Epistemic Logic for Polyglots

Luis Garcia
Chris Martens
garcia.lui@northeastern.edu
c.martens@northeastern.edu
Northeastern University
Boston, USA

Abstract

A pattern in programming is to stitch together code from disparate programming languages to build a complex program. This pattern is a manifestation of multilanguage computation, a subject concerning the interoperation of languages in a greater system. We discuss ongoing work in developing a type system for multilanguage computation in the propositions-as-types discipline, drawing types from epistemic modal logic.

CCS Concepts: • **Theory of computation → Modal and temporal logics**.

Keywords: multilanguage computation, epistemic logic, modal logic, propositions-as-types

ACM Reference Format:
Luis Garcia and Chris Martens. 2025. Epistemic Logic for Polyglots. In *Proceedings of Partial Evaluation and Program Manipulation (PEPM '26).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

A pattern in programming is to stitch together code from disparate programming languages to build a complex program. For example, consider the three pieces of code in Figure 1. This is an illustration of a typical web tech stack, which includes a piece of frontend code, some middleware, and backend code. Each piece of code operates on its own runtime, but the overall system works by having the frontend send data to the backend via the middleware. Data may be sent back to the frontend in the form of a response.

This pattern is a manifestation of multilanguage computation, a subject concerning the interoperation of languages in a greater system. Many large-scale software systems are written in a variety of languages. There is a whole host of methods for formalizing this pattern of software development. We concern ourselves primarily with the one that we illustrated above: two (or more) languages interacting via a lightweight intermediary.

From our example, we can observe a challenge that we aim to address: the three languages may have differing data representations. It is the job of the middleware to convert one to the other. Writing code to do this can be arduous because the data schema in, say, the frontend, is not beholden to the requirements of the backend in a static way. Thus, programmers may find themselves finding out if their middleware does the job dynamically—that is, at system runtime. Testing for data compliance between the frontend and the backend, then, becomes a game of running the system, checking for errors, fixing things in the presence of errors, and then repeating. Surely, we can do better.

In this paper, we discuss ongoing work in developing a type system for this sort of multilanguage computation using a propositions-as-types discipline, where the propositions are drawn from *epistemic modal logic*, a family of logics for reasoning about the knowledge of principals. In our case, the principals in question are languages, and epistemic propositions are types for code. A computational interpretation of proofs in this logic give rise to a multilanguage.

We present $\lambda^{\Box K}$, a calculus of epistemic types. We begin with a "monolingual" treatment of the logic, derived from the well-known judgmental treatment of S4 by Pfenning and Davies [11]. We then generalize to a multilingual treatment of the logic, introducing proof terms and typing rules from which an operational semantics arises in the usual interaction between introduction and elimination forms. We close with our related and future work.

2 Background

We introduce the epistemic S4 type system with proof terms following the judgmental approach given by Pfenning and Davies [11]. Our presentation of the logic can be viewed as both an extension of the modal logic S4 developed by Pfenning and Davies, and as a translation of the sequent calculus given by Garg et al. [4].

In the logic developed by Pfenning and Davies, the $\Box$ modality represents validity (a.k.a., necessity), which can be

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PEPM '26, Rennes, France
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Listing (1) Frontend code that sends a POST request.

```
function sendMessage(){
    request({
        "method": "POST",
        "body": "Hello,"
    });
}
```

Listing (2) Middleware code that processes only POST requests.

```
onRequest :: Request
onRequest request =
    Request (Method POST) (Body msg)
        = send(msg)
    _ = Error
```

Listing (3) Backend code that responds to the message in the request.

```
def respond(message):
    if message = "Hello,":
        return "World!"
    else:
        return Error
```

Figure 1. A typical web tech stack written with three different languages. The frontend sends a request to the backend through some middleware. The backend responds in kind.

seen as a stronger form of truth. Specifically, valid judgments are ones that "do not depend on hypotheses about the truth of propositions" [11]. A computational interpretation of this modality found by Davies and Pfenning is a realization of staged computation, where valid propositions are interpreted as types for code [3].

We can generalize validity by asking to whom is a proposition valid. Such propositions can be stratified among principals (a.k.a., agents), yielding to us a notion of knowledge. Epistemic logic is a family of logics for reasoning about and characterizing such knowledge. The epistemic logic S4 is a generalization over the modal logic S4, where validity is indexed by principals. Thus, the interested reader may find that, aside from a couple of subtle differences, the system presented in this section is quite similar to that of Davies and Pfenning. For those who are not familiar with the judgmental approach to modal logic, we hope that this section acts as an adequate primer.

2.1 Syntax

For now, we consider the judgments A **true** and K **knows** A, and the types given by the grammar

$$\begin{array}{rrcl} \textit{Agent} & K, L & & \\ \textit{Proposition} & A, B, C & ::= & p \mid \Box_K A \end{array}$$

where p is an arbitrary proposition, and $\Box_K A$ reads "K knows A". We elide definitions for other standard propositions such as implication, conjunction, and disjunction.

We introduce two contexts as primitives in our proof theory: one containing hypotheses pertaining to truth, and one containing hypotheses pertaining to knowledge. They are defined according to the grammar

$$\begin{array}{rrcl} \textit{Truth Context} & \Delta & ::= & \cdot \mid \Delta, A\ \textbf{true} \\ \textit{Knowledge Context} & \Gamma & ::= & \cdot \mid \Gamma, K\ \textbf{knows}\ A. \end{array}$$

For terms, we have the grammar

$$\textit{Expression} \quad E \quad ::= x \mid \textbf{box}_K E \mid \textbf{letbox}_K\ x = E_1\ \textbf{in}\ E_2$$

where x is a variable, $\textbf{box}_K E$ is a "boxed" expression, and $\textbf{letbox}_K\ x = E_1\ \textbf{in}\ E_2$ will be used to "unbox" boxed expressions.

2.2 Typing Rules

From the logic angle, we concern ourselves with judgments of the form A **true** and K **knows** A, and the hypothetical judgment

$$\Gamma; \Delta \vdash C\ \textbf{true}.$$

This expresses that under hypotheses about agents' knowledge Γ and the truth Δ, we may conclude C is true.

From the other side of the Curry-Howard correspondence, we are interested in typing our expressions given at the end of Section 2.1. Thus, we annotate all of our propositions with terms. In Γ, we will generally have variable-type pairings of the form $u ::_K A$. In Δ, we will have pairings of the form $x : A$. Finally, our conclusion will be annotated with a general expression, allowing us to reintroduce our hypothetical judgment:

$$\Gamma; \Delta \vdash E : C.$$

Before getting to our typing rules, we must define one operation over the knowledge context. Intuitively, it is an operation that filters away knowledge variables from a context given a single agent.

Definition 2.1 (Knowledge Restriction)**.** Let Γ be a context consisting of only judgements of the form $u ::_K A$. Then, the *knowledge restriction* on Γ is defined inductively as follows:

$$\begin{array}{rcll} \cdot|_K & = & \cdot & \\ (\Gamma, u ::_K A)|_K & = & \Gamma|_K, u ::_K A & \\ (\Gamma, u ::_L A)|_K & = & \Gamma|_K & \text{if } L \neq K \end{array}$$

We may now define inference rules characterizing our notion of knowledge. We begin with the rules for $\Box_K$, given by the following introduction and elimination rules.

$$\frac{\Gamma|_K; \cdot \vdash E : A}{\Gamma; \Delta \vdash \mathbf{box}_K E : \Box_K A}\ \Box_K \text{ I}$$

$$\frac{\Gamma; \Delta \vdash E_1 : \Box_K A \qquad \Gamma, u ::_K A; \Delta \vdash E_2 : C}{\Gamma; \Delta \vdash \mathbf{letbox}_K\ u = E_1\ \mathbf{in}\ E_2 : C}\ \Box_K \text{ E}$$

Read from the bottom-up, the introduction rule ($\Box_K$ I) imposes that in order to prove that an agent knows a fact, we must only use the facts that the agent already knows. The truth context is empty to enforce this independence of agent knowledge. The knowledge restriction presents a departure from the modal logic S4. Whereas in that system, all valid hypotheses are allowed to be used in the derivation, in this one, only the knowledge of a single agent is permitted. The elimination rule ($\Box_K$ E) allows us to permanently reuse the fact that an agent knows a proposition in the scope of E_2.

Next, are our hypothesis rules.

$$\frac{}{\Gamma; \Delta, x : A \vdash x : A}\ \text{hyp} \qquad \frac{}{\Gamma, x ::_K A; \Delta \vdash x : A}\ \text{hyp}^*$$

The first hypothesis rule (hyp) is standard for natural deduction systems, and admits a uniform substitution principle. The second rule allows us to conclude that A is true if an agent knows A. This establishes that if an agent knows a fact, that fact must be true. This second rule admits a second substitution principle, which we state below.

Proposition 1 (Substitution Principle for Knowledge)**.** *If* $\Gamma|_K; \cdot \vdash E_1 : A$ *and* $\Gamma, u ::_K A; \Delta \vdash E_2 : C$*, then* $\Gamma; \Delta \vdash [E_1/u]E_2 : C$.

2.3 $\beta\eta$-Principles

Reduction rules and equivalence laws for the boxed expressions arise from the interaction between their elimination and introduction forms. First, we have a *local reduction*, which gives us a clue for what our β-reduction rule should be. This can be witnessed by introducing a boxed expression and then immediately eliminating it from the top-down. Thus, we have

$$\frac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma|_K; \cdot \vdash E_1 : A\end{array}}{\Gamma; \Delta \vdash \mathbf{box}_K E_1 : \Box_K A}\ \Box_K\text{I} \qquad \begin{array}{c}\mathcal{E}\\ \Gamma, u ::_K A; \Delta \vdash E_2 : C\end{array}}{\Gamma; \Delta \vdash \mathbf{letbox}_K\ u = \mathbf{box}_K E_1\ \mathbf{in}\ E_2 : C}\ \Box_K\text{E}$$

$$\Rightarrow_{\text{R}}$$

$$\begin{array}{c}\mathcal{F}\\ \Gamma; \Delta \vdash [E_1/u]E_2 : C,\end{array}$$

where $\mathcal{F}$ is given by the substitution principle applied to $\mathcal{D}$ and $\mathcal{E}$. Our β-reduction rule should then be

$$\frac{}{\mathbf{letbox}_K\ u = \mathbf{box}_K E_1\ \mathbf{in}\ E_2 \mapsto [E_1/u]E_2}\ \beta_{\Box K}$$

We can get a clue for an η-law via *local expansion*, which is witnessed by using an elimination form followed immediately by an introduction form from the bottom-up.

$$\begin{array}{c}\mathcal{D}\\ \Gamma; \Delta \vdash E : \Box_K A\end{array}$$

$$\Rightarrow_{\text{E}}$$

$$\frac{\begin{array}{c}\mathcal{D}\\ \Gamma; \Delta \vdash E : \Box_K A\end{array} \qquad \dfrac{\dfrac{}{\Gamma|_K, u ::_K A; \cdot \vdash u : A}\ \text{hyp}^*}{\Gamma, u ::_K A; \Delta \vdash \mathbf{box}_K u : \Box_K A}\ \Box_K\text{I}}{\Gamma; \Delta \vdash \mathbf{letbox}_K\ u = E\ \mathbf{in}\ \mathbf{box}_K u : \Box_K A}\ \Box_K\text{E}$$

Therefore, we have the following law:

$$\frac{}{E \equiv \mathbf{letbox}_K\ u = E\ \mathbf{in}\ \mathbf{box}_K u}\ \eta_{\Box K}$$

While the $\beta\eta$-principles point the way to an operational semantics for our language, they do not yield all the answers. For example, what of the congruence rules? Should a boxed expression be allowed to run under a box? The answers to such questions are out of scope for this paper. They are design decisions that can have ramifications over the practical use of a language. For example, Davies and Pfenning's decision to make boxed expressions in the modal S4 *not* run is part of the reason why their language is suitable for staged computation [3].

3 Epistemic Logic for Polyglots

The language given in Section 2 can be seen as a generalization on the system of Davies and Pfenning. If we limited ourself to a single agent, then we'd have a system that is exactly the same. The question, then, is what does having more than one agent afford us? We propose that having more than one agent grants the ability to express a type system for multiple interoperating languages. Intuitively, each agent is a stand-in for a language. A boxed expression is an expression in a foreign language, typed using that foreign languages' rules.

To develop this interpretation of epistemic logic, we must introduce some new notions. To aid in differentiating between the constructs of a foreign language and our language, we will color-code, use the metavariable F to range over foreign expressions, and use the metavariable τ for foreign types.

3.1 Typing Rules

We begin with mapping types of a foreign language to types in our language. We write $A \sim_K \tau$ to mean "foreign type τ from language K maps to local type A". This idea of type conversion is based on the one given by Patterson et al. [10]. We think of this as a programmer-specified judgment that will be used by a type checker to ensure that the rest of the

program is well-typed even under the presence of foreign code.

Next, we redefine our epistemic context to map directly to foreign types, and define foreign contexts.

$$\begin{array}{rrcl} \textit{Knowledge Context} & \Gamma & ::= & \cdot \mid \Gamma, u ::_K \tau \\ \textit{Foreign Context} & \Phi & ::= & \cdot \mid \Phi, x : \tau \end{array}$$

Now, we consider the mapping of the knowledge context in our language to foreign contexts. We denote a context mapping with the relation $\Gamma \sim_K \Phi$.

Definition 3.1. Let Γ be a knowledge context and Φ be a foreign context from language K. We define a translation from the former to the latter inductively as follows:

$$\frac{}{\cdot \sim_K \cdot}$$

$$\frac{\Gamma \sim_K \Phi}{\Gamma, x ::_K \tau \sim_K \Phi, x : \tau} \qquad \frac{\Gamma \sim_K \Phi \qquad K \neq L}{\Gamma, x ::_L \tau \sim_K \Phi}$$

Finally, we introduce a new judgment

$$\Phi \vdash_K F : \tau,$$

denoting that a foreign function F is typed at τ under the foreign context Φ.

We may now discuss our new typing rules.

$$\frac{\Gamma \sim_K \Phi \qquad \Phi \vdash_K F : \tau}{\Gamma; \Delta \vdash \mathbf{box}_K F : \Box_K \tau}\ \Box_K \mathrm{I}$$

This new rule requires that we show that the boxed expression type checks at τ under the foreign context constructed from translating Γ. Note that, by Definition 3.1, the types transferred to Φ are exactly those from $\Gamma|_K$.

Next is our new elimination form.

$$\frac{\Gamma; \Delta \vdash E_1 : \Box_K \tau \qquad \Gamma, u ::_K \tau; \Delta \vdash E_2 : C}{\Gamma; \Delta \vdash \mathbf{letbox}_K\ u = E_1\ \mathbf{in}\ E_2 : C}\ \Box_K \mathrm{E}$$

The new rule allows us to switch from a foreign typing context back to our host typing context. We may permanently use the fact that the boxed code is typed at τ.

Last but not least are our variable rules.

$$\frac{}{\Gamma; \Delta, x : A \vdash x : A}\ \mathrm{hyp} \qquad \frac{A \sim_K \tau}{\Gamma, u ::_K \tau; \Delta \vdash u : A}\ \mathrm{hyp}^*$$

The first rule (hyp) remains the same. The second rule allows us to use the fact that an agent knows τ to prove A given that A "acts like" τ.

We must now take stock of our development thus far. Up to this point, we have reintroduced our introduction and elimination forms in the context of multilanguage computation. If our goal was to develop a glue language that acts as a shared compilation target a la Patterson et al. [10], then our rules may be enough. However, our motivation is to have the glue language act as an intermediary between multiple languages. Our elimination form ($\Box_K$E) shows us that we may use data from a foreign language in our glue language. Thus, we have a way to take foreign code as input. What about output? Unfortunately, the rules of epistemic S4 do not give us a way to transfer data from the glue language to a foreign language. Therefore, we must invent a new expression external to the logic to allow us to do this.

We extend $\lambda^{\Box K}$ with a new expression as follows:

$$\textit{Expression} \quad E \quad ::= \cdots \mid \mathbf{tell}_K\ u = E_1\ \mathbf{in}\ E_2,$$

where **tell** is named to suggest the act of telling a principal some fact. We define a single typing rule for this new expression.

$$\frac{\Gamma; \Delta \vdash E_1 : A \qquad A \sim_K \tau \qquad \Gamma, u ::_K \tau; \Delta \vdash E_2 : C}{\Gamma; \Delta \vdash \mathbf{tell}_K\ u = E_1\ \mathbf{in}\ E_2 : C}\ \mathrm{tell}$$

This rule allows us to push a true proposition into the knowledge base of an agent.

Let's take a look at an example. We have two typical simply-typed lambda calculi a and b extended with particular base types. Language a will have a String base type, and language b will support natural numbers ($\mathbb{N}$). Suppose fib is a library function in language b that computes a Fibonacci number efficiently. Now, we extend $\lambda^{\Box K}$ with natural numbers as base types, such that $\mathbb{N} \sim_a \mathrm{String}$ and $\mathbb{N} \sim_b \mathbb{N}$. Then, we may write the following example program:

$$\mathbf{letbox}_a\ u = \mathbf{box}_a \text{"nine"}\ \mathbf{in}\ (\mathbf{tell}_b\ w = u\ \mathbf{in}\ \mathbf{box}_b \mathrm{fib}\ w).$$

A type derivation for this program can be seen in Figure 2.

3.2 $\beta\eta$-Principles

Just like with the monolingual version of $\lambda^{\Box K}$, we witness local reduction and expansion to give us a clue for the operational semantics of our language. Before we begin, we must develop the concept of substitution in the presence of foreign code. We write $\ulcorner F \urcorner^K = E$ to mean "foreign expression F translates to glue expression E". We picture the function $\ulcorner \_ \urcorner$ to be a parameter for the language that dictates how the glue code processes its inputs with the stipulation that the following principle holds.

Principle 1. *If* $\Phi \vdash_K F : \tau$, $\Gamma \sim_K \Phi$, *and* $A \sim_K \tau$, *then* $\Gamma; \cdot \vdash \ulcorner F \urcorner^K : A$.

Next, we need have a substitution principle.

Proposition 2 (Substitution Principle for Polyglot Knowledge)**.** *If* $\Phi \vdash_K F : \tau$ *and* $\Gamma, u ::_K \tau; \Delta \vdash E : C$, *then* $\Gamma; \Delta \vdash [\ulcorner F \urcorner^K / u]E : C$.

This substitution principle captures one aspect of the glue language's role as a mediator: it should translate well-typed inputs into well-typed intermediate representations. With this principle in hand, we may witness our local reduction.

$$\dfrac{\dfrac{\vdots}{\cdot \vdash_a \text{“nine”} : \mathrm{String}}}{\cdot;\cdot \vdash \mathbf{box}_a \text{“nine”} : \Box_a \mathrm{String}}\,\Box_a\mathrm{I} \qquad \dfrac{\dfrac{\dfrac{\vdots}{w : \mathbb{N} \vdash \mathrm{fib} : \mathbb{N} \to \mathbb{N}} \quad \dfrac{}{w : \mathbb{N} \vdash w : \mathbb{N}}\,\mathrm{hyp}}{w : \mathbb{N} \vdash \mathrm{fib}\ w : \mathbb{N}}\,\supset \mathrm{E}}{u ::_a \mathbb{N}, w ::_b \mathbb{N};\cdot \vdash \mathbf{box}_b \mathrm{fib}\ w : \Box_b \mathbb{N}}\,\Box_a\mathrm{I}}{u ::_a \mathbb{N};\cdot \vdash \mathbf{tell}_b\ w = u\ \mathbf{in}\ \mathbf{box}_b \mathrm{fib}\ w : \Box_b \mathbb{N}}\,\mathrm{tell} \qquad \dfrac{}{u ::_a \mathbb{N};\cdot \vdash u : \mathbb{N}}\,\mathrm{hyp}^*$$

$$\cdot;\cdot \vdash \mathbf{letbox}_a\ u = \mathbf{box}_a \text{“nine”}\ \mathbf{in}\ (\mathbf{tell}_b\ w = u\ \mathbf{in}\ \mathbf{box}_b \mathrm{fib}\ w) : \Box_b \mathbb{N}\ \Box_a\mathrm{E}$$

Figure 2. Fragment of the typing derivation for our running example, eliding side conditions for context and type translations.

$$\begin{array}{ll} & \mathbf{letbox}_a\ u = \mathbf{box}_a \text{“nine”}\ \mathbf{in}\ (\mathbf{tell}_b\ w = u\ \mathbf{in}\ \mathbf{box}_b \mathrm{fib}\ w) \\ \mapsto (\beta_{\Box K}) & \mathbf{tell}_b\ w = 9\ \mathbf{in}\ \mathbf{box}_b \mathrm{fib}\ w \qquad \text{with } \ulcorner \text{“nine”} \urcorner^a = 9 \\ \mapsto (\beta_{tell_K}) & \mathbf{box}_b \mathrm{fib}\ 9 \qquad \text{with } \llcorner 9 \lrcorner^b = 9 \end{array}$$

Figure 3. Speculative run of our example program.

$$\dfrac{\dfrac{\mathcal{D}}{\Phi \vdash_K F : \tau}}{\Gamma;\Delta \vdash \mathbf{box}_K F : \Box_K \tau}\,\Box_K\mathrm{I} \qquad \dfrac{\mathcal{E}}{\Gamma, u ::_K \tau;\Delta \vdash E : C}\,\Box_K\mathrm{E}$$

$$\Rightarrow_{\mathrm{R}}$$

$$\mathcal{F}$$

$$\Gamma;\Delta \vdash [\ulcorner F \urcorner^K / u]E : C,$$

where $\mathcal{F}$ is given by the multilingual substitution principle. With this derivation in mind, we have a candidate β-reduction rule:

$$\dfrac{}{\Gamma;\Delta \vdash \mathbf{letbox}_K\ u = \mathbf{box}_K F\ \mathbf{in}\ E : C \mapsto [\ulcorner F \urcorner^K / u]E}\,\beta_{\Box K}$$

Local expansion is much the same as it was before. We assume that the foreign language has a hypothesis rule akin to the one we described in Section 2.2.

$$\dfrac{\mathcal{D}}{\Gamma;\Delta \vdash E : \Box_K \tau}$$

$$\Rightarrow_{\mathrm{E}}$$

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma;\Delta \vdash E : \Box_K \tau} \qquad \dfrac{\dfrac{}{\Phi, u : \tau \vdash_K u : \tau}\,\mathrm{hyp}}{\Gamma, u ::_K \tau;\Delta \vdash \mathbf{box}_K u : \Box_K \tau}\,\Box_K\mathrm{I}}{\Gamma;\Delta \vdash \mathbf{letbox}_K\ u = E\ \mathbf{in}\ \mathbf{box}_K u : \Box_K \tau}\,\Box_K\mathrm{E}$$

Thus, we have the following η-law:

$$\dfrac{}{E \equiv \mathbf{letbox}_K\ u = E\ \mathbf{in}\ \mathbf{box}_K u}\,\eta_{\Box K}$$

As we stated in Section 2.3, the $\beta\eta$-principles help us understand some of the behavior of our expressions, while leaving us with a space of design decisions. We must ask some of the same questions we asked then. Answering these is the subject of ongoing work.

One such question is on the behavior of the **tell** expression. Without a logical account for it, we'll need to design one from scratch. Toward a speculative reduction rule, we write $\llcorner E \lrcorner^K = F$ to mean "glue expression E translates to foreign expression F". This is another function to be a parameter for the language specified by a user with the requirement that the following principle holds:

Principle 2.

1. $\llcorner \ulcorner F \urcorner^K \lrcorner^K = F$
2. $\ulcorner \llcorner E \lrcorner^K \urcorner^K = E$

With this principle, we give the following candidate reduction rule:

$$\dfrac{}{\Gamma;\Delta \vdash \mathbf{tell}_K\ u = E_1\ \mathbf{in}\ E_2 \mapsto [\llcorner E_1 \lrcorner^K / u]E_2}\,\beta_{tell_K}$$

For an example of these rules in play, see Figure 3.

4 Related Work

Multilanguage computation is a field with a rich history and a wide variety of goals and techniques. We will discuss the most relevant pieces of work, which concern themselves with more syntactic formalisms for multilanguage computation.

Of course, we must mention the seminal work of Matthews and Findler [8], which establishes different ways to embed foreign code into a program in a type-directed way. We view our box and letbox expressions as two halves of the boundary expression in that work. Rather than focus on two languages interoperating directly with each other via these boundaries, our presentation centers on languages operating through glue code, much like the work of Patterson et al. [10], which we will discuss below. Note that we do not provide extra semantics for when specific expression forms are boxed. For example, we do not provide rules for transforming boxed lambda expressions. This is because our languages are not directly embedded into each other: when a lambda expression is inserted into code, it is an expression in that code's language.

The interpretation of epistemic logic as glue code for implementing type and expression conversions is heavily influenced by the work of Patterson et al. [10]. However, while they study languages compiling into a shared target, we are exploring the use of glue code as an intermediate representation that links disparate languages.

Readers may find our work to be very similar to that of Zdancewic et al. [13] due to the emphasis of thinking of principals as constructs in a programming language. In this work, principals are stand-ins for different pieces of a system, such as a client and a host in a file system. The principals of this work are not representations of different languages. Thus, they do not study language interoperation.

5 Future Work, Discussion, and Conclusion

There is, of course, some work to do on this concept. First, we need to explore some metatheory, and to understand how differing properties of the foreign languages might interact the properties of the glue language. Second, we need to think on our operational semantics. Third, we are interested in finding a logical account for the **tell** expression.

We speculate that such an account may be found in one of many different formulations of epistemic logic. The ability to transfer information held by one agent to another is realized in *dynamic epistemic logic* (DEL), an extension of epistemic logic that includes actions which update agent knowledge [1]. In this setting, agents are able to act in such ways that reveal information to other agents, such as simply telling each other facts. Aside from DEL, there is also the concept of *public knowledge* [9]. Glue code acting as an intermediary suggests a logical view of it as a source that different agents can gain knowledge from without directly interacting with each other. Lastly, we speculate that a treatment of the epistemic language with the lens of *adjoint logic*, a framework for combining logics, can be illumnating [7]. From this perspective, epistemic logic is a logic combining two *modes* of truth: truth itself and knowledge. Shift operations allow us to embed propositions from one mode into another mode. We speculate that **tell** may be expressible as a particular combination of these shifts.

Further on the horizon for this work is an exploration of its practical use. In this regard, we are inspired by work in *choreographic programming* [2, 6, 12]. Here, a choreography is a program that specifies, from a global point of view, the behavior of a multiagent system. Static type-checking gives guarantees about the system, such as freedom from deadlocks. By way of *endpoint projection*, local programs corresponding to locations in the system are derived. Now, there is some indication that the modal logic for choreographic programming is not epistemic, but *doxastic*—that is, relating to belief, the dual of knowledge [5]. Whether or not this is true is yet to be seen. In the meantime, we speculate that, from a language based on $\lambda^{\Box K}$, we may give rise to practical systems via an inverse to endpoint projection: rather than derive local programs that run independently, we generate one that acts between the local programs.

We have presented $\lambda^{\Box K}$, a computational interpretation of the epistemic S4, and an extension with **tell**. The monolingual epistemic S4 described in this paper is an extension of the modal S4 introduced by Pfenning and Daves [11], and a natural deduction expression of the sequent calculus given by Garg et al. [4]. The multilingual version of the logic interprets principals as programming languages, and a computational interpretation of it yields a language for middleware between multiple disjoint pieces of code. As we formulated this version of $\lambda^{\Box K}$, we observed that the flow of information is one-way: data from foreign code can be imported by the glue code, but not vice-versa. This led us to introduce the **tell** expression. Our $\beta\eta$-principles show us how these expressions behave, and give us some clues for operational semantics. However, the design of a full suite of semantics is the subject of future work.

6 Acknowledgments

The authors thank the anonymous reviewers for their thorough and helpful feedback, especially Reviewer 1 for the insight to track foreign types, rather than glue types, in the knowledge context. We also thank Andrew Wagner and Cameron Moy for helpful discussions about multilanguages and metaprogramming, which informed this work.

References

[1] Baltag, A., and Renne, B. Dynamic epistemic logic.

[2] Cruz-Filipe, L., Graversen, E., Lugović, L., Montesi, F., and Peressotti, M. Functional choreographic programming. In *International Colloquium on Theoretical Aspects of Computing* (2022), Springer, pp. 212–237.

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

[4] Garg, D., Bauer, L., Bowers, K. D., Pfenning, F., and Reiter, M. K. A linear logic of authorization and knowledge. In *European Symposium on Research in Computer Security* (2006), Springer, pp. 297–312.

[5] Hirsch, A. K. Corps: A core calculus of hierarchical choreographic programming. *arXiv preprint arXiv:2406.01456* (2024).

[6] Hirsch, A. K., and Garg, D. Pirouette: higher-order typed functional choreographies. *Proceedings of the ACM on Programming Languages 6*, POPL (2022), 1–27.

[7] Jang, J., Roshal, S., Pfenning, F., and Pientka, B. Adjoint natural deduction. In *9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024)* (2024), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 15–1.

[8] Matthews, J., and Findler, R. B. Operational semantics for multi-language programs. *ACM Transactions on Programming Languages and Systems (TOPLAS) 31*, 3 (2009), 1–44.

[9] Meyer, J.-J. C., Meyer, J.-J. C., and van der Hoek, W. *Epistemic logic for AI and computer science*. No. 41. Cambridge University Press, 2004.

[10] Patterson, D., Mushtak, N., Wagner, A., and Ahmed, A. Semantic soundness for language interoperability. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (2022), pp. 609–624.

[11] Pfenning, F., and Davies, R. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science 11*, 04 (Aug. 2001).

[12] Shen, G., Kashiwa, S., and Kuper, L. Haschor: Functional choreographic programming for all (functional pearl). *Proceedings of the ACM on Programming Languages 7*, ICFP (2023), 541–565.

[13] Zdancewic, S., Grossman, D., and Morrisett, G. Principals in programming languages: A syntactic proof technique. *ACM SIGPLAN Notices 34*, 9 (1999), 197–207.