Kami: A modal type theory for distributed systems

Maxim Urschumzew Miëtek Bak

April 17, 2024

We present Kami, a programming language for distributed systems based on Modal Type Theory[4]. In particular, we describe a mode theory that 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 a categorical semantics that gives rise to a generic compilation procedure for Kami.

1 Introduction

Modalities provide a way to extend a type theory with domain-specific type constructors, which can be used to track runtime properties of code directly in the type system. For example, modalities have been used to represent unevaluated code for metaprogramming and staged computation[2], and 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 in choreographic programming 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 based on top of a simply typed lambda calculus (STLC) with modalities, a variant of Modal Type Theory[4]. This requires the introduction of two new concepts: common knowledge between multiple participants, and local knowledge of global computations.

2 Preliminaries

2.1 Chorλ

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

2.2 Simply typed modal type theory

Modal Type Theory (MTT)[4], introduced by Gratzer in his PhD dissertation, is a framework for constructing modal type theories, with Martin-Löf type theory at its core. The framework is parametrized by a system of modalities specified in the form of a *mode theory*. MTT includes so-called "full-spectrum" dependent types, but we restrict ourselves to a simply typed fragment of MTT.

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 an extension of STLC.
- For each pair of modes $m, n \in M$, a set $m \to n$ of *modalities* between the modes. 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 domains and codomains, a set $\mu \Rightarrow \nu$ of *transformations* between the modalities. 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 forms a 2-category, in the sense that identity modalities (of type $\mathsf{id}_m : m \to m$) and identity transformations (of type $\mathsf{id}_\mu : \mu \Rightarrow \mu$) exist, modalities and transformations can be composed (denoted by $\mu \fatsemi \nu$ and $\alpha \fatsemi \beta$ respectively) if the domains and codomains match, and identity and associativity laws hold as expected[5].

Definition 2 ([4, following Chapter 6.2])**.** Let $\mathcal{M}$ be a mode theory. A *simply typed modal type theory* $\mathrm{MTT}_{\mathcal{M}}$ is given by $|M|$ copies of an extension of STLC, combined as follows: For each mode $m \in M$, let $\Gamma \in \mathsf{Ctx}_m$, $A \in \mathsf{Type}_m$, $t \in \mathsf{Term}_m$, and $\Gamma \vdash_m t : A$ denote the contexts, types, terms, and typing judgements of the mth copy of an extension of STLC respectively.

The rules of $\mathrm{MTT}_{\mathcal{M}}$ are displayed in figures 1, 2, 3, and 4.

Well moded types

Types of $\mathrm{MTT}_{\mathcal{M}}$ (figure 1) mirror the standard types of STLC, with the following differences:

- A *modal type* $\langle A|\mu\rangle \in \mathsf{Type}_n$ is formed by lifting a type $A \in \mathsf{Type}_m$ to mode n using a modality $\mu : m \to n$.
- A *modal function type* $(A|\mu) \to B \in \mathsf{Type}_n$ is formed for each type $A : \mathsf{Type}_m$, $B : \mathsf{Type}_n$, and modality $\mu : m \to n$. This type is a convenience that behaves as the standard function type composed with a modal type. We recover the standard function type using the identity modality $\mathsf{id}_m : m \to m$.

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

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

Figure 1: Well moded types of $\mathrm{MTT}_{\mathcal{M}}$

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

Figure 2: Well moded contexts of $\mathrm{MTT}_{\mathcal{M}}$

Well moded contexts

Contexts of $\mathrm{MTT}_{\mathcal{M}}$ (figure 2) are defined as follows:

- The empty context rule Ctx-Empty.
- The context extension rule Ctx-Ext says that all assumptions in a context $\Gamma \in \mathsf{Ctx}_m$ must exist at a mode m, but they may originate at a different mode n by way of a modality $\mu : n \to m$. This means that an assumption in a context Γ is of the form $(A|\mu) \in \Gamma$, where A is a type, n is an arbitrary mode, and $\mu : n \to m$ is a modality that brings the assumption into mode m. If the assumption originates at the current mode, then $\mu = \mathsf{id}_m : m \to m$.
- The context restriction rule Ctx-Restr allows us to restrict the use of assumptions to those under a particular modality. That is, starting with a context $\Gamma \in \mathsf{Ctx}_n$ and a pair of modalities $\mu, \nu : m \to n$, the restricted context $\Gamma.\{\nu\} \in \mathsf{Ctx}_m$ only allows referring to the assumption $(A|\mu) \in \Gamma$ given a transformation $\alpha : \mu \Rightarrow \nu$. This is ensured by the variable rule Var (figure 4).

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

Figure 3: Well typed variables (de Bruijn indices) of $\mathrm{MTT}_{\mathcal{M}}$

Well typed terms

Terms of $\mathrm{MTT}_{\mathcal{M}}$ (figure 4) differ from terms of STLC as follows:

- There are new terms for introduction and elimination of modal types. The introduction rule Mod-Intro enforces that terms of a modal type $\langle A|\mu\rangle$ may only depend on variables that are themselves under the μ modality. The elimination rule Mod-Elim allows us to assume $x : (A|\mu)$ in order to eliminate a value of such a modal type.
- The function type introduction rule Fun-Intro preserves the additional modality by putting it into the context, alongside the new variables' type.
- The function type 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.

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

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

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

$$\frac{\Gamma \vdash_m s : A \qquad \Gamma \vdash_m t : B}{\Gamma \vdash_m s\, \boldsymbol{,}\, t : A \times B}\ \textsc{Prod-Intro} \qquad \frac{\Gamma \vdash_m t : A \times B}{\Gamma \vdash_m \textsf{fst}\ t : A}\ \textsc{Prod-Elim}_1 \qquad \frac{\Gamma \vdash_m t : A \times B}{\Gamma \vdash_m \textsf{snd}\ t : B}\ \textsc{Prod-Elim}_2$$

$$\frac{\Gamma \vdash_m t : A}{\Gamma \vdash_m \textsf{left}\ t : A + B}\ \textsc{Sum-Intro}_1 \qquad \frac{\Gamma \vdash_m t : B}{\Gamma \vdash_m \textsf{right}\ t : A + B}\ \textsc{Sum-Intro}_2$$

$$\frac{\Gamma \vdash_m s : A + B \qquad \Gamma.(x : A|\mathsf{id}_m) \vdash_m t : C \qquad \Gamma.(y : B|\mathsf{id}_m) \vdash_m u : C}{\Gamma \vdash_m \textsf{case}\ s\ \textsf{of left}\ x \mapsto t\, \boldsymbol{;}\, \textsf{right}\ y \mapsto u : C}\ \textsc{Sum-Elim}$$

Figure 4: Well typed terms of $\mathrm{MTT}_{\mathcal{M}}$

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 that 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 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 that 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 that 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 $\square$[1], which allows roles to reference global choreographies locally. That is, if A is a global choreography type, then $t : \langle A|\square\rangle$ is a local term, containing a quoted representation of such a choreography.

We define our mode theory with @ and $\square$ 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: $\bigcirc$ and $\triangle$. The global mode $\bigcirc$ 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.

- For each location $u \in \mathrm{Loc}_\rho$ there is a modality $@u : \triangle \to \bigcirc$. Each of these modalities represents a different way of how a local computation can be embedded

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

in the global system. Concretely, $@u$ expresses that the computation exists at location u.

- An additional modality $\Box : \bigcirc \to \triangle$, 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 $\mathsf{eval}_v : (\Box; @v) \Rightarrow \mathsf{id}_\bigcirc$. 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 $\mathcal{M}^\rho_{\mathrm{Chor}}$.
2. Extending it with the Let-Eval rule.
3. Restricting the reduction relation analogously how Chorλ restricts the reduction relation of STLC.

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

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

Figure 5: Typing and reduction rule for deferred transformations.

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 that 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$_{\mathrm{MTT}}$ by using $\mathsf{prepare}$ and eval.

Example 1. In Chor$_{\mathrm{MTT}}$, 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 that need to be aware of this choice. In Chor$_{\mathrm{MTT}}$ 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 that 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$_{\mathrm{MTT}}$:

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

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

[2]For the sake of brevity our mode theory $\mathcal{M}^{\rho}_{\mathrm{Chor}}$ 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.

$z : Z$ 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 that allows induction "from under a modality" is called a *crisp induction principle*.

4.2 Categorical semantics

Following Gratzer, a categorical model for $\mathrm{MTT}_{\mathcal{M}}$ 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 $\mathrm{MTT}_{\mathcal{M}^{\rho}_{\mathrm{Chor}}}$, 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 $\mathrm{MTT}_{\mathcal{M}}$ 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.
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 $\mathrm{MTT}_{\mathcal{M}}$ can be assembled into a contravariant model, by freely adjoining context restriction operators.*

[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.

For our concrete use-case, we intend to obtain a covariant model as follows:

Let ρ be a finite set of roles and let **STLC** be the syntax category of STLC 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 ith 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 that 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 $MTT_{\mathcal{M}^\rho_{Chor}}$ that has $\boldsymbol{SyncIntProc}_{\mathcal{C}}$ as its compilation target category.*

In particular, in this model, only the transformation $\mathsf{eval}_v : (\Box; @v) \Rightarrow \mathsf{id}_{\bigcirc}$ 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. URL https://doi.org/10.1007/978-3-031-17715-6_15.

[2] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, 2001. URL https://doi.org/10.1145/382780.382785.

[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. URL https://doi.org/10.1145/3632398.

[4] Daniel Gratzer. *Syntax and semantics of modal type theory*. PhD thesis, Aarhus University, 2023. URL https://www.danielgratzer.com/papers/phd-thesis.pdf.

[5] Daniel R. Licata and Michael Shulman. Adjoint logic with a 2-category of modes. In Sergei Artemov and Anil Nerode, editors, *Logical Foundations of Computer Science*, pages 219–235. Springer, 2016. URL https://doi.org/10.1007/978-3-319-27683-0_16.

[6] Tom Murphy, VII. *Modal types for mobile code*. PhD thesis, Carnegie Mellon University, 2008. URL http://tom7.org/papers/modal-types-for-mobile-code.pdf.