Kami: A modal type theory for distributed systems

Maxim Urschumzew Miëtek Bak

April 18, 2024

We present Kami, a programming language for distributed systems, based on Modal Type Theory. In particular, we describe a mode theory that is suitable for distributed programming, and show how the primitives of Chorλ, 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 programs directly in the type system. For example, modalities have been used to represent unevaluated programs for meta-programming[1] and staged computation[3], and to encode the notion of location in ML5, a language for distributed systems[7]. In fact, the "`at`" modality of ML5 serves the same purpose as the location annotations in choreographic programming languages[4], and it stands to reason that existing research on modalities could provide a stepping stone for the design of type systems for such languages.

In this paper, we explore how Chorλ can be based on top of a simply typed lambda calculus with modalities, a variant of Modal Type Theory. This requires introducing two new concepts: common knowledge between multiple participants, and the representation of global choreographies as values local to individual participants.

2 Preliminaries

2.1 Chorλ

Chorλ is a functional choreographic programming language introduced by Cruz-Filipe et al.[2], and extends the simply typed lambda calculus (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), introduced by Gratzer in his PhD thesis[5], 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 ([5, §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 may 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 may be multiple distinct transformations $\alpha, \beta : \mu \Rightarrow \nu$.

Concretely, these data form 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[6].

Definition 2 ([5, following §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.

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 $\mathsf{id}_m : m \to m$.

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

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

Figure 1: Types of $\mathrm{MTT}_{\mathcal{M}}$

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

Figure 2: Contexts of $\mathrm{MTT}_{\mathcal{M}}$

Contexts

Contexts of $\mathrm{MTT}_{\mathcal{M}}$ (figure 2) are defined in the following way:

- The empty context rule Ctx-Empty.
- The context extension rule Ctx-Ext says that all types 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 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 type 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 types 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).

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

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

$$\textsc{Var}\ \frac{\mu, \nu : m \to n \qquad \alpha : \mu \Rightarrow \nu \qquad x : (A|\mu) \in_\nu \Gamma}{\Gamma \vdash_n x^\alpha : A} \qquad \textsc{Mod-Intro}\ \frac{\mu : m \to n \qquad \Gamma.\{\mu\} \vdash_m t : A}{\Gamma \vdash_n \mathbf{mod}_\mu\ t : \langle A|\mu\rangle}$$

$$\textsc{Mod-Elim}\ \frac{\mu : m \to n \qquad \nu : n \to o \qquad \Gamma.\{\nu\} \vdash_n s : \langle A|\mu\rangle \qquad \Gamma.\, x : (A|\mu \fatsemi \nu) \vdash_o t : B}{\Gamma \vdash_o \mathbf{let}_\nu\ \mathbf{mod}_\mu\ x \leftarrow s\ \mathbf{in}\ t : B}$$

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

$$\textsc{Fun-Elim}\ \frac{\mu : m \to n \qquad \Gamma \vdash_n t : (A|\mu) \to B \qquad \Gamma \vdash_n s : A}{\Gamma \vdash_n t\, s : B}$$

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

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

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

Figure 4: Terms of $\mathrm{MTT}_{\mathcal{M}}$

Terms

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

- The modality introduction rule Mod-Intro ensures that terms of a modal type $\langle A|\mu\rangle$ may only depend on variables that are themselves under a μ modality. Given a value of such type, the modality elimination rule Mod-Elim allows us to assume $(A|\mu)$. The *framing modality* ν allows the type A to exist at a different mode than the motive B[5, §6.1.9]. We omit this modality when it is the identity.
- The terms for introduction and elimination of the function type reflect that the type is endowed with an additional modality.
- All other terms involve a single mode only, and any modalities that occur in the terms are the identity.

3 Choreographic programming with MTT

The first, essential feature of modalities in MTT is that they restrict program 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 participants 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 must introduce a transformation $\tau_{r,s} : @r \Rightarrow @s$ between the corresponding modalities. These transformations can be chosen freely: we may disallow communications between some roles, and include multiple channels between other roles.

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 role r, requires the following additional features:

1. We need modalities expressing the fact that some data are common knowledge between 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. Furthermore, a meet-semilattice structure arises: e.g., it holds that $r \wedge s \leq r$, since the knowledge available at $r \wedge s$ is a subset of the knowledge at r.

2. We need roles to reference data that are about to be sent to other roles. For this, we use a new modality $\Box$, 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 that occur at all roles. The *local mode* $\triangle$ represents the perspective of a single location, which may be a conjunction of multiple roles, that participates in the choreography.

- For each location $u \in \mathrm{Loc}_\rho$, a *location 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 *quotation modality* $\Box : \bigcirc \to \triangle$, which allows global computations to be referenced locally.
- For each location $v \in \mathrm{Loc}_\rho$, a *quotation transformation* $\mathsf{quote}_v : \mathsf{id}_\triangle \Rightarrow (@v \fatsemi \Box)$, which allows a role to quote a local term to be evaluated by an arbitrary role v. This transformation happens in local mode $\triangle$, and its interpretation involves no communication.
- For each location $u \in \mathrm{Loc}_\rho$, an *evaluation transformation* $\mathsf{eval}_u : (\Box \fatsemi @u) \Rightarrow \mathsf{id}_\bigcirc$, which is the only one involving communication between roles. This transformation describes that a global choreography, available in quoted form at u, can be scheduled to be performed by all involved roles.
- For each pair of locations $u, v \in \mathrm{Loc}_\rho$ with $u \leq v$, a *narrowing transformation* $\mathsf{narrow}_{u,v} : @u \Rightarrow @v$. The intuition is that if u is a sublocation of v, then information available at the former is also available at the latter.
- Additional equalities governing the interactions of the transformations. For instance, composing quote_u and eval_u for the same role u is equal to the identity transformation, since it represents a role communicating with itself.

Corollary 4. *In $\mathcal{M}^\rho_{\mathrm{Chor}}$, it is possible to recover a transformation $\tau_{u,v} : @u \Rightarrow @v$ by composing quote_v and eval_u. Additionally, a role communicating with itself is equal to the identity transformation $\tau_{u,u} = \mathsf{id}_{@u}$.*

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 MTT has no notion of *deferred transformations,* because 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 communications that have not been performed yet.

In our semantics, only eval_v involves communications, so the only rule we need to add is Let-Eval (figure 5).

Definition 5. Let $\mathrm{Chor}_{\mathrm{MTT}}$ be the type theory obtained by:

1. Instantiating simply typed MTT with $\mathcal{M}^\rho_{\mathrm{Chor}}$.
2. Extending it with the Let-Eval rule.
3. Restricting the reduction relation analogously to how Chorλ restricts the reduction relation of STLC.

$$\textsc{Let-Eval}\quad \frac{\nu : \bigcirc \to m \qquad \Gamma.\{\nu\} \vdash_{\bigcirc} s : \langle A|\square \fatsemi @u\rangle \qquad \Gamma.\, x : (A|\nu) \vdash_m t : B}{\Gamma \vdash_m \mathbf{let}_\nu\ \mathbf{eval}_u\ x \leftarrow s\ \mathbf{in}\ t : B}$$

$$\textsc{Let-Eval-}\beta\quad \frac{}{\Gamma \vdash_m \mathbf{let}_\nu\ \mathbf{eval}_u\ x \leftarrow s\ \mathbf{in}\ t \implies \Gamma \vdash_m \mathbf{let}_\nu\ \mathbf{mod}_{(\square \fatsemi @u)}\ y \leftarrow s\ \mathbf{in}\ t[y^{\mathsf{eval}_u}/x]}$$

Figure 5: Typing and reduction rules for deferred transformations

4.1 Informal semantics and relation to Chorλ

In order to allow for out-of-order execution of of independent processes, we can use the same machinery as Chorλ: we can restrict reduction rules to ensure that communications between a pair of processes always occur in the same order, and add 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**. Communicating data is done using **com**, and can be recovered in Chor_{MTT} using the Let-Eval rule in the following way:

Example 6. 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.

$$\begin{aligned}\mathsf{com}_{A,u,v}\ x = \ & \mathbf{let}\ \mathbf{mod}_{@u}\ y \leftarrow x \\ & \mathbf{let}\ \mathbf{eval}_u\ z \leftarrow \mathbf{mod}_{@v}\ (\mathbf{mod}_{(\square \fatsemi @u)}\ y^{\mathsf{quote}_v \circledast \mathsf{id}}) \\ & \mathbf{in}\ z^{\mathsf{id}}\end{aligned}$$

In Chorλ, **select** is used to communicate choice of branching. This 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_{MTT}, the same functionality is available, but must be stated in reverse order. First, the required data for choosing future behaviour are 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 7. Let A, B be local types, and Z a global type. A function of the following type can be defined 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 potentially involving all roles.[1] The semantics of choice is: Given the knowledge of $A + B$ at role r, a global

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

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. Gratzer refers to functions that allow induction "from under a modality" as *crisp induction principles*[5, §6.3.1].

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 types and 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.[2]

While such a generic model of MTT is required for Gratzer's goals, our intended semantics, and in particular, the fact that we want to compile Kami programs into real-world executables, leads us to consider 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 8. 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 target category.
5. For each transformation $\alpha : \mu \Rightarrow \nu \in \mathcal{M}$, a natural transformation in the target category, of type $\tau_\alpha : \iota_m \circ G(\mu) \Rightarrow \iota_m \circ G(\nu)$.

Such a covariant model is a special case of a contravariant model in the sense of Gratzer. Specifically, we claim that:

Conjecture 9. *A covariant model of* $\mathrm{MTT}_{\mathcal{M}}$ *can be assembled into a contravariant model, by freely adjoining context restriction operators.*

[2] 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 system.

For our purposes, 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.

Next, we freely adjoin arrows that axiomatize synchronous communication between the processes. To properly express these arrows, we need the following definition:

Definition 10. Let $i \in \rho$ be a role. Define

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

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 11. 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 12. Let the *category of synchronously interacting processes* $\mathbf{SIP}_\rho$ 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 13. *Let $\mathcal{C}$ be a cartesian closed category. There is a covariant model of* $\mathrm{MTT}_{\mathcal{M}^\rho_{\mathrm{Chor}}}$ *that has* $\mathbf{SIP}_\rho$ *as its compilation target category.*

In particular, in this model, only the transformation $\mathsf{eval}_v : (\Box \fatsemi @v) \Rightarrow \mathsf{id}_\bigcirc$ 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 $\mathbf{STLC}^\rho$.

Corollary 14. *There is a translation function from the syntax category of* $\mathrm{MTT}_{\mathcal{M}^\rho_{\mathrm{Chor}}}$ *to* $\mathbf{SIP}_\rho$.

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.

Acknowledgements

The Kami programming language[3] has been envisioned and initially developed together with Olivia Röhrig. The authors are grateful for many hours of debate on syntax and semantics.

This project is funded through NGI Zero Core,[4] a fund established by NLnet with financial support from the European Commission's Next Generation Internet program.

References

[1] Miëtek Bak. On self-interpreters for the $\lambda^{\Box}$-calculus and other modal λ-calculi. In *26th International Conference on Types for Proofs and Programs (TYPES 2020), Book of Abstracts*. Ed. by Ugo de' Liguoro and Stefano Berardi. 2020, pp. 133–135. https://types2020.di.unito.it/abstracts/BookOfAbstractsTYPES2020.pdf.

[2] Luís Cruz-Filipe, Eva Graversen, Lovro Lugović, Fabrizio Montesi, and Marco Peressotti. Functional choreographic programming. In *International Colloquium on Theoretical Aspects of Computing*. Springer, 2022, pp. 212–237. DOI: 10.1007/978-3-031-17715-6_15.

[3] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. In *Journal of the ACM* 48.3 (2001), pp. 555–604. DOI: 10.1145/382780.382785.

[4] Saverio Giallorenzo, Fabrizio Montesi, and Marco Peressotti. Choral: Object-oriented choreographic programming. In *ACM Transactions on Programming Languages and Systems* 46.1 (2024), pp. 1–59. DOI: 10.1145/3632398.

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

[6] Daniel R. Licata and Michael Shulman. Adjoint logic with a 2-category of modes. In *Logical Foundations of Computer Science*. Ed. by Sergei Artemov and Anil Nerode. Springer, 2016, pp. 219–235. DOI: 10.1007/978-3-319-27683-0_16.

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

[3] https://nlnet.nl/project/Kami/
[4] https://nlnet.nl/core