \vdash M : \tau$ *implies* $[\![\Gamma_0; \ldots; \Gamma_n]\!]^A \vdash^A [\![M]\!]^A : [\![\tau]\!]$.

Proof. An easy induction on the type derivation $\Gamma_0; \ldots; \Gamma_n \vdash M : \tau$. $\qquad\square$

Theorem 6.4. *Let A be a sequence of distinct transition variables of length n. Provided that $\Gamma_0; \ldots; \Gamma_n \vdash M : \tau$ is derivable in the Kripke-style modal λ-calculus, then $M \longrightarrow_\beta N$ iff $[\![M]\!]^A \longrightarrow [\![N]\!]^A$.*

Proof. By induction on the structure of M. □

Taha and Nielsen have shown a similar embedding from the Kripke-style calculus into λ^α. In their embedding, the translation of **unbox**, which corresponds to the elimination rule for $\Box$, is slightly more involved than ours, since they use **run** and the CSP operator. In our embedding, on the other hand, **unbox** is expressed uniformly by $M\,B$, which corresponds to the elimination rule for $\forall$.

6.3. **Comparing with λ^α and λ^i.** Comparing $\lambda^\triangleright$ with λ^α [9], we can point out two differences: the meaning of **run** and the absence of CSP primitive.

In λ^α, **run** is a primitive, while, in $\lambda^\triangleright$, **run** M is defined as a syntax sugar for $M\,\varepsilon$. The following rules are typing rules for **run** in λ^α and $\lambda^\triangleright$, respectively.

$$\frac{\Gamma \vdash^A M : (\alpha)\langle\tau\rangle^\alpha}{\Gamma \vdash^A \mathbf{run}\ M : (\alpha)\tau} \qquad \frac{\Gamma \vdash^A M : \forall\alpha.\tau}{\Gamma \vdash^A M\varepsilon : \tau[\alpha := \varepsilon]}$$

Aside from the presence of a binder (α), which is not essential, an important difference is how code type constructors are removed in the conclusion. In λ^α, **run** removes only the *outermost* bracket annotated with α, while, in $\lambda^\triangleright$, $M\,\varepsilon$ removes *all* code-type constructor $\triangleright_\alpha$ in τ. This difference in typing rules is also reflected in reductions.

$$\begin{aligned} \mathbf{run}\ (\alpha)\langle v\rangle^\alpha &\longrightarrow (\alpha)v \\ (\Lambda\alpha.v)\,\varepsilon &\longrightarrow v[\alpha := \varepsilon] \end{aligned}$$

(Here we assume that v does not contain the CSP constructor to simplify the argument— See [9] for the complete definition.) So, it does not seem very easy to give an embedding of either direction.

From a practical point of view, we do not think this difference is very significant. It is not clear when one wants to share one environment classifier or transition variable among different stages. If the use of classifiers or transition variables is staged, as we discussed in Section 4.3, then the difference is very little. In fact, the current implementation of MetaOCaml is based on λ^i, which can be considered a subset of both λ^α and $\lambda^\triangleright$ (if CSP is ignored), and this fact shows that the difference is practically insignificant.

As another difference, λ^α allows the CSP constructor % to be applied to any terms to embed the value of the term inside a quotation. It is easy to see that $\phi@\varepsilon \vdash^\alpha \phi$ is not provable in general and so such a universal CSP operator would *not* be expressible in $\lambda^\triangleright$. However, we can support CSP for many types. CSP for values of base types such as integers and Booleans is easy. CSP for function closures is also possible if they do not contain open code in their bodies or environment. We can deal with CSP for *closed code*, i.e., the terms which have types of the form $\forall\alpha.\,\triangleright_\alpha \tau$ with $\alpha \notin \mathrm{FTV}(\tau)$, as syntactic sugar in $\lambda^\triangleright$. The following rules are for CSP in λ^α and for CSP as syntactics sugar in $\lambda^\triangleright$.

$$\frac{\Gamma \vdash^A M : \tau}{\Gamma \vdash^{A\alpha} \%\ M : \tau} \qquad \frac{\Gamma \vdash^A M : \forall\beta.\,\triangleright_\beta \tau \qquad \beta \notin \mathrm{FTV}(\tau)}{\Gamma \vdash^{A\alpha} \Lambda\beta.(\blacktriangleleft_\alpha(M\,\alpha\beta)) : \forall\beta.\,\triangleright_\beta \tau}$$

$$
\begin{aligned}
[\![b]\!] &= b \\
[\![\tau \to \sigma]\!] &= [\![\tau]\!] \to [\![\sigma]\!] \\
[\![\langle\tau\rangle^\alpha]\!] &= \triangleright_\alpha [\![\tau]\!] \\
[\![\langle\tau\rangle]\!] &= \forall\alpha.\, \triangleright_\alpha [\![\tau]\!] \quad (\text{where } \alpha \notin \mathrm{FV}(\tau))
\end{aligned}
$$

$$
\begin{aligned}
[\![x]\!]^A &= x \\
[\![\lambda x.M]\!]^A &= \lambda x : \tau.[\![M]\!]^A \quad (\tau \text{ should be chosen appropriately}) \\
[\![M\ N]\!]^A &= [\![M]\!]^A\, [\![N]\!]^A \\
[\![\langle M\rangle]\!] &= \blacktriangleright_\alpha [\![M]\!] \quad (\alpha \text{ should be chosen appropriately}) \\
[\![\tilde{}\, M]\!] &= \blacktriangleleft_\alpha [\![M]\!] \quad (\alpha \text{ should be chosen appropriately}) \\
[\![\mathbf{run}\ M]\!] &= [\![M]\!]\, \varepsilon \\
[\![\mathbf{open}\ M]\!] &= [\![M]\!]\, \alpha \quad (\alpha \text{ should be chosen appropriately}) \\
[\![\mathbf{close}\ M]\!] &= \Lambda\alpha.[\![M]\!] \quad (\alpha \text{ should be chosen appropriately})
\end{aligned}
$$

$$
[\![x_1 : \tau_1^{A_1}, \ldots x_n : \tau_n^{A_n}]\!] = x_1 : [\![\tau_1]\!]@A_1, \ldots, x_n : [\![\tau_n]\!]@A_n
$$

Figure 10: Embedding from λ^i without CSP primitive to $\lambda^\triangleright$. Precisely speaking, in order to recover τ and α, which appear only on the right hand side, this embedding takes type derivations of λ^i rather than terms as an input.

The only problematic case is CSP for open code and, as mentioned above, functions containing open code, but we think it is rarely needed.

λ^i [16] is developed as a subset of λ^α where type inference is possible (although there are slight differences in syntax and typing). The difference between $\lambda^\triangleright$ and λ^i, is smaller than that between $\lambda^\triangleright$ and λ^α. In fact, we can construct an embedding that preserves typing from λ^i (without the CSP operator) into $\lambda^\triangleright$, by observing that the executable code type $\langle\tau\rangle$ in λ^i corresponds to $\forall\alpha.\, \triangleright_\alpha \tau$ (where $\alpha \notin \mathrm{FTV}(\tau)$) in $\lambda^\triangleright$. Figure 10 shows the complete description of the embedding. Precisely speaking, it takes a type derivation rather than terms: For example, the rule for **open** means

$$
\left[\!\!\left[\frac{\Gamma \vdash^A M : \langle\tau\rangle}{\Gamma \vdash^A \mathbf{open}\ M : \langle\tau\rangle^\alpha}\right]\!\!\right] = [\![\Gamma \vdash^A M : \langle\tau\rangle]\!]\, \alpha,
$$

(note that α does not appear in the term **open** M and it comes from the derivation) and the rule for **close** means

$$
\left[\!\!\left[\frac{\Gamma \vdash^A M : \langle\tau\rangle^\alpha \qquad \alpha \notin \mathrm{FV}(\Gamma, A, \tau)}{\Gamma \vdash^A \mathbf{close}\ M : \langle\tau\rangle}\right]\!\!\right] = \Lambda\alpha.[\![\Gamma \vdash^A M : \langle\tau\rangle^\alpha]\!].
$$

Theorem 6.5. *For any λ^i term M, which does not contain %, if $\Gamma \vdash^A M : \tau$ in λ^i, then $[\![\Gamma]\!] \vdash^A [\![M]\!] : [\![\tau]\!]$.*

Proof. Easy induction on the type derivation. $\qquad\qquad\square$

Preservation of the semantics is hard to discuss precisely. First of all, the semantics of λ^i is not given in [16] in spite of the subtle syntactical differences between λ^α and λ^i. However, as far as we guess, the semantics of λ^i seems very close to the erasure semantics in Section 4.3, and then we expect to have preservation of the semantics.

7. Related Work

Multi-Stage Calculi Based on Modal Logics and Their Extensions. Our work can be considered a generalization of the previous work on the Curry-Howard isomorphism between multi-stage calculi and modal logics [7, 8, 10]. As we have seen in Section 6, there are embeddings from $\lambda^\bigcirc$ and $\lambda^\square$ to $\lambda^\triangleright$.

The restriction of $\lambda^\square$ that all code be closed precludes the definition of a code generator like $\mathtt{power}_\forall$, which generates both efficient and runnable code. Nanevski and Pfenning [21] have extended $\lambda^\square$ with the notion of names, similar to the symbols in Lisp, and remedied the defect of $\lambda^\square$ by allowing newly generated names (not variables) to appear in closed code.

Taha and Sheard [5] added **run** and CSP to $\lambda^\bigcirc$ and developed MetaML, but its type system was not strong enough—**run** may fail at run-time. Then, Moggi, Taha, Benaissa, and Sheard [14] developed the calculus AIM ("An Idealized MetaML"), in which there are types for both open and closed code; it was then simplified to λ^{BN}, which replaced closed code types with closedness types for closed terms that are not necessarily code. Both calculi are based on categorical models and have sound type systems. The notion of α-closedness in λ^α can be considered a generalization of λ^{BN}'s closed types. In fact, the typing rule for **run** in λ^{BN} is similar to the one in λ^α. Although some of these calculi have sound type systems, it is hard to regard them as logic, mainly due to the presence of CSP, which delays the stage of the type judgment to any later stage, and the typing rule for **run** (as discussed in Section 6.3).

More recently, Yuse and Igarashi have proposed the calculus $\lambda^{\bigcirc\square}$ [10] by combining $\lambda^\bigcirc$ and $\lambda^\square$, while maintaining the Curry-Howard isomorphism. The main idea was to consider LTL with modalities "always" ($\square$) and "next" ($\bigcirc$), which represent closed and open code types, respectively. It is similar to AIM in this respect. Although $\lambda^{\bigcirc\square}$ is based on logic, it cannot be embedded into $\lambda^\triangleright$ simply by combining the two embeddings above. In fact, in $\lambda^{\bigcirc\square}$, both directions of $\square \bigcirc \tau \leftrightarrow \bigcirc\square\tau$ are provable, whereas neither direction of $(\forall\alpha. \triangleright_\alpha \triangleright_\beta \tau) \leftrightarrow \triangleright_\beta \forall\alpha. \triangleright_\alpha \tau$ is provable in $\lambda^\triangleright$. However, in $\lambda^{\bigcirc\square}$ it seems impossible to program a code generator like $\mathtt{power}_\forall$, which generates specialized code used at any stage; the best possible one presented can generate specialized code used only at any *later* stage, so running the specialized code is not possible.

It is considered not easy to develop a sound type system for staging constructs *with side effects*. Calcagno, Moggi, and Sheard developed a sound type system for a multi-stage calculus with references using closed types [22]. It is interesting to study whether their closedness condition can be relaxed by using α-closedness.

Other Multi-Stage Calculi. Kim, Yi, and Calcagno's λ^{poly}_{open} [11] is a rather powerful multi-stage calculus with open and closed code fragments, intentionally variable-capturing substitution, lifting values into code, and even references and ML-style type inference. The type structure of λ^{poly}_{open} is rather different: Since variables in code values can escape from

its initial binder's scope and get captured by other binders, a code type records the names of free variables and their types, as well as the type of the whole code. It is not clear how (a pure fragment of) the calculus can be related to other foundational calculi; possible directions may be to use the calculus of contexts [23] by Sato, Sakurai, and Kameyama, and the contextual modal type theory by Nanevski, Pfenning, and Pientka [24].

Viera and Pardo [25] have proposed a multi-stage language with intensional code analysis, that is, pattern matching on code. The language requires typechecking at run-time.

More recently, Kameyama, Kiselyov, and Shan [26] have developed an extension of (a two-level version of) $\lambda^{\bigcirc}$ with control operators shift/reset [27], which enable an interesting pattern of code generation such as let-insertion [28] in the direct style. It will be interesting to investigate how this calculus extends to dynamic code execution (i.e., **run**).

Modal Logics. As we discussed above, the $\Box$-fragment of modal logic, the $\bigcirc$-fragment of LTL can be embedded into our logic, and the $\Box\bigcirc$-fragment of LTL and our logic is incomparable.

Our logic has three characteristic features: (1) it is multi-modal, (2) it has universal quantification over modalities and (3) modal operators are "relative", meaning their semantics depends on the possible world at which they are interpreted. Most other logics do not have all of these features.

Dynamic logic [29] is a multi-modal logic for reasoning about programs. Its modal operators are $[\alpha]$ for each program α, and $[\alpha]\phi$ means "when α halts, ϕ must stand after execution of α from the current state". Dynamic logic is multi-modal and its modal operators are "relative", but does not have quantification over programs. Therefore, there is no formula in Dynamic logic which would correspond to $\forall\alpha.\triangleright_\alpha\triangleright_\alpha\phi$. There is, however, a formula which is expressive in Dynamic logic but not in our logic: e.g., a Dynamic logic formula $[\alpha^*]\phi$, which means intuitively $\phi \wedge [\alpha]\phi \wedge [\alpha][\alpha]\phi \wedge \ldots$, cannot be expressed in our logic.

Hybrid logic [30] is a modal logic with a new kind of atomic formula called *nominals*, each of which must be true exactly one state in any model (therefore, a nominal names a state). For each nominal i, $@_i$ is a modal operator and $@_i\phi$ means "ϕ stands at the state denoted by i". Hybrid logic has a universal quantifier over nominals (and another binder $\downarrow$: $\downarrow x.\phi$ means "let x stand for the nominal for the current world, then ϕ stands"). Hybrid logic differs from our logic, in that modal operators $@_i$ indicate worlds directly, hence are not "relative". In Hybrid logic, $@_i@_j\phi \leftrightarrow @_j\phi$, but $\triangleright_\alpha\triangleright_\beta\phi$ and $\triangleright_\beta\phi$ are not equivalent in our logic.

8. Conclusion and Future Work

We have studied a logical aspect of environment classifiers by developing a simply typed multi-stage calculus $\lambda^\triangleright$ with environment classifiers. This calculus corresponds to a multi-modal logic with quantifier over transitions by the Curry-Howard isomorphism and satisfies time-ordered normalization as well as basic properties such as subject reduction, confluence, and strong normalization. The classical proof system is sound and complete with respect to the Kripke semantics. Our calculus simplifies the previous calculus λ^α of environment classifiers by reducing **run** and some uses of CSP to an extension of another construct. We believe our work helps clarify the semantics of environment classifiers.

We have also studied evaluation of (a slight extension of) $\lambda^\triangleright$ and shown staged execution of a program is possible. Also, it is shown that erasure execution is possible for a certain subset, which is close to λ^i, the basis of MetaOCaml.

From a theoretical perspective, it is interesting to study the semantics of the intuitionistic version of the logic, as mentioned earlier, and also the calculus corresponding to the classical version of the logic. It is known that the naive combination of staging constructs and control operators is problematic since bound variables in quotation may escape from its scope by a control operator. We expect that a logical analysis, like the one presented here and Reed and Pfenning [31], will help analyze the problem.

From a practical perspective, one feature missing from $\lambda^\triangleright$ is CSP for all types. As argued in the introduction, we think typical use of CSP is rather limited and so easy to support. Type inference for full $\lambda^\triangleright$ would not be possible for the same reason as λ^α [16]. However, it would be easy to applying type inference for λ^i_{let} [16] to a similar subset of $\lambda^\triangleright$.

References

[1] Jones, N.D., Gomard, C.K., Sestoft, P.: Partial Evaluation and Automatic Program Generation. Prentice-Hall (1993)

[2] Consel, C., Lawall, J.L., Meur, A.F.L.: A tour of Tempo: A program specializer for the C language. Science of Computer Programming **52**(1-3) (2004) 341–370

[3] Wickline, P., Lee, P., Pfenning, F.: Run-time code generation and Modal-ML. In: Proceedings of 1998 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'98), Montreal, Quebec, Canada (1998) 224–235

[4] Poletto, M., Hsieh, W.C., Engler, D.R., Kaashoek, M.F.: 'C and tcc: A language and compiler for dynamic code generation. ACM Transactions on Programming Languages and Systems **21**(2) (1999) 324–369

[5] Taha, W., Sheard, T.: MetaML and multi-stage programming with explicit annotations. Theoretical Computer Science **248** (2000) 211–242

[6] Glück, R., Jørgensen, J.: Efficient multi-level generating extensions for program specialization. In: Proceedings of Programming Languages, Implementations, Logics and Programs (PLILP'95). Volume 982 of LNCS. (1995) 259–278

[7] Davies, R.: A temporal-logic approach to binding-time analysis. In: Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS 1996), IEEE Computer Society Press (1996) 184–195

[8] Davies, R., Pfenning, F.: A modal analysis of staged computation. J. ACM **48**(3) (2001) 555–604

[9] Taha, W., Nielsen, M.F.: Environment classifiers. In: Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '03), New York, NY, USA, ACM (2003) 26–37

[10] Yuse, Y., Igarashi, A.: A modal type system for multi-level generating extensions with persistent code. In: Proceedings of Eighth ACM-SIGPLAN International Symposium on Principles and Practice of Declarative Programming (PPDP'06). (2006) 201–212

[11] Kim, I.S., Yi, K., Calcagno, C.: A polymorphic modal type system for lisp-like multi-staged languages. In: Proceedings of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '06), New York, NY, USA, ACM (2006) 257–268

[12] Davies, R., Pfenning, F.: A modal analysis of staged computation. In: Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '96). (1996) 258–270

[13] Stirling, C.: Modal and temporal logics. In: Handbook of Logic in Computer Science. Volume 2. Oxford University Press (1992) 477–563

[14] Moggi, E., Taha, W., Benaissa, Z.E.A., Sheard, T.: An idealized MetaML: Simpler, and more expressive. In: Proceedings of European Symposium on Programming (ESOP'99). Volume 1576 of LNCS. (1999) 193–207

[15] Benaissa, Z.E.A., Moggi, E., Taha, W., Sheard, T.: Logical modalities and multi-stage programming. In: Proceedings of Workshop on Intuitionistic Modal Logics and Applications (IMLA'99). (1999)

[16] Calcagno, C., Moggi, E., Taha, W.: ML-like inference for classifiers. In: Proceedings of European Symposium on Programming (ESOP '04), Springer-Verlag (2004) 79–93

[17] Plotkin, G., Stirling, C.: A framework for intuitionistic modal logics. In: Proceedings of Theoretical Aspects of Reasoning about Knowledge, Monterey, CA, Morgan Kaufmann (1986)

[18] Kojima, K., Igarashi, A.: On constructive linear-time temporal logic. In: Proceedings of Intuitionistic Modal Logics and Applications Workshop (IMLA '08), Pittsburgh, PA (2008)

[19] Kojima, K., Igarashi, A.: Constructive linear-time temporal logic: Proof systems and Kripke semantics. Information and Computation (2010) Accepted for publication.

[20] Takahashi, M.: Parallel reductions in λ-calculus. Information and Computation **118**(1) (1995) 120–127

[21] Nanevski, A., Pfenning, F.: Staged computation with names and necessity. Journal of Functional Programming **15**(5) (2005) 893–939

[22] Calcagno, C., Moggi, E., Sheard, T.: Closed types for a safe imperative MetaML. Journal of Functional Programming **13**(3) (2003) 545–571

[23] Sato, M., Sakurai, T., Kameyama, Y.: A simply typed context calculus with first-class environments. Journal of Functional and Logic Programming **2002**(4) (2002) 1–41

[24] Nanevski, A., Pfenning, F., Pientka, B.: Contextual modal type theory. ACM Transactions on Computational Logic **9**(3) (2008)

[25] Viera, M., Pardo, A.: A multi-stage language with intensional analysis. In: Proceedings of Fifth International Conference on Generative Programming and Component Engineering (GPCE'06), Portland, OR (2006) 11–20

[26] Kameyama, Y., Kiselyov, O., chieh Shan, C.: Shifting the stage: Staging with delimited control. In: Proceedings of ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'09), Savannah, GA (2009)

[27] Danvy, O., Filinski, A.: Abstracting control. In: Proceedings of ACM Conference on Lisp and Functional Programming. (1990) 151–160

[28] Lawall, J.L., Danvy, O.: Continuation-based partial evaluation. In: Proceedings of ACM Conference on Lisp and Functional Programming. (1994) 227–238

[29] Harel, D., Kozen, D., Tiuryn, J.: Dynamic logic. In Gabbay, D., Guenther, F., eds.: Handbook of Philosophical Logic. Volume 4. 2nd edn. Springer-Verlag (2002) 99–218

[30] Areces, C., ten Cate, B.: Hybrid logics. In Blackburn, P., Wolter, F., van Benthem, J., eds.: Handbook of Modal Logics. Elsevier (2006) 821–868

[31] Reed, J., Pfenning, F.: Intuitionistic letcc via labelled deduction. In: Proceedings of Workshop on Methods for Modalities (M4M). (2007)

Appendix A. Proof of Time Ordered Normalization (Theorem 3.10)

First, we give an inductive characterization of the class of T-normal terms. A judgment of the form $\Delta \vdash \Downarrow^T M$, where Δ is a finite set of transition variables, means "M is T-normal under bound transition variables Δ." We distinguish transition variables bound outside of M since they are interpreted as the empty transition in annotated reduction. We also need an auxiliary judgment of the form $\Delta \vdash \bigtriangledown^T M$, read "$M$ is T-neutral under bound transition variables Δ". We write $T[\Delta := \varepsilon]$ for $T[\alpha_1 := \varepsilon]\ldots[\alpha_n := \varepsilon]$, where $\Delta = \{\alpha_1, \ldots, \alpha_n\}$. The inference rules for these judgments are shown in Figure 11.

We first prove that this proof system characterize T-normal forms as in Lemma A.2, which is obtained as a special case of the lemma below. In what follows, $\mathrm{FTV}(T)$ denotes the set of transition variables in the path T.

Lemma A.1. *Let M be a typable term, T be a path, Δ be a finite set of transition variables. Then, the following two conditions are equivalent:*

(1) *For any term N and path U, if $M \xrightarrow{U} N$, then $U[\Delta := \varepsilon] \not\leq T[\Delta := \varepsilon]$.*
(2) $\Delta \vdash \Downarrow^T M$.

Proof. We show (1) $\implies$ (2) by induction on M.

- Case $M = x$: $\Delta \vdash \Downarrow^T M$ trivially holds.

- Case $M = \blacktriangleright_\alpha M'$: We first show that $M' \xrightarrow{U} N'$ implies $U[\Delta := \varepsilon] \not\leq (\alpha^{-1}T)[\Delta := \varepsilon]$ for any N' and U. Assume $M' \xrightarrow{U} N'$. Then, we have $\blacktriangleright_\alpha M' \xrightarrow{\alpha U} \blacktriangleright_\alpha N'$. By (1), we have $(\alpha U)[\Delta := \varepsilon] \not\leq T[\Delta := \varepsilon]$ and then $U[\Delta := \varepsilon] \not\leq (\alpha^{-1}T)[\Delta := \varepsilon]$. m By the induction hypothesis, we have $\Delta \vdash \Downarrow^{\alpha^{-1}T} M'$, so $\Delta \vdash \Downarrow^T \blacktriangleright_\alpha M'$.

- Case $M = \blacktriangleleft_\alpha M'$: Similarly to the case above, we have $\Delta \vdash \Downarrow^{\alpha T} M'$. What remains to show is $\Delta \vdash \bigtriangledown^{\alpha T} M'$.

Since M is typable, a possible form of M' is a variable, an application, an instantiation, or $\blacktriangleright_\alpha M''$ for some M''. In the first three cases, $\Delta \vdash \bigtriangledown^{\alpha T} M'$ is trivial. If $M = \blacktriangleright_\alpha M''$, then we have $M' \xrightarrow{\alpha^{-1}} M''$. Then, by (1), we have $\alpha^{-1}[\Delta := \varepsilon] \not\leq T[\Delta := \varepsilon]$ and then $\varepsilon \not\leq (\alpha T)[\Delta := \varepsilon]$. By the first rule for $\cdot \vdash \bigtriangledown^{(\cdot)}\cdot$, we have $\Delta \vdash \bigtriangledown^{\alpha T} M'$.

- Case $M = \Lambda\alpha.M'$:

$$\frac{\varepsilon \not\leq T[\Delta := \varepsilon]}{\Delta \vdash \bigtriangledown^T M} \qquad \frac{M = x \text{ or } M_0\,M_1 \text{ or } M_0 A}{\Delta \vdash \bigtriangledown^T M}$$

$$\frac{}{\Delta \vdash \Downarrow^T x} \qquad \frac{\Delta \vdash \Downarrow^T M}{\Delta \vdash \Downarrow^T \lambda x : \tau.M} \qquad \frac{\Delta \vdash \Downarrow^T M \qquad \Delta \vdash \bigtriangledown^T M \qquad \Delta \vdash \Downarrow^T N}{\Delta \vdash \Downarrow^T M\,N}$$

$$\frac{\Delta \vdash \Downarrow^T M}{\Delta \vdash \Downarrow^{\alpha T} \blacktriangleright_\alpha M} \qquad \frac{\Delta \vdash \Downarrow^T M \qquad \Delta \vdash \bigtriangledown^T M}{\Delta \vdash \Downarrow^{\alpha^{-1}T} \blacktriangleleft_\alpha M}$$

$$\frac{\Delta, \alpha \vdash \Downarrow^T M \qquad \alpha \notin \mathrm{FTV}(T) \cup \Delta}{\Delta \vdash \Downarrow^T \Lambda\alpha.M} \qquad \frac{\Delta \vdash \Downarrow^T M \qquad \Delta \vdash \bigtriangledown^T M}{\Delta \vdash \Downarrow^T M\,A}$$

Figure 11: T-neutral terms and T-normal terms.

Assume $M' \xrightarrow{U} N'$ and $\alpha \notin \mathrm{FTV}(T) \cup \Delta$. Then, we have $\Lambda\alpha.M' \xrightarrow{U[\alpha := \varepsilon]} \Lambda\alpha.N'$. By (1), $(U[\alpha := \varepsilon])[\Delta := \varepsilon] \nleq T[\Delta := \varepsilon]$. We have $(U[\alpha := \varepsilon])[\Delta := \varepsilon] = U[\Delta, \alpha := \varepsilon]$ and $T[\Delta, \alpha := \varepsilon] = T[\Delta := \varepsilon]$ because $\alpha \notin \mathrm{FTV}(T)$. So, $U[\Delta, \alpha := \varepsilon] \nleq T[\Delta, \alpha := \varepsilon]$. Then, we have $\Delta, \alpha \vdash \Downarrow^T M'$ by the induction hypothesis. So, $\Delta \vdash \Downarrow^T \Lambda\alpha.M'$ as required.

Other cases are similar.

The proof of $(2) \Rightarrow (1)$ is by easy induction on the structure of the derivation $\Delta \vdash \Downarrow^T M$. We show the case $M = \blacktriangleleft_\alpha M'$ as a representative case.

Assume $\Delta \vdash \Downarrow^T \blacktriangleleft_\alpha M'$. By definition, we have $\Delta \vdash \Downarrow^{\alpha T} M'$ and $\Delta \vdash \bigtriangledown^{\alpha T} M'$. Assume $\blacktriangleleft_\alpha M' \xrightarrow{U} N$. There are two cases.

- Case $\blacktriangleleft_\alpha M' \xrightarrow{U} \blacktriangleleft_\alpha N'$ with $M' \xrightarrow{\alpha U} N'$: Because $\Delta \vdash \Downarrow^{\alpha T} M'$, by induction hypothesis we have $(\alpha U)[\Delta := \varepsilon] \nleq (\alpha T)[\Delta := \varepsilon]$. This implies that $U[\Delta := \varepsilon] \nleq T[\Delta := \varepsilon]$.

- Case $M' = \blacktriangleright_\alpha M''$ and $\blacktriangleleft_\alpha \blacktriangleright_\alpha M'' \xrightarrow{\alpha^{-1}} M''$: Since $\Delta \vdash \bigtriangledown^{\alpha T} \blacktriangleright_\alpha M''$, we have $\varepsilon \nleq (\alpha T)[\Delta := \varepsilon]$. This equation is equivalent to $\alpha^{-1}[\Delta := \varepsilon] \nleq T[\Delta := \varepsilon]$. $\qquad\square$

Lemma A.2. *Suppose M is typable. Then, M is T-normal if and only if $\emptyset \vdash \Downarrow^T M$.*

Proof. Immediate from Lemma A.1. $\qquad\square$

Now, we prove that reduction preserves T-normality and T-neutrality, as in Lemma A.5, from which Theorem 3.10 immediately follows. Before that, we prove that (term and transition) substitution preserves T-normality and T-neutrality under a certain condition.

Lemma A.3. *Suppose $\Gamma, x : \sigma@B \vdash^A M : \tau$ and $\Gamma \vdash^B N : \sigma$ and $B[\Delta := \varepsilon] \nleq (AT)[\Delta := \varepsilon]$. If $\Delta \vdash \Downarrow^T M$ and $\Delta \vdash \Downarrow^{B^{-1}AT} N$, then $\Delta \vdash \Downarrow^T (M[x := N])$. Similarly, if $\Delta \vdash \bigtriangledown^T M$, then $\Delta \vdash \bigtriangledown^T (M[x := N])$.*

Proof. By induction on M. We show only the main cases.

- Case $M = y$: The subcase $x \neq y$ is trivial. Assume $x = y$. Because $\Gamma, x : \sigma@B \vdash^A x : \tau$, it is the case that $A = B$. So $B^{-1}AT = T$ and $\Delta \vdash \Downarrow^T N$. Moreover, we have $\Delta \vdash \bigtriangledown^T N$ because $\varepsilon \nleq T[\Delta := \varepsilon]$, which follows from $B[\Delta := \varepsilon] \nleq (AT)[\Delta := \varepsilon]$ and $B = A$.

- Case $M = \blacktriangleright_\alpha M'$: From the type derivation of M, we have $\tau = \triangleright_\alpha \tau_0$ and $\Gamma, x : \sigma@B \vdash^{A\alpha} M' : \tau_0$. From $\Delta \vdash \Downarrow^T \blacktriangleright_\alpha M'$, we obtain $\Delta \vdash \Downarrow^{\alpha^{-1}T} M'$. Moreover, $B[\Delta := \varepsilon] \nleq (AT)[\Delta := \varepsilon] = ((A\alpha)(\alpha^{-1}T))[\Delta := \varepsilon]$ holds. By the induction hypothesis, we now have $\Delta \vdash \Downarrow^{\alpha^{-1}T} (M'[x := N])$. Therefore $\Delta \vdash \Downarrow^T (\blacktriangleright_\alpha M')[x := N]$.
  If $\Delta \vdash \bigtriangledown^T \blacktriangleright_\alpha M'$, it is the case that $\varepsilon \nleq T[\Delta := \varepsilon]$. Therefore $\Delta \vdash \bigtriangledown^T (\blacktriangleright_\alpha M')[x := N]$.

- Case $M = \Lambda\alpha.M'$: Without loss of generality, we can assume that $\alpha \notin \mathrm{FTV}(A) \cup \mathrm{FTV}(N) \cup \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(T) \cup \mathrm{FTV}(B) \cup \Delta$. From the type derivation of M, we have $\tau = \forall\alpha.\tau_0$ and $\Gamma, x : \sigma@B \vdash^A M' : \tau_0$. From $\Delta \vdash \Downarrow^T M$, we also have $\Delta, \alpha \vdash \Downarrow^T M'$. Because α is fresh and $B[\Delta := \varepsilon] \nleq (AT)[\Delta := \varepsilon]$, we have $B[\Delta, \alpha := \varepsilon] \nleq (AT)[\Delta, \alpha := \varepsilon]$. By the induction hypothesis, we have $\Delta, \alpha \vdash \Downarrow^T M'[x := N]$, which implies $\Delta \vdash \Downarrow^T (\Lambda\alpha.M')[x := N]$.
  If $\Delta \vdash \bigtriangledown^T \Lambda\alpha.M'$, it is the case that $\varepsilon \nleq T[\Delta := \varepsilon]$. Therefore $\Delta \vdash \bigtriangledown^T (\Lambda\alpha.M')[x := N]$. $\qquad\square$

Lemma A.4. *Suppose* $\Gamma \vdash^A M : \tau$ *and* $\alpha \notin \mathrm{FTV}(AT)$.
(1) *If* $\Delta, \alpha \vdash \Downarrow^T M$, *then* $\Delta \vdash \Downarrow^{T[\alpha:=B]} (M[\alpha := B])$; *and*
(2) *if* $\Delta, \alpha \vdash \bigtriangledown^T M$, *then* $\Delta \vdash \bigtriangledown^{T[\alpha:=B]}(M[\alpha := B])$.

Proof. We first show (2) by case analysis on the last rule used to derive $\Delta, \alpha \vdash \bigtriangledown^T M$.

- Case $\Delta, \alpha \vdash \bigtriangledown^T M$ with $\varepsilon \not\leq T[\Delta, \alpha := \varepsilon]$: Because $\alpha \notin \mathrm{FTV}(AT)$ and $\varepsilon \leq A$, we have $A = A'\alpha^n$ and $T = \alpha^{-n}T'$ and $\alpha \notin \mathrm{FTV}(A') \cup \mathrm{FTV}(T')$ for some $n \geq 0$. Then, $\varepsilon \not\leq T[\Delta, \alpha := \varepsilon] = (\alpha^{-n}T')[\Delta, \alpha := \varepsilon] = T'[\Delta := \varepsilon]$. Therefore $B^{-n}[\Delta := \varepsilon] \not\leq (B^{-n}[\Delta := \varepsilon])(T'[\Delta := \varepsilon]) = T[\alpha := B][\Delta := \varepsilon]$. It follows that $\varepsilon \not\leq T[\alpha := B][\Delta := \varepsilon]$ because $B^{-n}[\Delta := \varepsilon] \leq \varepsilon$. So we have $\Delta \vdash \bigtriangledown^{T[\alpha:=B]}(M[\alpha := B])$.

- The other case is easy.

Then, (1) is proved by induction on the derivation of $\Delta, \alpha \vdash \Downarrow^T M$. We show only the main cases.

- Case $\Delta, \alpha \vdash \Downarrow^T N_0 N_1$: Applying the induction hypothesis and (2) to, respectively,

$$\Delta, \alpha \vdash \Downarrow^T N_0 \qquad\qquad \Delta \vdash \Downarrow^{T[\alpha:=B]} (N_0[\alpha := B])$$
$$\Delta, \alpha \vdash \bigtriangledown^T N_0 \qquad \text{results in} \qquad \Delta \vdash \bigtriangledown^{T[\alpha:=B]}(N_0[\alpha := B])$$
$$\Delta, \alpha \vdash \Downarrow^T N_1 \qquad\qquad \Delta \vdash \Downarrow^{T[\alpha:=B]} (N_1[\alpha := B]).$$

So we obtain $\Delta \vdash \Downarrow^{T[\alpha:=B]} (N_0 N_1)[\alpha := B]$.

- Case $\Delta, \alpha \vdash \Downarrow^T \blacktriangleleft_\beta M'$: It is the case that $\Delta, \alpha \vdash \Downarrow^{\beta T} M'$ and $\Delta, \alpha \vdash \bigtriangledown^{\beta T} M'$. Because $\Gamma \vdash^A \blacktriangleleft_\beta M' : \tau$, we have $A = A'\beta$ and $\Gamma \vdash^{A'} M' : \triangleright_\beta \tau$. Since $\alpha \notin \mathrm{FTV}(AT) = \mathrm{FTV}(A'\beta T)$, by applying the induction hypothesis and (2), we have $\Delta \vdash \Downarrow^{(\beta T)[\alpha:=B]} M'[\alpha := B]$ and $\Delta \vdash \bigtriangledown^{(\beta T)[\alpha:=B]}M'[\alpha := B]$. Therefore $\Delta \vdash \Downarrow^{T[\alpha:=B]} (\blacktriangleright_{\beta[\alpha:=B]}(M'[\alpha := B]))$. $\square$

Lemma A.5. *Suppose* $\Gamma \vdash^A M : \tau$ *and* $M \xrightarrow{U} N$ *and* $U[\Delta := \varepsilon] \not\leq T[\Delta := \varepsilon]$. *If* $\Delta \vdash \Downarrow^T M$, *then* $\Delta \vdash \Downarrow^T N$. *Similarly, if* $\Delta \vdash \bigtriangledown^T M$, *then* $\Delta \vdash \bigtriangledown^T N$.

Proof. By induction on the derivation of $M \xrightarrow{U} N$.

- Case $M = (\lambda x : \sigma.P) Q \xrightarrow{\varepsilon} P[x := Q] = N$: It is the case that $U = \varepsilon$ and so $\varepsilon \not\leq T[\Delta := \varepsilon]$. Assume $\Delta \vdash \Downarrow^T M$. Then we have $\Delta \vdash \Downarrow^T P$ and $\Delta \vdash \bigtriangledown^T P$ and $\Delta \vdash \Downarrow^T Q$. From $\varepsilon \not\leq T[\Delta := \varepsilon]$, it follows that $A[\Delta := \varepsilon] \not\leq (AT)[\Delta := \varepsilon]$. By Lemma A.3, we have $\Delta \vdash \Downarrow^T (P[x := Q])$. Moreover, we have $\Delta \vdash \bigtriangledown^T N$ because $U[\Delta := \varepsilon] = \varepsilon \not\leq T[\Delta := \varepsilon]$.

- Case $M = (\Lambda\alpha.P) B \xrightarrow{\varepsilon} P[\alpha := B] = N$: We can assume without loss of generality that $\alpha \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(A) \cup \mathrm{FTV}(T) \cup \Delta$. Assume $\Delta \vdash \Downarrow^T M$. Then we have $\Delta, \alpha \vdash \Downarrow^T P$. Because $\alpha \notin \mathrm{FTV}(AT)$, by applying Lemma A.4, we have $\Delta \vdash \Downarrow^{T[\alpha:=B]} P[\alpha := B]$. Because $\alpha \notin \mathrm{FTV}(T)$, $T[\alpha := B] = T$. So we have $\Delta \vdash \Downarrow^T (P[\alpha := B])$.
  The proof of $\Delta \vdash \bigtriangledown^T N$ is similar to the prior case since $U = \varepsilon$.

- Case $M = \blacktriangleright_\alpha P \xrightarrow{U} \blacktriangleright_\alpha P' = N$ with $P \xrightarrow{\alpha^{-1}U} P'$: Assume $\Delta \vdash \Downarrow^T \blacktriangleright_\alpha P$. Then $\Delta \vdash \Downarrow^{\alpha^{-1}T} P$. The typability of M implies the typability of P. We have $(\alpha^{-1}U)[\Delta := \varepsilon] \not\leq (\alpha^{-1}T)[\Delta := \varepsilon]$ because $U[\Delta := \varepsilon] \not\leq T[\Delta := \varepsilon]$. So, by the induction hypothesis, we obtain $\Delta \vdash \Downarrow^{\alpha^{-1}T} P'$. Therefore $\Delta \vdash \Downarrow^T N$.
  Assume $\Delta \vdash \bigtriangledown^T \blacktriangleright_\alpha P$. It must be the case that $\varepsilon \not\leq T[\Delta := \varepsilon]$. Therefore $\Delta \vdash \bigtriangledown^T \blacktriangleright_\alpha P'$.

- Case $M = \Lambda\alpha.P \xrightarrow{U} \Lambda\alpha.P' = N$ with $P \xrightarrow{U'} P'$ and $U = U'[\alpha := \varepsilon]$: We can assume without loss of generality that $\alpha \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(A) \cup \mathrm{FTV}(T) \cup \Delta$. Moreover $\alpha \notin \mathrm{FTV}(U)$ because $U = U'[\alpha := \varepsilon]$. It follows that

$$
\begin{aligned}
U'[\Delta, \alpha := \varepsilon] \;&=\; (U'[\alpha := \varepsilon])[\Delta := \varepsilon] \\
&\qquad \text{(by the definition of } U) \\
&=\; U[\Delta := \varepsilon] \\
&\qquad \text{(by assumption)} \\
&\not\preceq\; T[\Delta := \varepsilon] \\
&\qquad \text{(because } \alpha \notin \mathrm{FTV}(T)) \\
&=\; T[\Delta, \alpha := \varepsilon]
\end{aligned}
$$

Assume that $\Delta \vdash\Downarrow^T M$. Then $\Delta, \alpha \vdash\Downarrow^T P$. The typability of M implies the typability of P. So, by the induction hypothesis, we have $\Delta \vdash\Downarrow^T P'$, which implies $\Delta \vdash\Downarrow^T N$.

The proof of the remaining part is similar to other cases. $\qquad\qquad\square$

Appendix B. List of Error-generating and Error-propagating Rules

The List of Error-generating Rules.

$$
\frac{\vdash^\varepsilon M \Downarrow M' \qquad M' \notin \mathbb{Z} \qquad \Diamond \in \{+, -, *, =\}}{\vdash^\varepsilon M \Diamond N \Downarrow \mathbf{err}}
$$

$$
\frac{\vdash^\varepsilon M \Downarrow M' \qquad M' \in \mathbb{Z} \qquad \vdash^\varepsilon N \Downarrow N' \qquad N' \notin \mathbb{Z} \qquad \Diamond \in \{+, -, *, =\}}{\vdash^\varepsilon M \Diamond N \Downarrow \mathbf{err}}
$$

$$
\frac{\vdash^\varepsilon M \Downarrow M' \qquad M' \notin \{\mathbf{true}, \mathbf{false}\}}{\vdash^\varepsilon \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2 \Downarrow \mathbf{err}} \qquad \frac{}{\vdash^\varepsilon x \Downarrow \mathbf{err}}
$$

$$
\frac{\vdash^\varepsilon M \Downarrow M' \qquad M' \neq \lambda x : \tau.M''}{\vdash^\varepsilon M\ N \Downarrow \mathbf{err}} \qquad \frac{\vdash^\varepsilon M \Downarrow M' \qquad M' \neq \Lambda\alpha.M''}{\vdash^\varepsilon M\ A \Downarrow \mathbf{err}}
$$

$$
\frac{\vdash^\varepsilon M \Downarrow M' \qquad M' \neq \blacktriangleright_\alpha M''}{\vdash^\alpha \blacktriangleleft_\alpha M \Downarrow \mathbf{err}} \qquad \frac{}{\vdash^\varepsilon \blacktriangleleft_\alpha M \Downarrow \mathbf{err}}
$$

The List of Error-propagating Rules.

$$
\frac{\vdash^A M \Downarrow \mathbf{err} \qquad \Diamond \in \{+, -, *, =\}}{\vdash^A M \Diamond N \Downarrow \mathbf{err}} \qquad \frac{\vdash^A M \Downarrow M' \qquad \vdash^A N \Downarrow \mathbf{err} \qquad \Diamond \in \{+, -, *, =\}}{\vdash^A M \Diamond N \Downarrow \mathbf{err}}
$$

$$
\frac{\vdash^A M \Downarrow \mathbf{err}}{\vdash^A \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2 \Downarrow \mathbf{err}} \qquad \frac{\vdash^\varepsilon M \Downarrow \mathbf{true} \qquad \vdash^\varepsilon N_1 \Downarrow \mathbf{err}}{\vdash^\varepsilon \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2 \Downarrow \mathbf{err}}
$$

$$
\frac{\vdash^\varepsilon M \Downarrow \mathbf{false} \qquad \vdash^\varepsilon N_2 \Downarrow \mathbf{err}}{\vdash^\varepsilon \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2 \Downarrow \mathbf{err}} \qquad \frac{\vdash^A M \Downarrow M' \qquad \vdash^A N_1 \Downarrow \mathbf{err} \qquad A \neq \varepsilon}{\vdash^A \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2 \Downarrow \mathbf{err}}
$$

$$
\frac{\vdash^A M \Downarrow M' \qquad \vdash^A N_1' \Downarrow N_1' \qquad \vdash^A N_2 \Downarrow \mathbf{err} \qquad A \neq \varepsilon}{\vdash^A \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2 \Downarrow \mathbf{err}} \qquad \frac{\vdash^A M \Downarrow \mathbf{err} \qquad A \neq \varepsilon}{\vdash^A \lambda x : \tau.M \Downarrow \mathbf{err}}
$$

$$\frac{\vdash^A M \Downarrow \mathbf{err}}{\vdash^A M\,N \Downarrow \mathbf{err}} \qquad \frac{\vdash^A M \Downarrow M' \qquad \vdash^A N \Downarrow \mathbf{err}}{\vdash^A M\,N \Downarrow \mathbf{err}} \qquad \frac{\vdash^{A\alpha} M \Downarrow \mathbf{err}}{\vdash^A \blacktriangleright_\alpha M \Downarrow \mathbf{err}}$$

$$\frac{\vdash^A M \Downarrow \mathbf{err}}{\vdash^{A\alpha} \blacktriangleleft_\alpha M \Downarrow \mathbf{err}} \qquad \frac{\vdash^A M \Downarrow \mathbf{err}}{\vdash^A \Lambda\alpha.M \Downarrow \mathbf{err}} \qquad \frac{\vdash^A M \Downarrow \mathbf{err}}{\vdash^A M\,B \Downarrow \mathbf{err}}$$

$$\frac{\vdash^\varepsilon M[f := \mathbf{fix}\ f : \tau \to \sigma.M] \Downarrow \mathbf{err}}{\vdash^\varepsilon \mathbf{fix}\ f : \tau \to \sigma.M \Downarrow \mathbf{err}} \qquad \frac{\vdash^A M \Downarrow \mathbf{err} \qquad A \neq \varepsilon}{\vdash^A \mathbf{fix}\ f : \tau \to \sigma.M \Downarrow \mathbf{err}}$$

APPENDIX C. PROOFS OF PROPERTIES ABOUT MINI-ML

C.1. **Proof of Theorem 4.4 (Type Soundness of MiniML$^\triangleright$).**

Lemma C.1. *If* $\Gamma \vdash^{\alpha A} v : \tau$ *and* $v \in V^{\alpha A}$, *then* $\Gamma^{-\alpha} \vdash^A v : \tau$.

Proof. By induction on the structure of v. We show only cases for $v = x$ and $v = \blacktriangleleft_\beta v'$.

- Case $v = x$: We have $x : \tau@\alpha A \in \Gamma$. Because $x : \tau@A \in \Gamma^{-\alpha}$, $\Gamma^{-\alpha} \vdash^A x : \tau$.

- Case $v = \blacktriangleleft_\beta v'$: We have $A \neq \varepsilon$ because $\blacktriangleleft_\beta v' \in V^{\alpha A}$. That means $A = A'\beta$ for some A' and $\Gamma \vdash^{\alpha A'\beta} \blacktriangleleft_\beta v' : \tau$. So we have $\Gamma \vdash^{\alpha A'} v' : \triangleright_\beta \tau$. By the induction hypothesis, $\Gamma^{-\alpha} \vdash^{A'} v' : \triangleright_\beta \tau$. Therefore, by ($\blacktriangleleft$), we have $\Gamma^{-\alpha} \vdash^{A'\beta} \blacktriangleleft_\beta v' : \tau$. $\qquad\square$

Proof of Theorem 4.4. The first part of the theorem

> If Γ is ε-free and $\Gamma \vdash^A M : \tau$ and $\vdash^A M \Downarrow R$, then $R = v$ and $v \in V^A$ for some v and $\Gamma \vdash^A v : \tau$.

is proved by induction on the derivation of $\vdash^A M \Downarrow R$ with case analysis on the form of M and the last rule used to derive $\vdash^A M \Downarrow R$.

We only show representative cases here.

- Case $M = x$
  - Subcase $\dfrac{}{\vdash^A x \Downarrow x}(A \neq \varepsilon)$
    (1) Immediate.
  - Subcase $\dfrac{}{\vdash^\varepsilon x \Downarrow \mathbf{err}}$
    (1) We have $x : \tau@\varepsilon \in \Gamma$ because $\Gamma \vdash^\varepsilon x : \tau$ but this contradicts the assumption that Γ is ε-free.
- Case $M = \blacktriangleleft_\alpha N$
  - Subcase $\dfrac{\vdash^\varepsilon N \Downarrow \blacktriangleright_\alpha N'}{\vdash^\alpha \blacktriangleleft_\alpha N \Downarrow N'}$
    (1) We have $\Gamma \vdash^\varepsilon N : \triangleright_\alpha \tau$ because $\Gamma \vdash^\varepsilon \blacktriangleleft_\alpha N$.
    (2) By the induction hypothesis, $\Gamma \vdash^\varepsilon \blacktriangleright_\alpha N' : \triangleright_\alpha \tau$ and $\blacktriangleright_\alpha N' \in V^\varepsilon$.
    (3) So we have $\Gamma \vdash^\alpha N' : \tau$ and $N' \in V^\alpha$.
  - Subcase $\dfrac{\vdash^A N \Downarrow N'}{\vdash^{A\alpha} \blacktriangleleft_\alpha N \Downarrow \blacktriangleleft_\alpha N'}(A \neq \varepsilon)$
    (1) We have $\Gamma \vdash^A N : \triangleright_\alpha \tau$ because $\Gamma \vdash^A \blacktriangleleft_\alpha N : \tau$.
    (2) By the induction hypothesis, $\Gamma \vdash^A N' : \triangleright_\alpha \tau$ and $N' \in V^A$.
    (3) Therefore, $\Gamma \vdash^{A\alpha} \blacktriangleleft_\alpha N' : \tau$ by ($\blacktriangleleft$) and $\blacktriangleleft_\alpha N' \in V^{A\alpha}$.

– Subcase $\dfrac{\vdash^\varepsilon N \Downarrow N' \qquad N' \neq \blacktriangleright_\alpha N''}{\vdash^\alpha \blacktriangleleft_\alpha N \Downarrow \mathbf{err}}$

(1) We have $\Gamma \vdash^\varepsilon N : \triangleright_\alpha \tau$ because $\Gamma \vdash^\varepsilon \blacktriangleleft_\alpha N : \tau$.

(2) By the induction hypothesis, $\Gamma \vdash^\varepsilon N' : \triangleright_\alpha \tau$ and $N' \in V^\varepsilon$, which is a contradiction.

- Case $M = \Lambda\alpha.N$

– Subcase $\dfrac{\vdash^A N \Downarrow N'}{\vdash^A \Lambda\alpha.N \Downarrow \Lambda\alpha.N'}$

(1) We have $\tau = \forall\alpha.\tau'$ and $\Gamma \vdash^A N : \tau'$ for some τ' and $\alpha \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(A)$ because $\Gamma \vdash^A \Lambda\alpha.N : \tau$.

(2) By the induction hypothesis, $\Gamma \vdash^A N' : \tau'$ and $N' \in V^A$.

(3) Therefore, $\Gamma \vdash^A \Lambda\alpha.N' : \tau$ by (GEN) and $\Lambda\alpha N' \in V^A$.

- Case $M = N\,B$

– Subcase $\dfrac{\vdash^\varepsilon N \Downarrow \Lambda\alpha.N' \qquad \vdash^\varepsilon N'[\alpha := B] \Downarrow M}{\vdash^\varepsilon N\,B \Downarrow M}$

(1) We have $\tau = \sigma[\alpha := B]$ and $\Gamma \vdash^\varepsilon N : \forall\alpha.\sigma$ for some σ and α, because $\Gamma \vdash^\varepsilon N\,B : \tau$.

(2) By the induction hypothesis, $\Gamma \vdash^\varepsilon \Lambda\alpha.N' : \forall\alpha.\sigma$. Therefore $\Gamma \vdash^\varepsilon N' : \sigma$ and $\alpha \notin \mathrm{FTV}(\Gamma)$.

(3) By Substitution Lemma (Lemma 3.1) and $\alpha \notin \mathrm{FTV}(\Gamma)$, $\Gamma \vdash^\varepsilon N'[\alpha := B] : \tau$.

(4) By the induction hypothesis, $\Gamma \vdash^\varepsilon M : \tau$ and $M \in V^\varepsilon$.

– Subcase $\dfrac{\vdash^A N \Downarrow N'}{\vdash^A N\,B \Downarrow N'\,B}(A \neq \varepsilon)$

(1) We have $\tau = \sigma[\alpha := B]$ and $\Gamma \vdash^A N : \forall\alpha.\sigma$ for some σ and α, because $\Gamma \vdash^A N\,B : \tau$.

(2) By the induction hypothesis, $\Gamma \vdash^A N' : \forall\alpha.\sigma$ and $N' \in V^A$.

(3) Therefore, $\Gamma \vdash^A N'\,B : \tau$ by (INS) and $N'\,B \in V^A$.

– Subcase $\dfrac{\vdash^\varepsilon N \Downarrow N' \qquad N' \neq \Lambda\alpha.N''}{\vdash^\varepsilon N\,B \Downarrow \mathbf{err}}$

(1) We have $\tau = \sigma[\alpha := B]$ and $\Gamma \vdash^\varepsilon N : \forall\alpha.\sigma$ for some σ and α, because $\Gamma \vdash^\varepsilon N\,B : \tau$.

(2) By the induction hypothesis, $\Gamma \vdash^\varepsilon N' : \forall\alpha.\sigma$ and $N' \in V^A$, which is a contradiction.

Now, we prove the second part. By the first part, $\Gamma \vdash^\varepsilon v : \triangleright_\alpha \tau$ and $v \in V^\varepsilon$. Therefore $v = \blacktriangleright_\alpha v'$ and $v' \in V^\alpha$. From the typing rules, we have $\Gamma \vdash^\alpha v' : \tau$. Then, we have $\Gamma^{-\alpha} \vdash^\varepsilon v' : \tau$ by Lemma C.1. $\qquad\square$

C.2. **Proof of Theorem 4.5.** We first prove type soundness for the new type system (Lemma C.8). Although the new type system identifies a subset of well-typed $\lambda^\triangleright$ terms, it has to be proved again, since we want to gurantee that the evaluation of a term typed under the new type system results in (if converges) a value that is also typed under it. We start with proving various substitution lemmas.

Lemma C.2 (Term Substitution Preserves Typing). *If* $\Gamma, x : \sigma@B; \Delta \vdash^A_s M : \tau$ *and* $\Gamma; \Delta \vdash^B_s M : \sigma$, *then* $\Gamma; \Delta \vdash^A_s M[x := N] : \tau$.

Proof. Similar to the proof of Lemma 3.1. $\qquad\square$

Next, we show that substitution for transition variables preserves various kinds of well-formedness and typing.

Lemma C.3. *If* $\Delta, \alpha@B \vdash_s A$ *and* $\Delta \vdash_s BC$, *then* $\Delta[\alpha := C] \vdash_s A[\alpha := C]$.

Proof. By induction of the length of A. The base case is trivial because $\Delta \vdash_s \varepsilon$ for any Δ. So, we show the induction step.

Let $A = \alpha_1 \dots \alpha_n$. Assume $\Delta, \alpha @ B \vdash_s A$. It is easy to see that $\Delta, \alpha @ B \vdash_s \alpha_1 \dots \alpha_n$ implies $\Delta, \alpha @ B \vdash_s \alpha_1 \dots \alpha_{n-1}$. Then $\Delta[\alpha := C] \vdash_s (\alpha_1 \dots \alpha_{n-1})[\alpha := C]$ by the induction hypothesis. We have $\alpha_n @ \alpha_1 \dots \alpha_{n-1} \in (\Delta, \alpha @ B)$ from the definition of $\Delta, \alpha @ B \vdash_s A$. The proof is divided into two cases.

- Case $\alpha_n \neq \alpha$: It is the case that $\alpha_n @ \alpha_1 \dots \alpha_{n-1} \in \Delta$. The conclusion is trivial, because $\alpha_n @ (\alpha_1 \dots \alpha_{n-1})[\alpha := C] \in \Delta[\alpha := C]$.

- Case $\alpha_n = \alpha$: It is the case that $\alpha_1 \dots \alpha_{n-1} = B$. Let $C = \beta_1 \dots \beta_m$. Because $\Delta \vdash_s BC$, we have

$$\Delta_0 = \{\alpha_1 @ \varepsilon, \alpha_2 @ \alpha_1, \dots, \alpha_{n-1} @ \alpha_1 \dots \alpha_{n-2}, \beta_1 @ B, \beta_2 @ B\beta_1, \dots, \beta_m @ B\beta_1 \dots \beta_{m-1}\} \subseteq \Delta.$$

Let $\Delta_1 = \Delta - \Delta_0$. Note that $\alpha \notin \{\alpha_1, \dots, \alpha_{n-1}, \beta_1, \dots, \beta_m\} = \text{FTV}(\Delta_0)$ and $\Delta_0 \vdash_s BC$. Therefore we have $\Delta[\alpha := C] = \Delta_0[\alpha := C] \cup \Delta_1[\alpha := C] = \Delta_0 \cup \Delta_1[\alpha := C]$. Because $(B\alpha)[\alpha := C] = BC$ and $\Delta_0 \vdash_s BC$, we obtain $\Delta[\alpha := C] \vdash_s A[\alpha := C]$. $\square$

Lemma C.4. *If $\vdash_s \Delta, \alpha @ B$ and $\Delta \vdash_s BC$, then $\vdash_s \Delta[\alpha := C]$.*

Proof. Let $\beta @ A \in \Delta[\alpha := C]$. We show $\Delta[\alpha := C] \vdash_s A$.

There exists some A' such that $\beta @ A' \in \Delta$ and $A = A'[\alpha := C]$ because $\beta @ A \in \Delta[\alpha := C]$. Since $\vdash_s \Delta, \alpha @ B$, we have $\Delta, \alpha @ B \vdash_s A'$. So by Lemma C.3, we have $\Delta[\alpha := C] \vdash_s A'[\alpha := C]$ as required. $\square$

Lemma C.5. *If $\Delta, \alpha @ B \vdash_s^A \tau$ and $\Delta \vdash_s BC$, then $\Delta[\alpha := C] \vdash_s^{A[\alpha := C]} \tau[\alpha := C]$.*

Proof. By induction on the structure of the type τ.

- Case $\tau = \textbf{int}$: Because $\Delta, \alpha @ B \vdash_s^A \textbf{int}$, we have $\Delta, \alpha @ B \vdash_s A$. So by Lemma C.3, we obtain $\Delta[\alpha := C] \vdash_s A[\alpha := C]$. Therefore we obtain $\Delta[\alpha := C] \vdash_s^{A[\alpha := C]} \textbf{int}$ by applying (ST-BASE).

- Case $\tau = \triangleright_\beta \sigma$: Because $\Delta, \alpha @ B \vdash_s^A \triangleright_\beta \sigma$, we have $\Delta, \alpha @ B \vdash_s^{A\beta} \sigma$. By the induction hypothesis, we have $\Delta[\alpha := C] \vdash_s^{(A\beta)[\alpha := C]} \sigma[\alpha := C]$. Therefore by applying (ST-CODE) as many times as needed, we obtain $\Delta[\alpha := c] \vdash_s^{A[\alpha := C]} \triangleright_{\beta[\alpha := C]}(\sigma[\alpha := C])$.

- Case $\tau = \forall \beta @ A'.\sigma$: Because $\Delta, \alpha @ B \vdash_s^A \forall \beta @ A'.\sigma$, we have $\Delta, \alpha @ B, \beta @ AA' \vdash_s^A \sigma$. By the induction hypothesis, we have $\Delta[\alpha := C], \beta @ (AA')[\alpha := C] \vdash_s^{A[\alpha := C]} \sigma[\alpha := C]$. Hence by applying (ST-UNIV), we obtain $\Delta[\alpha := C] \vdash_s^{A[\alpha := C]} \forall \beta @ (A'[\alpha := C]).(\sigma[\alpha := C])$. $\square$

Lemma C.6. *If $\Delta, \alpha @ B \vdash_s \Gamma$ and $\Delta \vdash_s BC$, then $\Delta[\alpha := C] \vdash_s \Gamma[\alpha := C]$.*

Proof. Take $x : \tau @ A \in \Gamma$. By the well-formedness of Γ, we have $\Delta, \alpha @ B \vdash_s^A \tau$. By Lemma C.5, we have $\Delta[\alpha := C] \vdash_s^{A[\alpha := C]} \tau[\alpha := C]$, as required. $\square$

Now we show the substitution lemma for transition variables. However, transition substitution does *not* preserve typing in general, and, even worse, transition substitution is undefined in some cases. This is because of the distinction between two kinds of applications $M[\alpha]$ and $M\,A$. For example, if we substitute A for α into $M[\alpha]$, the result is $M[A]$, which is not a valid term.

So, the substitution lemma we will prove is a restricted one. Transition substitution preserves typing in the following two cases:

(1) Substitution of a single transition variable, $M[\alpha := \beta]$ for any M.
(2) Substitution of an arbitrary transition A at the initial stage in a value, $v[\alpha := A]$, where v is a value (an element of V^ε) and $\alpha@\varepsilon$.

They are what we need to prove type soundness.

Lemma C.7.

(1) *If $\Gamma; \Delta, \alpha@B \vdash_s^A M : \tau$ and $\beta@B \in \Delta$, then $\Gamma[\alpha := \beta]; \Delta[\alpha := \beta] \vdash_s^{A[\alpha:=\beta]} M[\alpha := \beta] : \tau[\alpha := \beta]$.*

(2) *For any $v \in V^A$, if $\Gamma; \Delta, \alpha@\varepsilon \vdash_s^A v : \tau$ and $\Delta \vdash_s B$, then $\Gamma[\alpha := B]; \Delta[\alpha := B] \vdash_s^{A[\alpha:=B]} v[\alpha := B] : \tau[\alpha := B]$.*

Proof. We can prove (1) by induction on the type derivation.

• Case (S-VAR): We have $M = x$ and $\Delta, \alpha@B \vdash_s \Gamma, x : \tau@A$. By Lemma C.6, we have $\Delta[\alpha := \beta] \vdash_s \Gamma[\alpha := \beta], x : \tau[\alpha := \beta]@A[\alpha := \beta]$. By applying the (S-VAR) rule, we obtain $\Gamma[\alpha := \beta]; \Delta[\alpha := \beta] \vdash_s^{A[\alpha:=\beta]} x : \tau[\alpha := \beta]$.

• Case (S-GEN): We have $M = \Lambda\gamma.M'$ and $\tau = \forall\gamma@C.\sigma$ and $\Gamma; \Delta, \alpha@B, \gamma@AC \vdash_s^A M' : \sigma$. By the induction hypothesis, we have $\Gamma[\alpha := \beta]; \Delta[\alpha := \beta], \gamma@(AC)[\alpha := \beta] \vdash_s^{A[\alpha:=\beta]} M'[\alpha := \beta] : \sigma[\alpha := \beta]$ (note that $\alpha \neq \gamma$). By applying the (S-GEN) rule, we obtain $\Gamma[\alpha := \beta]; \Delta[\alpha := \beta] \vdash_s^{A[\alpha:=\beta]} \Lambda\gamma.(M'[\alpha := \beta]) : \forall\gamma@(C[\alpha := \beta]).\sigma[\alpha := \beta]$.

• Case (S-INS2): We have $M = M'[\gamma_1]$, and there is some σ and γ_0 and C such that $\tau = \sigma[\gamma_0 := \gamma_1]$ and $\Gamma; \Delta, \alpha@B \vdash_s^A M' : \forall\gamma_0@C.\sigma$ and $\gamma_1@AC \in \Delta$. We can assume without loss of generality $\gamma_0 \notin \mathrm{FTV}(C) \cup \{\alpha, \beta\}$. By the induction hypothesis, we have $\Gamma[\alpha := \beta]; \Delta[\alpha := \beta] \vdash_s^{A[\alpha:=\beta]} M'[\alpha := \beta] : (\forall\gamma_0@C.\sigma)[\alpha := \beta]$. Since $(\forall\gamma_0@C.\sigma)[\alpha := \beta] = \forall\gamma_0@(C[\alpha := \beta]).(\sigma[\alpha := \beta])$ and $\gamma_1[\alpha := \beta]@(AC)[\alpha := \beta] \in \Delta[\alpha := \beta]$, we can apply (S-INS2) to obtain $\Gamma[\alpha := \beta]; \Delta[\alpha := \beta] \vdash_s^{A[\alpha:=\beta]} (M'[\gamma_1])[\alpha := \beta] : \sigma[\alpha := \beta][\gamma_0 := \gamma_1[\alpha := \beta]]$. Because $\gamma_0 \notin \{\alpha, \beta\}$, we have

$$\begin{aligned} \sigma[\alpha := \beta][\gamma_0 := \gamma_1[\alpha := \beta]] &= \sigma[\gamma_0 := \gamma_1][\alpha := \beta] \\ &= \tau[\alpha := \beta]. \end{aligned}$$

The proof of (2) is the same as the proof of (1), except for the case (S-INS2). We prove this case. Let $v = v'[\gamma_1]$. To prove this case, it is important to notice that $\gamma_1 \neq \alpha$. This fact can be shown by the following observation. Assume that $\gamma_1 = \alpha$. Then, the stage A of v must be ε because the stage of α is ε, but $v \in V^\varepsilon$ contradicts the fact that v is a transition application. Therefore $\gamma_1 \neq \alpha$. The rest of the proof is the same as (1). $\square$

Then we show the preservation of $\vdash_s$ by evaluation.

Lemma C.8 (Type Soundness (2)). *If $\Gamma; \Delta \vdash_s^A M$ and $\vdash^A M \Downarrow R$, then $R = v^A$ and $\Gamma; \Delta \vdash_s^A v^A$ for some $v^A \in V^A$.*

Proof. Similar to the proof of Theorem 4.4.   □

We also have the counterpart of Lemma C.1 for the new type system. We define
$\Delta^{-\alpha} = \{\beta@B \mid \beta@\alpha B \in \Delta\}$.

Lemma C.9. *If* $\Gamma; \Delta \vdash_s^{\alpha A} M : \tau$ *and* $M \in V^{\alpha A}$*, then we have* $\Gamma^{-\alpha}; \Delta^{-\alpha} \vdash_s^A M : \tau$.

Proof. Similar to the proof of Lemma C.1.   □

Finally, we prove Theorem 4.5 via a more general property for which proof by induction works.

Theorem C.10. *If* Γ *is* ε-*free and* $\Gamma; \Delta \vdash_s^A M : \tau$ *and* n *is the length of* A*, then*
(1) if $\vdash^A M \Downarrow N$*, then* $\vdash_e^n \flat(M) \Downarrow \flat(N)$*; and*
(2) if $\vdash_e^n \flat(M) \Downarrow N$*, there is some* N' *such that* $\vdash^A M \Downarrow N'$ *and* $N = \flat(N')$.

Proof. By induction on the derivation of $\vdash^A M \Downarrow N$ (for (1)) or $\vdash_e^n \flat(M) \Downarrow N$ (for (2)) with case analysis on the last rule used in the derivation. Actually, the only interesting case is when $M = M_0\, B$ and $\vdash^A M_0\, B \Downarrow N$.

- Proof of (1): Assume $M = M_0\, B$ and $\vdash^A M_0\, B \Downarrow N$. The case $A \neq \varepsilon$ is easy. Assume $A = \varepsilon$. So, the last derivation step for $\vdash^\varepsilon M_0\, B \Downarrow N$ is of the form

$$\frac{\vdash^\varepsilon M_0 \Downarrow \Lambda\alpha.M_1 \qquad \vdash^\varepsilon M_1[\alpha := B] \Downarrow N}{\vdash^\varepsilon M_0\, B \Downarrow N}.$$

  Since $\Gamma; \Delta \vdash_s^\varepsilon M_0\, B : \tau$, we have $\Gamma; \Delta \vdash_s^\varepsilon M_0 : \forall\alpha@\varepsilon.\, \rhd_\alpha \sigma$ and $\tau = \sigma[\alpha := B]$ for some σ. By Lemma C.8, we have $\Gamma; \Delta \vdash_s^\varepsilon \Lambda\alpha.M_1 : \forall\alpha@\varepsilon.\, \rhd_\alpha \sigma$ and $\Lambda\alpha.M_1 \in V^\varepsilon$. Therefore, $\Gamma, \Delta, \alpha@\varepsilon \vdash_s^\varepsilon M_1 : \rhd_\alpha\sigma$ and $M_1 \in V^\varepsilon$. Because $M_1 \in V^\varepsilon$, we have $M_1 = \blacktriangleright_\alpha M_2$ with $M_2 \in V^\alpha$. So $\Gamma; \Delta, \alpha@\varepsilon \vdash_s^\alpha M_2 : \sigma$. By Lemma C.9, $\Gamma^{-\alpha}; (\Delta, \alpha@\varepsilon)^{-\alpha} \vdash_s^\varepsilon M_2 : \sigma$. Because $(\Delta, \alpha@\varepsilon)^{-\alpha} = \Delta^{-\alpha}$, we have $\alpha \notin \mathrm{FTV}((\Delta, \alpha@\varepsilon)^{-\alpha})$, and therefore $\alpha \notin \mathrm{FTV}(M_2)$. So $(\blacktriangleright_\alpha M_2)[\alpha := B] = \blacktriangleright_B M_2$. Then, we know that the last derivation step must have a more specific shape:

$$\frac{\vdash^\varepsilon M_0 \Downarrow \Lambda\alpha.\blacktriangleright_\alpha M_2 \qquad \vdash^\varepsilon \blacktriangleright_B M_2 \Downarrow N}{\vdash^\varepsilon M_0\, B \Downarrow N}$$

  By the induction hypothesis, we have $\vdash_e^0 \flat(M_0) \Downarrow \Lambda \blacktriangleright \flat(M_2)$ and $\vdash_e^0 \overbrace{\blacktriangleright \ldots \blacktriangleright}^{m} \flat(M_2) \Downarrow \flat(N)$, where m is the length of B, because $\flat(\blacktriangleright_B M_2) = \overbrace{\blacktriangleright \ldots \blacktriangleright}^{m} \flat(M_2)$. Finally, we obtain $\vdash_e^0 \flat(M_0)\, m \Downarrow \flat(N)$ from them.

- Proof of (2): Assume $M = M_0\, B$ and $\vdash_e^n \flat(M_0\, B) \Downarrow N^\flat$. The case $n \neq 0$ is easy. We assume $n = 0$. Then, the last derivation step is of the form:

$$\frac{\vdash_e^0 \flat(M_0) \Downarrow \Lambda \blacktriangleright M_2^\flat \qquad \vdash_e^0 \overbrace{\blacktriangleright \ldots \blacktriangleright}^{m} M_2^\flat \Downarrow N^\flat}{\vdash_e^0 \flat(M_0)\, m \Downarrow N^\flat}$$

where m is the length of B. Since $\Gamma; \Delta \vdash_s^\varepsilon M_0\, B : \tau$, we have $\Gamma; \Delta \vdash_s^\varepsilon M_0 : \forall\alpha@\varepsilon.\, \rhd_\alpha \sigma$ for some σ.

By the induction hypothesis, there is some M_1 such that $\vdash^\varepsilon M_0 \Downarrow M_1$ and $\Lambda \blacktriangleright M_2^\flat = \flat(M_1)$. So, $M_1 = \Lambda\alpha.\blacktriangleright_\beta M_2$ for some α and β and M_2 such that $\flat(M_2) = M_2^\flat$. Actually, $\alpha = \beta$ since M_0 has the type $\forall\alpha@\varepsilon.\triangleright_\alpha\sigma$ and evaluation preserves types. By the same argument as in the proof of (1), we have $\alpha \notin \mathrm{FTV}(M_2)$. Therefore, $(\blacktriangleright_\alpha M_2)[\alpha := B] = \blacktriangleright_B M_2$.

Because $\flat(\blacktriangleright_B M_2) = \overbrace{\blacktriangleright\ldots\blacktriangleright}^{m} M_2^\flat$, by the induction hypothesis, we have $\vdash^\varepsilon \blacktriangleright_B M_2 \Downarrow N$ for some N with $\flat(N) = N^\flat$. $\qquad\square$

Proof of Theorem 4.5. Theorem 4.5 is a special case of Theorem C.10. $\qquad\square$

Appendix D. Proof of Completeness (Theorem 5.2)

D.1. **Completeness in the Quantifier-Free Setting.** We first prove completeness in the quantifier-free setting: the set of propositions has no quantifiers and the deduction rules and semantics has no rule for quantifiers.

Formally, the set $\Phi_\perp^{-\forall}$ of quantifier-free propositions is defined by the following grammar:

$$\text{Propositions} \qquad \phi \in \Phi_\perp^{-\forall} ::= b \mid \perp \mid \phi \to \phi \mid \triangleright_\alpha \phi \quad.$$

The natural deduction system consists of (Var), (Abs), (App), ($\blacktriangleright$), ($\blacktriangleleft$) and ($\perp$-E). The Kripke semantics for this fragment is essentially the same as the full logic, but the proposition ϕ in the satisfaction relation $\mathcal{T}, v, \rho; s \Vdash \phi$ is restricted to be quantifier-free one.

An assumption Γ is a (possibly infinite) set $\{\phi_i @ A_i\}$ where $\phi_i \in \Phi_\perp^{-\forall}$ and $A_i \in \Sigma^*$. We say an assumption Γ is *consistent* if $\Gamma \nvdash^\varepsilon \perp$. (When Γ is infinite, $\Gamma \vdash^A \phi$ means that there exists a finite assumption $\Gamma' \subset \Gamma$ such that $\Gamma' \vdash^A \phi$.) We say Γ is *maximally consistent* when Γ is consistent and maximal (under the ordinary set inclusion ordering).

Lemma D.1. *For any consistent assumption Γ, there is a maximally consistent assumption Γ' s.t. $\Gamma' \supseteq \Gamma$.*

Proof. We apply Zorn's Lemma to the set $\Theta = \{\Delta \mid \Gamma \subseteq \Delta$ and Δ is consistent$\}$. To apply Zorn's lemma, we prove that every totally ordered subsets $\Xi \subseteq \Theta$ has its upper bound in Θ, by showing $\bigcup \Xi \in \Theta$. It is clear that $\Gamma \subseteq \bigcup \Xi$.

We show that $\bigcup \Xi$ is consistent. Assume $\bigcup \Xi$ is inconsistent, i.e. $\bigcup \Xi \vdash^\varepsilon \perp$. Then there is a finite subset $\Delta \subseteq \bigcup \Xi$ such that $\Delta \vdash^\varepsilon \perp$. Let $\{\phi_i @ A_i \mid 0 \leq i \leq n\} = \Delta$. Because $\Delta \subseteq \bigcup \Xi$, for every $\phi_i @ A_i \in \Delta$, there is an assumption $\Xi_i \in \Xi$ which satisfies $\phi_i @ A_i \in \Xi_i$. Because the number n of elements in Δ is finite and Ξ is totally ordered, there is a maximal assumption $\Xi_j \supseteq \Xi_i$ $(0 \leq i \leq n)$. Therefore $\Delta \subseteq \Xi_j$ and Ξ_j is inconsistent, and this causes a contradiction. $\qquad\square$

Let Γ be maximally consistent. We construct a transition system $\mathcal{T}_\Gamma$ and valuations ρ_Γ and v_Γ from it. The transition system[5] $\mathcal{T}_\Gamma$ is $(\Sigma^*, \Sigma, \{\xrightarrow{\alpha} \mid \alpha \in \Sigma\})$, where $A \xrightarrow{\alpha} A\alpha$, and valuation ρ_Γ for transition variables is defined by $\rho_\Gamma(\alpha) = \alpha$. We define valuation v_Γ for propositional variables as $v_\Gamma(A, p) = 1$ if and only if $p@A \in \Gamma$.

Lemma D.2. $\mathcal{T}_\Gamma, v_\Gamma, \rho_\Gamma; \varepsilon \Vdash \triangleright_A \phi$ *if and only if* $\phi@A \in \Gamma$.

Proof. Easy induction with respect to the proposition ϕ.   □

Theorem D.3 (Completeness for the quantifier-free fragment)**.** *If* $\Gamma \Vdash \phi$ *then* $\Gamma \vdash^\varepsilon \phi$.

Proof. We prove by contraposition. Assume $\Gamma \nvdash^\varepsilon \phi$. Then $\Gamma, (\phi \to \bot)@\varepsilon$ is consistent. By Lemma D.1 there is a maximally consistent assumption $\Gamma' \supseteq \Gamma \cup \{(\phi \to \bot)@\varepsilon\}$. By Lemma D.2, $\mathcal{T}_{\Gamma'}, v_{\Gamma'}, \rho_{\Gamma'}; \varepsilon \Vdash \Gamma$ and $\mathcal{T}_{\Gamma'}, v_{\Gamma'}, \rho_{\Gamma'}; \varepsilon \Vdash \phi \to \bot$. So, $\mathcal{T}_{\Gamma'}, v_{\Gamma'}, \rho_{\Gamma'}; \varepsilon \nVdash \phi$ and, therefore, $\Gamma \nVdash \phi$.   □

D.2. **Completeness of the Full Logic.** We prove completeness for the full logic. The way to prove completeness is similar to that for first-order predicate logic.

We abbreviate $\phi \to \bot$ to $\neg\phi$, $(\neg\phi) \to \psi$ to $\phi \lor \psi$, $\neg(\phi \to \neg\psi)$ to $\phi \land \psi$ and $\neg\forall\alpha.\neg\phi$ to $\exists\alpha.\phi$. The deduction rules for these connectives can be given as in ordinary classical logic, e.g. as follows:

$$\frac{\Gamma \vdash^A \phi \qquad \Gamma \vdash^A \psi}{\Gamma \vdash^A \phi \land \psi} \ (\land\text{-I})$$

$$\frac{\Gamma \vdash^A \exists\alpha.\phi \qquad \Gamma, \phi@A \vdash^A \psi \qquad \alpha \notin \mathrm{FTV}(\Gamma, \psi@A)}{\Gamma \vdash^A \psi} \ (\exists\text{-E}) \ .$$

We abbreviate $\forall\alpha_1\forall\alpha_2\ldots\forall\alpha_n\phi$ to $\forall^n\bar{\alpha}\phi$. We also write ϕ as $\forall^0\bar{\alpha}\phi$. We say a proposition is in *prenex normal form* when it has the form

$$\forall^{n_0}\bar{\alpha}_0 \exists\beta_1 \forall^{n_1}\bar{\alpha}_1 \ldots \exists\beta_m \forall^{n_{m+1}}\bar{\alpha}_{m+1}\psi$$

and ψ has no quantifier. It is easy to see that for any proposition ϕ there is an equivalent proposition ϕ' (i.e., $\vdash^\varepsilon \phi \leftrightarrow \phi'$) which is in prenex normal form. Therefore we assume without loss of generality that all propositions in an assumption are in prenex normal form.

We construct a quantifier-free assumption $\Delta(\Gamma)$ from an assumption Γ with quantifiers. Assume that all binding transition variables are different from other binding variables and free variables in Γ.

First we construct "Herbrand universe" from Γ. Herbrand universe is a term algebra, which is freely generated algebra from a signature. Let Σ_0 be the set of free transition variables in Γ and Σ_1 be the set of binding transition variables in Γ. The set S_Γ is defined as follows:

$$S_\Gamma = \{(\beta_k, \Sigma_{i=1}^{k-1} n_i) \mid \forall^{n_0}\bar{\alpha}_0\exists\beta_1 \ldots \exists\beta_k \ldots \forall^{n_m}\bar{\alpha}_m\psi@A \in \Gamma\}$$

We regard $\mathrm{dom}(S_\Gamma)$ as the set of pairs of function symbols and its arities. If there is no constant (i.e., arity 0 function) in S_Γ and $\Sigma_0 = \Sigma_1 = \emptyset$, then we add a constant $(c, 0)$ to S_Γ. If there is no function symbol whose arity is not 0, then we add a function symbol $(f, 1)$ to S_Γ. We define the set H_Γ as the least set which satisfies the following condition:

• if $(\beta, n) \in S_\Gamma$ and $\alpha_i \in H_\Gamma \cup \Sigma_0 \cup \Sigma_1 (1 \le i \le n)$, then $\beta(\alpha_1, \ldots, \alpha_n) \in H_\Gamma$

---

[5]If we want to prove completeness of the logic in Sec. 5.3, we define $\mathcal{T}$ as $(\{A \in \Sigma^* \mid \bot@A \notin \Gamma\}, \Sigma, \{\xrightarrow{\alpha} \mid \alpha \in \Sigma\})$, where $A \xrightarrow{\alpha} A\alpha$ if $\bot@A\alpha \notin \Gamma$ and otherwise undefined.

Second we regard Σ as the "Herbrand universe" by following observation. Because Σ is countably infinite and Σ_0 and Σ_1 is countable, we assume without loss of generality that $\Sigma\backslash(\Sigma_0 \cup \Sigma_1)$ is countably infinite. It is easy to see that H_Γ is countably infinite. Therefore there is an bijection $H_\Gamma \to (\Sigma\backslash(\Sigma_0 \cup \Sigma_1))$. By using this bijection, we regard any element in H_Γ as an element in Σ. So $\Sigma_0 \cup \Sigma_1 \cup H_\Gamma = \Sigma$ and $\Sigma_0 \cap H_\Gamma = \Sigma_1 \cap H_\Gamma = \emptyset$.

Then we define $\Delta(\Gamma)$ as follows:

$$\Delta(\Gamma) = \{\psi \quad [\bar{\alpha}_0, \beta_1, \bar{\alpha}_1, \beta_2, \ldots \bar{\alpha}_m := \bar{\gamma}_0, \beta_1(\bar{\gamma}_0), \bar{\gamma}_1, \beta_2(\bar{\gamma}_0, \bar{\gamma}_1), \ldots, \bar{\gamma}_m]@A$$
$$| \; \forall^{n_0}\bar{\alpha}_0 \exists\beta_1 \ldots \forall^{n_m}\bar{\alpha}_m \psi@A \in \Gamma, \gamma_{ij} \in \Sigma\}$$

Lemma D.4. *If $\Gamma \nvdash^\varepsilon \bot$, then $\Delta(\Gamma) \nvdash^\varepsilon \bot$ in quantifier-free logic.*

Proof. We prove this lemma by contraposition.

Assume that $\Delta(\Gamma) \vdash^\varepsilon \bot$. Then there is a finite subset $\{\phi_i@A_i \mid 0 \leq i \leq n\} \subseteq \Delta$ such that $\{\phi_i@A_i \mid 0 \leq i \leq n\} \vdash^\varepsilon \bot$. From the construction of $\Delta(\Gamma)$, for every ϕ_i, there are θ_i and $\gamma_{ij} \in \Sigma$ such that $\theta_i@A \in \Gamma$ and $\theta_i = \forall^{n_0}\bar{\alpha}_0\exists\beta_1 \ldots \forall^{n_m}\bar{\alpha}_m\psi_i$ and $\phi_i = \psi_i[\bar{\alpha}_0, \beta_1, \bar{\alpha}_1, \beta_2, \ldots \bar{\alpha}_m := \bar{\gamma}_0, \beta_1(\bar{\gamma}_0), \bar{\gamma}_1, \beta_2(\bar{\gamma}_0, \bar{\gamma}_1), \ldots, \bar{\gamma}_m]$. Then we can get $\{\theta_i@A_i \mid 0 \leq i \leq n\} \vdash^\varepsilon \bot$ by using (INS) and ($\exists$-E) in an appropriate order. Therefore $\Gamma \vdash^\varepsilon \bot$. □

Lemma D.5. *If $\mathcal{T}, v, \rho; \varepsilon \Vdash \Delta(\Gamma)$, then $\mathcal{T}, v, \rho; \varepsilon \Vdash \Gamma$.*

Proof. Let $\forall^{n_0}\bar{\alpha}_0\exists\beta_1 \ldots \forall^{n_m}\bar{\alpha}_m\psi@A \in \Gamma$. We can get the following proposition by easy induction on the number of existential quantifiers.

for all $\gamma_{i,j} \in \Sigma \mathcal{T}, v, \rho; \varepsilon \Vdash \exists\beta_k\forall^{n_k}\bar{\alpha}_k \ldots \forall^{n_m}\bar{\alpha}_n\psi[\bar{\alpha}_0, \beta_1, \ldots, \bar{\alpha}_k := \bar{\gamma}_0, \beta_1(\bar{\gamma}_0), \ldots, \bar{\gamma}_k]$

This lemma is an easy consequence of this proposition. □

Proof of Theorem 5.2. We prove by contraposition. Assume $\Gamma \nvdash^\varepsilon \phi$. Then $\Gamma, \neg\phi@\varepsilon \nvdash^\varepsilon \bot$. By Lemma D.4, $\Delta(\Gamma \cup \{\neg\phi@\varepsilon\})$ is a consistent assumption in the quantifier-free fragment. Therefore, from completeness of the quantifier-free fragment, there is a model such that $\mathcal{T}, v, \rho; \varepsilon \Vdash \Delta(\Gamma \cup \{\neg\phi@\varepsilon\})$. By Lemma D.5, $\mathcal{T}, v, \rho; \varepsilon \Vdash \Gamma$ and $\mathcal{T}, v, \rho; \varepsilon \Vdash \neg\phi$. Therefore $\mathcal{T}, v, \rho; \varepsilon \nVdash \phi$. So $\Gamma \nVdash \phi$. □