Kami: A modal type theory for distributed systems

Maxim Urschumzew Miëtek Bak

April 16, 2024

We present Kami, a programming language for distributed systems based on Modal Type Theory[4] by Gratzer. In particular, we describe a mode theory which is suitable for distributed programming and show how the primitives of Chorλ[1], a choreographic programming language, can be recovered in Kami. Finally we sketch categorical semantics which give rise to a generic compilation procedure for Kami.

1 Introduction

Modalities provide a way to extend a type theory with domain-specific "modes of being". These can be used to track runtime properties of code directly in the type system. For example, modalities have been used to distinguish between evaluated and unevaluated code in the context of metaprogramming and staged execution[2]. They have also been used to encode the notion of "location" in ML5[6], a language for distributed systems. In fact, the "`at`"-modality of ML5 serves the same purpose as the location annotations of choreographic languages[1, 3], and it stands to reason that existing research on modalities could provide some stepping stones for the design of type systems for choreographic programming languages.

In this paper we explore how Chorλ[1], a functional choreographic programming language, can be rebased on top of a simply typed lambda calculus with modalities (a variant of MTT[4]). This requires the introduction of two new concepts: Common knowledge between multiple participants, and local knowledge of global computations.

2 Preliminaries

We give a short introduction to Chorλ, and to Modal Type Theory (MTT).

2.1 Chorλ

Chorλ[1] is a functional choreographic programming language introduced by Cruz-Filipe et al. It is an extension of simply typed lambda calculus with location annotations. For example $t : \mathsf{Bool}@r$ means that t evaluates to a value of type Bool located at role r. It has two communication primitives: **com** for communicating data and **select** for communicating choice of branching.

2.2 Modal Type Theory

Modal Type Theory[4], recently introduced by Gratzer in his thesis, is a framework for modal type theories. The language is parametrized by a system of modalities which are specified in form of a *mode theory*. MTT has full dependent types, but we only present a simply typed fragment here.

Definition 1 ([4, Chapter 6.1.1])**.** A *mode theory* is given by the following data:

- A set of *modes* M. Every mode $m \in M$ instantiates a distinct copy of the lambda calculus.
- For each pair of modes $m, n \in M$, a set $m \to n$ of *modalities* between them. A modality $\mu : m \to n$ allows us to use types and terms of mode m in mode n. There might be multiple, distinct modalities $\mu, \nu : m \to n$.
- For every pair of modalities $\mu, \nu \in m \to n$ with matching domain and codomain, a set $\mu \Rightarrow \nu$ of *mode transformations* between them. A transformation $\alpha : \mu \Rightarrow \nu$ allows us to convert types and terms from under the modality μ to under the modality ν. There might be multiple, distinct transformations $\alpha, \beta : \mu \Rightarrow \nu$.

Concretely, this data should assemble into a 2-category, in the sense that identity modalities (of type $id_m : m \to m$) and identity transformations (of type $id_\mu : \mu \Rightarrow \mu$) exist; modalities and transformations can be composed if domain and codomain match up (we denote composition by $\mu \circ \nu$ and $\alpha \circ \beta$ respectively), and unitality and associativity laws hold as expected between these operations[5].

Definition 2 ([4, following Chapter 6.2])**.** Let $\mathcal{M}$ be a mode theory. The modal type theory $\mathrm{MTT}_{\mathcal{M}}$ is given by M copies of the simply typed lambda calculus, combined in the following way: For each mode $m \in M$, let $\Gamma \in \mathsf{Ctx}_m$, $A \in \mathsf{Type}_m$, $t \in \mathsf{Term}_m$ and $\Gamma \vdash_m t : A$ respectively denote the contexts, types, terms and typing judgements of the m'th copy of lambda calculus. The rules for simply typed $\mathrm{MTT}_{\mathcal{M}}$ are displayed in figure 1, 3 and 4

Contexts

Contexts of MTT (figure 1) are defined as follows:

- The rule for empty contexts Ctx-Empty is as usual.

- The context extension rule Ctx-Ext says that all variables in the context $\Gamma \in \mathsf{Ctx}_m$ have to exist at mode m, but they might originate at a different mode n and exist at m by way of a modality $\mu : n \to m$. This means that a variable in a context Γ is of the form $x : (A|\mu) \in \Gamma$, where A is its type, n is an arbitrary mode, and $\mu : n \to m$ is the modality which brought this variable into mode m. If the variable originates at the current mode, then $\mu = id_m : m \to m$.

- The rule Ctx-Restr allows us to restrict the use of variables to those which are under a particular modality. That is, starting with a context $\Gamma \in \mathsf{Ctx}_n$ and a modality $\mu : m \to n$, the restricted contex $\Gamma.\{\mu\} \in \mathsf{Ctx}_m$ only allows projecting of a variable $x : (A|\nu) \in \Gamma$ if there is a transformation $\xi : \nu \Rightarrow \mu$. This is ensured by the corresponding variable rule Var-Restr.

Types

Types of MTT (figure 2) mirror standard types of simply typed lambda calculus, with the following differences:

- Let $\mu : m \to n$ be a modality. There is a *modal type contructor* Mod which allows us to to lift a type $A \in \mathsf{Type}_m$ to mode n: we write the resulting type as $\langle A|\mu\rangle \in \mathsf{Type}_n$. The introduction rule Mod-Intro enforces that terms of such a modal type may only depend on variables which are themselves under the μ modality. The elimination rule Mod-Elim allows us to assume that we have a variable $x : (A|\mu)$ in order to eliminate from $\langle A|\mu\rangle$.

- The function type contains a built-in modality out of ergonomic reasons: for types $A : \mathsf{Type}_m$, $B : \mathsf{Type}_n$ and a modality $\mu : m \to n$, there is a primitive type $(A|\mu) \to B$. It behaves exactly as a non-modal function type composed with a modal type, i.e., it is a more usable syntax for the type $\langle A|\mu\rangle \to B$.

Terms

Terms of MTT (figure 4) are different from terms of simply typed lambda calculus in the following ways:

- There are new terms for introduction and elimination of modal types (Mod-Intro and Mod-Elim).

- The function introduction rule (Fun-Intro) preserves the additional modality by putting it into the context, alongside the new variables' type.

- The function elimination rule (Fun-Intro) deals with its modality annotation by restricting the context with it.

- All other terms are mode-local and all occuring modalities are required to be identities.

$$\textsc{Ctx-Empty}\ \frac{}{\epsilon \in \mathsf{Ctx}_n} \qquad \textsc{Ctx-Ext}\ \frac{A \in \mathsf{Type}_m \quad \Gamma \in \mathsf{Ctx}_n \quad \mu : m \to n}{\Gamma.(A|\mu) \in \mathsf{Ctx}_n}$$

$$\textsc{Ctx-Restr}\ \frac{\Gamma \in \mathsf{Ctx}_n \quad \mu : m \to n}{\Gamma.\{\mu\} \in \mathsf{Ctx}_m}$$

Figure 1: Contexts in MTT

$$\textsc{Mod}\ \frac{\mu : m \to n \quad A \in \mathsf{Type}_m}{\langle A|\mu\rangle \in \mathsf{Type}_n} \qquad \textsc{Fun}\ \frac{\mu : m \to n \quad A \in \mathsf{Type}_m \quad B \in \mathsf{Type}_n}{(A|\mu) \to B \in \mathsf{Type}_n}$$

$$\textsc{Prod}\ \frac{A \in \mathsf{Type}_m \quad B \in \mathsf{Type}_m}{A \times B \in \mathsf{Type}_m} \qquad \textsc{Sum}\ \frac{A \in \mathsf{Type}_m \quad B \in \mathsf{Type}_m}{A + B \in \mathsf{Type}_m}$$

Figure 2: Types in MTT

$$\textsc{Var-Zero}\ \frac{}{\mathsf{zero} : (A|\mu) \in_{\mathsf{id}} \Gamma.(A|\mu)} \qquad \textsc{Var-Suc}\ \frac{x : (A|\mu) \in_\nu \Gamma}{\mathsf{suc}\ x : (A|\mu) \in_\nu \Gamma.(B|\eta)}$$

$$\textsc{Var-Restr}\ \frac{x : (A|\mu) \in_\nu \Gamma}{x : (A|\mu) \in_{\eta\circ\nu} \Gamma.\{\eta\}}$$

Figure 3: De-Brujin variables with modality annotations in MTT

$$\textsc{Var}\ \frac{x : (A|\mu) \in_\nu \Gamma \qquad \xi : \mu \Rightarrow \nu}{\Gamma \vdash x^\xi : A}$$

$$\textsc{Mod-Intro}\ \frac{\Gamma.\{\mu\} \vdash_m t : A}{\Gamma \vdash_n \mathbf{mod}\ t : \langle A|\mu\rangle} \qquad \textsc{Mod-Elim}\ \frac{\Gamma \vdash_m t : \langle A|\mu\rangle \qquad \Gamma.(x : A|\mu) \vdash_m s : B}{\Gamma \vdash_m \mathbf{letmod}\ x \leftarrow t\ \mathbf{in}\ s : B}$$

$$\textsc{Fun-Intro}\ \frac{\Gamma.(x : A|\mu) \vdash_n t : B}{\Gamma \vdash_n \lambda x.t : (A|\mu) \to B} \qquad \textsc{Fun-Elim}\ \frac{\Gamma \vdash_n f : (A|\mu) \to B \qquad \Gamma \vdash_m x : A}{\Gamma \vdash_m f\ x : B}$$

$$\textsc{Prod-Intro}\ \frac{\Gamma \vdash_m a : A \qquad \Gamma \vdash_m b : B}{\Gamma \vdash_m (a\,,b) : A \times B} \qquad \textsc{Prod-Fst}\ \frac{\Gamma \vdash_m x : A \times B}{\Gamma \vdash_m \mathbf{fst}\ x : A} \qquad \textsc{Prod-Snd}\ \frac{\Gamma \vdash_m x : A \times B}{\Gamma \vdash_m \mathbf{snd}\ x : B}$$

$$\textsc{Sum-Left}\ \frac{\Gamma \vdash_m x : A}{\Gamma \vdash_m \mathbf{left}\ x : A + B} \qquad \textsc{Sum-Right}\ \frac{\Gamma \vdash_m x : B}{\Gamma \vdash_m \mathbf{right}\ x : A + B}$$

$$\textsc{Sum-Elim}\ \frac{\Gamma \vdash_m x : A + B \qquad \Gamma.(a : A|\mathsf{id}) \vdash_m t : C \qquad \Gamma.(b : B|\mathsf{id}) \vdash_m s : C}{\Gamma \vdash_m \mathbf{case}\ x\ \mathbf{of}\ \{\mathbf{left}\ a \mapsto t; \mathbf{right}\ b \mapsto s\} : C}$$

Figure 4: Typing rules for variables and types in MTT

3 Choreographic programming with MTT

The first, essential feature of modalities in MTT is that they restrict code availability: When constructing a term of type $\langle A|\mu\rangle$, only variables which are themselves under a μ modality can be used. This leads us naturally to the idea that we can use MTT for a distributed system with a set of participating roles ρ by introducing modalities $@r$ for each role $r \in \rho$. The type $\langle A|@r\rangle$ then is interpreted as "data of type A, located at role r".

The second feature of MTT is that transformations between modalities can be introduced in a controlled way. In order to allow a term at role r to be transformed into a term at role s, we simply have to introduce a transformation $\tau_{r,s} : @r \Rightarrow @s$ between the corresponding modalities. These transformations can be chosen freely: we might disallow communications between some nodes and include multiple channels between some others.

3.1 A mode theory for choreographic programming

However, a mode theory with only @-modalities and transformations $\tau_{r,s}$ is not enough to recover the full expressive power of Chorλ. In particular, expressing the **select** operator, which is used to notify other roles about decisions which happened locally at node r, requires the following additional features:

1. We need modalities expressing the fact that some data is "common knowledge" of multiple roles. We do so by allowing arbitrary conjunctions $r_1 \wedge \ldots \wedge r_k$ in the modality. For example, the type $\langle \mathbb{N}|@(r \wedge s)\rangle$ expresses the fact that there is a natural number which is known by both roles r and s.

2. We need roles to reference data that is about to be sent to other nodes. For this we use a new modality $\Box$[1], which allows roles to reference global choreographies locally. That is, if A is a global choreography type, then $t : \langle A|\Box\rangle$ is a local term, containing a quoted representation of such a choreography.

We define our mode theory with @ and $\Box$ modalities and common knowledge locations as follows:

Definition 3. Let ρ be a set of roles and Loc_ρ be the freely generated meet-semilattice on ρ. The mode theory $\mathcal{M}^\rho_{\mathrm{Chor}}$ is defined as follows:

- There are two modes: $\circ$ and $\triangle$. The global mode $\circ$ represents the global perspective on a choreography, encompassing computations and data occuring at all roles. The local mode $\triangle$ represents the perspective of a single location (which might be the conjunction of multiple roles) participating in the choreography.

[1] Usually this symbol is pronounced "box", but we also refer to it by "quote".

- For each location $u \in \mathrm{Loc}_\rho$ there is a modality $@u : \triangle \to \circ$. Each of these modalities represents a different way of how a local computation can be embedded in the global system. Concretely, $@u$ expresses that the computation exists at location u.
- An additional modality $\Box : \circ \to \triangle$, allowing global computations to be referenced locally.
- For each location $u \in \mathrm{Loc}_\rho$, a transformation $\mathsf{prepare}_u : \mathsf{id}_\triangle \Rightarrow (@u; \Box)$. This transformation allows a process to prepare a local term to be evaluated by an arbitrary role u. It happens in local mode ($\triangle$) and its interpretation involves no communication.
- For each location $v \in \mathrm{Loc}_\rho$, a transformation $\mathsf{eval}_v : (\Box; @v) \Rightarrow \mathsf{id}_\circ$. This transformation is the only one involving communication between roles. It describes that a global choreography, available in quoted form at location v, can be scheduled to be executed by all involved roles.
- For each pair of locations $u, v \in \mathrm{Loc}_\rho$ with $u \leq v$, a transformation $\mathsf{narrow}_{u,v} : @u \Rightarrow @v$.
- Additional equalities governing the interactions of the transformations. For instance, composing $\mathsf{prepare}_u$ and eval_u for the same role u is equal to the identity transformation since it represents a process communicating with itself.

Corollary 1. *In $\mathcal{M}^\rho_{\mathrm{Chor}}$, it is possible to recover a transformation $\tau_{u,v} : @u \Rightarrow @v$ by composing $\mathsf{prepare}_v$ and eval_u. Additionally, a process communicating with itself is equal to the identity, $\tau_{u,u} = \mathsf{id}_u$.*

4 Kami: An MTT based language for choreographic programming

Our mode theory expresses the interactions between participating roles in a distributed system, but standard MTT instantiated with $\mathcal{M}^\rho_{\mathrm{Chor}}$ is not suitable to be used as a choreographic programming language. The problem is that there is no notion of deferred transformations: transformations are always pushed down the syntax tree as far as possible and only recorded at the variables. This means that in order to control the communication behaviour of terms, we need to introduce a dedicated term for not yet executed communications. In our semantics only eval_v involves communications, so we simply add a dedicated term representing it. Its typing and reduction rules are displayed in figure 5.

Definition 4. Let $\mathrm{Chor}_{\mathrm{MTT}}$ be the type theory obtained by:

1. Initializing simply typed MTT with $\mathcal{M}^\rho_{\mathrm{Chor}}$.

$$\textsc{Let-Eval}\quad \frac{\Gamma \vdash t : \langle A|\Box;@v\rangle \qquad \Gamma.(x : A|\mathsf{id}_\circ) \vdash s : B}{\Gamma \vdash \textbf{leteval}_v\ x \leftarrow t\ \textbf{in}\ s : B}$$

$$\textsc{Let-Eval-}\beta\quad \frac{}{\Gamma \vdash \textbf{leteval}_v\ x \leftarrow t\ \textbf{in}\ s \rightsquigarrow \Gamma \vdash \textbf{mod}\ y \leftarrow t\ \textbf{in}\ s[x/y^{\mathsf{eval}_v}]}$$

Figure 5: Typing and reduction rule for deferred transformations.

2. Extending it with the Let-Eval rule.
3. Restricting the reduction relation analogously how Chorλ restricts the reduction relation of simply typed lambda calculus.

4.1 Behavioural semantics and relation with Chorλ

In order to allow for out-of-order execution of of independent processes, we can use the same machinery as Chorλ: restricting reduction rules to ensure that communications between a pair of processes always occur in the same order and an additional rewriting relation which allows for independent processes to evaluate their terms independently.

The expressivity of Chorλ arises from the two primitives involved in process interaction: **com** and **select**. Communication is easily reproduced in Chor_{MTT} by using prepare and eval.

Example 1. In Chor_{MTT}, we can define a function $\mathsf{com}_{A,u,v} : \langle A|@u\rangle \to \langle A|@v\rangle$ for each local type A and pair of locations u, v.

$$\begin{aligned}\mathsf{com}_{A,u,v}\ a = \ &\textbf{letmod}\ a_1 \leftarrow a\\ &\textbf{leteval}_u\ a_2 \leftarrow \textbf{mod}\ a_1^{\mathsf{prepare}_v * \mathsf{id}_{@u}}\\ &\textbf{in mod}\ a_2^{\mathsf{id}}\end{aligned}$$

Selection in Chorλ works as follows: a process at role r chooses its future behaviour based on locally available data and afterwards communicates its choice using **select** statements to those processes which need to be aware of this choice. In Chor_{MTT} the same functionality is available, but has to be stated in reverse order. First the required data for choosing future behaviour is communicated from r to all roles which need to be aware of this choice. After receiving the data, all relevant processes synchronously decide their future behaviour.

Example 2. Let A, B be local types, and Z a global type. A function of the following type can be derived in Chor_{MTT}:

$$\mathsf{choice}_{A,B,Z,r} : \langle A + B | @r \rangle \to (\langle A | @r \rangle \to Z) \to (\langle B | @r \rangle \to Z) \to Z$$

Note that since Z is global, a value $z : Z$ is a choreography involving possibly all roles[2]. The semantics of choice is: Given the knowledge of $A + B$ at role r, a global behaviour $z : Z$ can be chosen for all roles, with the additional knowledge of either the value $a : A$ or $b : B$ at role r. The function can be implemented in such a way that only the information regarding which branch is going to be chosen is communicated from r to other processes, the actual value of a or b stays at r.

The interaction of the $\Box$ and $@r$ modalities are key to the definition of choice. In [4] such a function which allows induction "from under a modality" is called *crisp induction principle*.

4.2 Categorical semantics

Following Gratzer, a categorical model for $\text{MTT}_{\mathcal{M}}$ is given by a functor $F : \mathcal{M}^{\mathsf{coop}} \to \mathbf{Cat}$, with additional conditions ensuring that F supports all type formers and their terms. This definition entails, for each modality $\mu : m \to n$, a functor $F(\mu) : F(n) \to F(m)$ modeling the *contravariant* context restriction operation $-.\{\mu\} : \mathsf{Ctx}_n \to \mathsf{Ctx}_m$. Additionally, the existence of modal types in Gratzer's model implies the existence of a further *covariant* functor $M_\mu : F(m) \to F(n)$, such that $F(\mu)$ and M_μ are "adjoint" in an appropriate sense[3].

While such a generic Model of MTT is required for Gratzer's goals, our intended semantics, in particular the fact that we want to compile Kami programs into real-world executables leads us to considering a less general, but more specifically useful class of models.

In the following we give the definition of our special class of models and sketch how $\text{MTT}_{\mathcal{M}^{\rho}_{\text{Chor}}}$, i.e., the underlying type theory of Kami can be interpreted in them.

Definition 5. Let $\mathcal{M}$ be a mode theory. A *covariant model* for simply typed $\text{MTT}_{\mathcal{M}}$ is given by the following data:

1. A category $\mathcal{C}$ representing the compilation target.
2. Closure of $\mathcal{C}$ under all type and term formers of MTT.
3. A (covariant) 1-functor $G : \mathcal{M} \to \mathbf{Cat}$ modeling the semantics of individual modes, and the modal types between them.

[2] For the sake of brevity our mode theory $\mathcal{M}^{\rho}_{\text{Chor}}$ as defined in this paper does not track which roles are actually involved in a given term of global type. It can be done though by extending the mode theory with a family of global modes.

[3] In the dependently typed MTT of Gratzer the exact statement is that M_μ is a *dependent right adjoint*, but we can simplify the condition somewhat in our simply typed case.

4. A family of 1-functors $\iota_m : G(m) \to \mathcal{C}$, where $m \in \mathcal{M}$, describing how the category $G(m)$ is represented in the compilation target category.

5. For each transformation $\alpha : \mu \Rightarrow \nu \in \mathcal{M}$, a natural transformation $\tau_\alpha : \iota_m \circ G(\mu) \Rightarrow \iota_m \circ G(\nu)$ in the target category.

Such a covariant model is a special case of a contravariant model in the sense of Gratzer; we claim:

Conjecture 1. *A covariant model of $MTT_{\mathcal{M}}$ can be assembled into a contravariant model, by freely adjoining context restriction operators.*

For our concrete use-case, we intend to obtain a covariant model in the following way: Let ρ be a finite set of roles and let **STLC** be the syntax category of the simply typed lambda calculus with sum types. Viewing the set ρ as a discrete category, we denote by $\mathbf{STLC}^\rho$ the functor category $\rho \to \mathbf{STLC}$. That is, an object $(\Gamma_i)_{i\in\rho}$ is a ρ-indexed family of **STLC**-contexts and a morphism $(\Gamma_i)_{i\in\rho} \to (\Delta_i)_{i\in\rho}$ is given by a ρ-indexed family of substitutions $(\sigma_i : \Gamma_i \to \Delta_i)_{i\in\rho}$. The intuition is that $\mathbf{STLC}^\rho$ describes the category of ρ processes running independently of each other, with no way to interact. To further add synchronous communication, we freely adjoin arrows axiomatizing such. To properly express these arrows, we need the following definition:

Definition 6. Let $i \in \rho$ be a role, define

$$\delta_i(-) : \mathbf{STLC} \to \mathbf{STLC}^\rho$$
$$X \mapsto j \mapsto \begin{cases} j = i & \implies X \\ j \neq i & \implies 1 \end{cases}$$

to be the function mapping an object $X \in \mathbf{STLC}$ to a family X_j, whose i'th component is X, and all other components are the terminal object.

Definition 7. Define

$$[-] : \mathbf{STLC}^\rho \to \mathbf{STLC}$$
$$(Y_j)_{j\in\rho} \mapsto \prod_{j\in\rho} Y_j$$

to be the function mapping a family $(Y_j)_{j\in\rho} \in \mathbf{STLC}^\rho$ to the product of its components.

With these definitions in hand, we can represent the type of a hypothetical communication operation which communicates a global state $X \in \mathbf{STLC}^\rho$ from all processes to a single process $i \in \rho$ as follows:

$$\mathsf{com}_{X,i} : X \to \delta_i([X])$$

.

Definition 8. Let the *category of synchronously interacting processes*, $\mathbf{SyncIntProc}_{\mathcal{C}} = \mathbf{STLC}^{\rho}[\mathsf{com}]$ be defined as the category $\mathbf{STLC}^{\rho}$ of independent processes with freely adjoined arrows of the shape $\mathsf{com}_{X,i} : X \to \delta_i([X])$ for any $X \in \mathbf{STLC}^{\rho}$ and $i \in \rho$.

Theorem 1. *Let $\mathcal{C}$ be a cartesian closed category. There is a covariant model of $MTT_{\mathcal{M}^{\rho}_{Chor}}$ which has $\boldsymbol{SyncIntProc}_{\mathcal{C}}$ as its compilation target category.*

In particular, in this model, only the transformation $\mathsf{eval}_v : (\Box; @v) \Rightarrow \mathsf{id}_{\circ}$ is built up from the freely adjoined com arrows, since it is the only one involving communication. All other transformations are modeled by arrows already existing as part of the cartesian closed structure of each processes' individual category $\mathcal{C}$.

Corollary 2. *There is a translation function from the syntax category of $MTT_{\mathcal{M}^{\rho}_{Chor}}$ to $\boldsymbol{SyncIntProc}_{\mathcal{C}}$.*

In other words, this gives us a compilation procedure for Kami programs into any target language with cartesian closed categorical semantics and synchronous communication primitives.

References

[1] Luís Cruz-Filipe, Eva Graversen, Lovro Lugović, Fabrizio Montesi, and Marco Peressotti. Functional choreographic programming. In *International Colloquium on Theoretical Aspects of Computing*, pages 212–237. Springer, 2022.

[2] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, 2001.

[3] Saverio Giallorenzo, Fabrizio Montesi, and Marco Peressotti. Choral: Object-oriented choreographic programming. *ACM Transactions on Programming Languages and Systems*, 46(1):1–59, 2024.

[4] Daniel Gratzer. *Syntax and semantics of modal type theory.* PhD thesis, Aarhus University, 2023.

[5] Daniel R Licata and Michael Shulman. Adjoint logic with a 2-category of modes. In *Logical Foundations of Computer Science: International Symposium, LFCS 2016, Deerfield Beach, FL, USA, January 4-7, 2016. Proceedings*, pages 219–235. Springer, 2016.

[6] Tom Murphy, VII. *Modal types for mobile code.* PhD thesis, Carnegie Mellon University, 2008.