Modal Proofs As Distributed Programs*

Limin Jia David Walker
Princeton University

July, 2003

Abstract

We develop a new foundation for distributed programming languages by defining an intuitionistic, modal logic and then interpreting the modal proofs as distributed programs. More specifically, the proof terms for the various modalities have computational interpretations as *remote procedure calls*, commands to *broadcast* computations to all nodes in the network, commands to use *portable* code, and finally, commands to invoke computational *agents* that can find their own way to safe places in the network where they can execute. We prove some simple meta-theoretic results about our logic as well as a safety theorem that demonstrates that the deductive rules act as a sound type system for a distributed programming language.

1 Introduction

One of the characteristics of distributed systems that makes developing robust software for them far more difficult than developing software for single stand-alone machines is *heterogeneity*. Different places in the system may have vastly different properties and resources. For instance, different machines may be attached to different hardware devices, have different software installed and run different services. Moreover, differing security concerns may see different hosts providing different interfaces to distributed programs, even when the underlying computational resources are similar.

In order to model such heterogeneous environments, programming language researchers usually turn to formalisms based on one sort of process algebra or another. Prime examples include the distributed join calculus [5] and the ambient calculus [3]. These calculi often come with rich theories of process equivalence and are useful tools for reasoning about distributed systems. However, a significant disadvantage of starting with process algebras as a foundation for distributed computing is that they immediately discard the wealth of logical

*This research was supported in part by NSF Career Award CCR-0238328 and DARPA award F30602-99-1-0519.

principles that underlie conventional sequential programming paradigms and that form the sequential core of any distributed computation.

In this paper, we develop a foundation for safe distributed programming, which rather than rejecting the logical foundations of sequential (functional) programming, extends them conservatively with new principles tuned to programming in heterogeneous distributed environments. More specifically, we develop an intuitionistic, modal logic and provide an operational interpretation of the logical proofs as distributed programs. Our logic is conservative over ordinary intuitionistic propositional logic in the sense that at any given place, all of the propositional tautologies are provable. Consequently, our correspondence between proofs and programs implies we can safely execute any (closed) functional program at any place in our distributed programming environment.

We extend these simple intuitionistic proofs with modal connectives and provide computational interpretations of the connectives as operations for remote code execution:

- Objects with type $\tau \mathbin{@} z$ are return values with type τ. They are the results of *remote procedure calls* from the place z.
- Objects with type $n[\tau]$ are also return values with type τ. However, they are the results of remote procedure calls that use *place-relative* rather than *absolute* addressing. These RPCs send a computation along the link in the network named n from the current place.
- Objects with type $\Box\,\tau$ are computations that run safely everywhere, and may be *broadcast* to all places in the network.
- Objects with type $\boxminus\tau$ are logically equivalent to those objects with type $\Box\,\tau$, but are treated operationally simply as *portable* code that can run everywhere, but is not actually broadcast.
- Objects with type $\Diamond\tau$ are computational *agents* that have internalized the place z where they may execute safely to produce a value with type τ.

Contributions The central technical contributions of our work may be summarized as follows.

- We develop an intuitionistic, modal logic from first principles following the logical design techniques espoused by Martin Löf [8] and Frank Pfenning [11, 12]. Our logic is a relative of the hybrid logics, which are discussed in more detail at the end of the paper (see Section 4).
- The logic obeys a number of simple properties that give evidence it will serve as a solid foundation for distributed programming languages. In particular, each connective is defined orthogonally to all others; is shown to be locally sound and complete; and supports the relevant substitution principles.

- This paper concentrates on the natural deduction formulation of the logic due to its correspondence with functional programs. However, we have also developed a sequent calculus that has cut elimination and shown that the sequent calculus can prove the same theorems as the natural deduction system.
- We give an operational interpretation of the proofs in our logic as distributed functional programs. We prove that the logical deduction rules are sound when viewed as a type system for the programming language.

Due to space considerations, we have omitted the detailed proofs of our theorems. Further details may be found in an extended technical report [7].

2 A Logic of Places

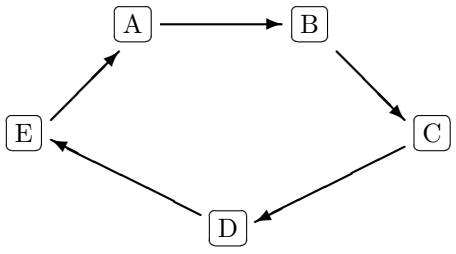
2.1 Preliminaries

The central purpose of our logic is to facilitate reasoning about heterogeneous distributed systems where different nodes may have different properties and may contain different resources. Hence, the primary judgment in the logic not only considers *whether* a formula is true, but also *where* it is true. More precisely, each primitive judgment has the form

$$\vdash^{P,N} F \ \mathsf{at}\ z$$

where F is a formula from the logic and z is a particular place in the system. These places may either be simple places, p, which are drawn from the set P, or compound places $p.n_1.\ldots.n_n$ that specify the relative address of a place in terms of a simple place p and a path with edges labeled n_1, ..., n_n. Each edge name is drawn from the set N. We consider judgments $\vdash^{P,N} F \ \mathsf{at}\ z$ to have no meaning if $\mathsf{FPN}(F) \cup \mathsf{FPN}(z) \not\subseteq P \cup N$ where $\mathsf{FPN}(X)$ denotes the set of free places and names that appears in X. We also use $\mathsf{FP}(X)$ ($\mathsf{FN}(X)$) to denote the set of free places (names) in X. In the inference rules to come, we do not generally specify these well-formedness conditions explicitly. When P and N are unimportant or easy to guess from the context (i.e., most of the time) we omit them from the judgment and simply write $\vdash F \ \mathsf{at}\ z$.

To gain some intuition about this preliminary set up, consider a network of computers connected in a ring:



This picture shows five simple places (A, B, etc.), which may be referenced directly. Alternatively, if we assume that each arc in the diagram is labeled with the name next, we may refer to B indirectly using the path A.next or E.next.next.

In another scenario, each of A, B, C might support several virtual machines named VM_1, VM_2, etc. If the set N includes names VM_1 and VM_2 then we can use indirect addressing to build a hierarchy of places by refering to A.VM_1, A.VM_2, B.VM_1, etc.

In general, each of these computers (A, B, C, etc.) may be attached to different physical devices. For instance, E may be attached to a printer and B may be attached to a scanner. If sc ("there is a scanner here") and pt ("there is a printer here") are propositions in the logic, we might assert judgments such as $\vdash\ \mathsf{pt}\ \mathsf{at}\ E$ and $\vdash\ \mathsf{sc}\ \mathsf{at}\ B$ to describe this situation.

Hypothetical Judgments In order to engage in more interesting reasoning about distributed resources, we must define hypothetical judgments, facilities for reasoning from hypotheses and the appropriate notion of substitution. To begin with, hypothetical judgments have the form $\Delta \vdash^{P,N} F\ \mathsf{at}\ z$ where Δ is a list of (variable-labeled) assumptions:

$$\textit{contexts}\quad \Delta \quad ::= \quad \cdot \mid \Delta, x : F\ \mathsf{at}\ z$$

We make the usual assumption that no variables are repeated in the context, and whenever adding a new variable to the context, we implicitly assume it is different from all others (alpha-varying bound variables where necessary to ensure this invariant holds). In addition, we do not distinguish between contexts that differ only in the order of assumptions.

Logicians may use hypotheses according to the following inference rule.

$$\frac{}{\Delta, x : F\ \mathsf{at}\ z \vdash\ F\ \mathsf{at}\ z}\ \mathit{hyp}$$

Intuitionistic Connectives With the definition of hypothetical judgments in hand, we may proceed to give the meaning of the usual intuitionistic connectives for truth ($\top$), implication ($F_1 \to F_2$) and conjunction ($F_1 \wedge F_2$) in terms of their introduction and elimination rules.

$$\frac{}{\Delta \vdash\ \top\ \mathsf{at}\ z}\ \top I$$

$$\frac{\Delta, F_1\ \mathsf{at}\ z \vdash\ F_2\ \mathsf{at}\ z}{\Delta \vdash\ F_1 \to F_2\ \mathsf{at}\ z} \to I$$

$$\frac{\Delta \vdash\ F_1 \to F_2\ \mathsf{at}\ z \quad \Delta \vdash\ F_1\ \mathsf{at}\ z}{\Delta \vdash\ F_2\ \mathsf{at}\ z} \to E$$

$$\frac{\Delta \vdash\ F_1\ \mathsf{at}\ z \quad \Delta \vdash\ F_2\ \mathsf{at}\ z}{\Delta \vdash\ F_1 \wedge F_2\ \mathsf{at}\ z} \wedge I$$

$$\frac{\Delta \vdash\ F_1 \wedge F_2\ \mathsf{at}\ z}{\Delta \vdash\ F_1\ \mathsf{at}\ z} \wedge E1 \qquad \frac{\Delta \vdash\ F_1 \wedge F_2\ \mathsf{at}\ z}{\Delta \vdash\ F_2\ \mathsf{at}\ z} \wedge E2$$

None of the rules above are at all surprising: Each rule helps explain how one of the usual intuitionistic connectives operates at a particular place. Hence, if we limit the set of places to a single place "_" the logic will reduce to ordinary intuitionistic logic. So far, there is no interesting way to use assumptions at multiple different places, but we will see how to do that in a moment.

As a simple example, consider reasoning about the action of the printer at place E in the network we introduced earlier. Let pdf be a proposition indicating the presence of a PDF file waiting to be printed and po be a proposition indicating the presence of a printout. The following derivation demonstrates how we might deduce the presence of a printout at E. The context Δ referenced below is

$$\mathsf{f}_E : \mathsf{pdf}\ \mathsf{at}\ E, \mathsf{ptr}_E : \mathsf{pt}\ \mathsf{at}\ E, \mathsf{print} : \mathsf{pdf} \wedge \mathsf{pt} \rightarrow \mathsf{po}\ \mathsf{at}\ E$$

This context represents the presence of a PDF file and printer at E as well as some software (a function) installed at E that will initiate the printing process.

$$\dfrac{\dfrac{}{\Delta \vdash \mathsf{pdf} \wedge \mathsf{pt} \rightarrow \mathsf{po}\ \mathsf{at}\ E} \quad \dfrac{\dfrac{}{\Delta \vdash \mathsf{pdf}\ \mathsf{at}\ E} \quad \dfrac{}{\Delta \vdash \mathsf{pt}\ \mathsf{at}\ E}}{\Delta \vdash \mathsf{pdf} \wedge \mathsf{pt}\ \mathsf{at}\ E}}{\Delta \vdash \mathsf{po}\ \mathsf{at}\ E}$$

Simple Logical Properties Before continuing further, we should do some preliminary checks on the reasonableness of our definitions. More specifically, following Martin Löf's [8] and Frank Pfenning's [11] design principles, we should check that the elimination rules are *locally sound* and *locally complete* with respect to the corresponding introduction rules for the logic.

Local soundness guarantees that whenever an elimination rule directly follows an introduction rule in a proof, the two adjacent rules may be eliminated and the conclusion proven in a simpler way. From a proofs-as-programs perspective, local soundness corresponds to type preservation under a single beta reduction. Local soundness implies the elimination rules are not too strong for the introduction rules; the elimination rules do not introduce new information into a proof that was not already there. As an example, we exhibit one case of the local soundness proof for conjunction through the following proof reduction ($\Rightarrow_r$). The second case of the proof is similar to the first, but involves $\wedge E_2$. A calligraphic letter ($\mathcal{D}_1$, etc.) in a hypothesis position ranges over arbitrary derivations of the corresponding conclusion.

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\Delta \vdash F_1\ \mathsf{at}\ z} \quad \dfrac{\mathcal{D}_2}{\Delta \vdash F_2\ \mathsf{at}\ z}}{\Delta \vdash F_1 \wedge F_2\ \mathsf{at}\ z}\ \wedge I}{\Delta \vdash F_1\ \mathsf{at}\ z}\ \wedge E1 \quad \Rightarrow_r \quad \dfrac{\mathcal{D}_1}{\Delta \vdash F_1\ \mathsf{at}\ z}$$

Local completeness guarantees that the elimination rules are not too weak for the introduction rules. This property implies that the elimination rules can extract sufficient information from an arbitrary proof of the connective

to reconstruct that connective using the introduction rules. From a proofs-as-programs perspective, local completeness corresponds to type preservation under a single eta-expansion. Here, we present the local expansion ($\Rightarrow_e$) that witnesses the local completeness property for conjunction.

$$\frac{\mathcal{E}}{\Delta \vdash F_1 \wedge F_2 \text{ at } z} \quad \Rightarrow_e \quad \frac{\dfrac{\dfrac{\mathcal{E}}{\Delta \vdash F_1 \wedge F_2 \text{ at } z}}{\Delta \vdash F_1 \text{ at } z} \wedge E1 \quad \dfrac{\dfrac{\mathcal{E}}{\Delta \vdash F_1 \wedge F_2 \text{ at } z}}{\Delta \vdash F_2 \text{ at } z} \wedge E2}{\Delta \vdash F_1 \wedge F_2 \text{ at } z} \wedge I$$

Lemma 1
The introduction and elimination rules for truth, conjunction and implication are locally sound and complete.

Local soundness and completeness help check the consistency of the logic at an early stage of development and can be used to detect fundamental flaws in logical design. For instance, the elimination rules for conjunction are not locally complete with respect to the following more general introduction rule.

$$\frac{\Delta \vdash F_1 \text{ at } z_2 \quad \Delta \vdash F_2 \text{ at } z_3}{\Delta \vdash F_1 \wedge F_2 \text{ at } z_1} \wedge I \; Bad$$

Similarly, generalizing conjunction elimination as follows, without change to the introduction rules, will lead to a lack of local soundness.

$$\frac{\Delta \vdash F_1 \wedge F_2 \text{ at } z_2}{\Delta \vdash F_1 \text{ at } z_1} \wedge E1 \; Bad$$

The lack of local soundness or completeness in these "bad" rules is symptomatic of the fact that they deviate from the central principle guiding the logic (and therefore language) design: In each case, they allow formulas firmly rooted at a particular place to implicitly shift to a new place, making the presence of places in the judgment meaningless. For instance, use of the bad introduction rule allows a formula F_1 at z_1 to shift to the place z_2).

$$\frac{\dfrac{\dfrac{\mathcal{D}_1}{\Delta \vdash F_1 \text{ at } z_1} \quad \dfrac{\mathcal{D}_2}{\Delta \vdash F_2 \text{ at } z_2}}{\Delta \vdash F_1 \wedge F_2 \text{ at } z_2} \wedge I \; Bad}{\Delta \vdash F_1 \text{ at } z_2} \wedge E1$$

This implicit shift renders attempts to reason about immobile objects (such as our printer) impossible, and hence is at odds with the design we hope to achieve. In the following sections, however, we add connectives that allow us to *explicitly* shift perspective. The explicit shift gives the logician the flexibility to reason about objects at remote locations, but is completely compatible with our work so far.

2.2 Interplace Reasoning

To reason about relationships between objects located at different places we introduce two modal connectives, one which describes objects in terms of their *absolute* location in the system and one that describes objects in terms of their position *relative* to the current location.

Absolute Placement We derive our first modal connective by internalizing the judgmental notion that a formula is true at a particular place, but not necessarily elsewhere. We write this new modal formula as $F \mathbin{@} z$. The introduction and elimination rules follow.

$$\frac{\Delta \vdash F \ \mathsf{at}\ z_2}{\Delta \vdash F \mathbin{@} z_2 \ \mathsf{at}\ z_1}\ @\,I \qquad \frac{\Delta \vdash F \mathbin{@} z_2 \ \mathsf{at}\ z_1}{\Delta \vdash F \ \mathsf{at}\ z_2}\ @\,E$$

This connective allows us to reason about objects, software or devices "from a distance." For instance, in our printer example, it is possible to refer to the printer located at E while reasoning at D; to do so we might assert $\Delta \vdash \mathsf{pt} \mathbin{@} E \ \mathsf{at}\ D$. Moreover, we can relate objects at one place with objects at another. For instance, in order to share E's printer, D needs to have software that can convert local PDF files at D to files that may be used and print properly at E (perhaps this sofware internalizes some local fonts, inaccessible to E, within the document). An assumption of the form $\mathsf{DtoE} : \mathsf{pdf} \to \mathsf{pdf} \mathbin{@} E \ \mathsf{at}\ D$ would allow us to reason about such software.[1] If Δ' is the assumption DtoE above together with an assumption $\mathsf{f_D} : \mathsf{pdf}\ \mathsf{at}\ D$, the following derivation allows us to conclude that we can get the PDF file to E. We can easily compose this proof with the earlier one to demonstrate that PDF files at D can not only be sent to E, but actually printed there.

$$\dfrac{\dfrac{\dfrac{}{\Delta' \vdash \mathsf{pdf} \to \mathsf{pdf} \mathbin{@} E \ \mathsf{at}\ D}\ hyp \quad \dfrac{}{\Delta' \vdash \mathsf{pdf}\ \mathsf{at}\ D}\ hyp}{\Delta' \vdash \mathsf{pdf} \mathbin{@} E \ \mathsf{at}\ D}\ \to E}{\Delta' \vdash \mathsf{pdf}\ \mathsf{at}\ E}\ @\,E$$

Relative Placement The connective above allows positioning of a formula with respect to an absolute (fully-determined) place z. A closely related connective, $n[F]$, specifies that an object or resource described by F may be found by traveling along the edge labeled n when starting from the current place. The introduction and elimination rules for $[\,]$ have similar structure to the rules for $@$:

$$\frac{\Delta \vdash F \ \mathsf{at}\ z.n}{\Delta \vdash n[F] \ \mathsf{at}\ z}\ [\,]\,I \qquad \frac{\Delta \vdash n[F] \ \mathsf{at}\ z}{\Delta \vdash F \ \mathsf{at}\ z.n}\ [\,]\,E$$

[1] We assume that "@" binds tighter than implication or conjunction. When fully parenthesized, the assumption above has the following form.

$$(\mathsf{pdf} \to (\mathsf{pdf} \mathbin{@} E)) \ \mathsf{at}\ D$$

The owner of machine D often finds interesting technical reports on the Web that are only distributed in PostScript. Since E's printer will only print PDF files, D takes the step of installing Adobe Acrobat and its Distiller program. Distiller automatically takes PostScript documents that it finds in a local folder called "in," converts the documents into PDF and places them in the folder called "out." D's new software set up, including routines to copy PostScript and PDF files to and from the appropriate places can be described by the following context:

$$\begin{array}{l}\mathsf{distill} : \mathsf{in}[\mathsf{ps}] \rightarrow \mathsf{out}[\mathsf{pdf}] \text{ at } D,\\ \mathsf{copyIn} : \mathsf{ps} \rightarrow \mathsf{in}[\mathsf{ps}] \text{ at } D,\\ \mathsf{copyOut} : \mathsf{out}[\mathsf{pdf}] \rightarrow \mathsf{pdf} \text{ at } D\end{array}$$

These axioms can easily be composed to prove that PostScript files at D can be converted into PDF files at D—just what the owner of D was hoping to accomplish.

Lemma 2
The modal connectives $F @ z$ and $n[F]$ are locally sound and complete.

2.3 Global Reasoning

While our focus is on reasoning about networks with heterogeneous resources, we cannot avoid the fact that certain propositions are true *everywhere*. For instance, the basic laws of arithmetic do not change from one machine to the next, and consequently, we should not restrict the application of these laws to any particular place. Just as importantly, we might want to reason about distributed applications deployed over a network of machines, all of which support a common operating system interface. The functionality provided by the operating system is available everywhere, just like the basic laws of arithmetic, and use of the functionality need not be constrained to one particular place or another.

Global Judgments To support global reasoning, we generalize the judgment considered so far to include a second context that contains assumptions that are valid everywhere. Our extended judgments have the form

$$\Gamma; \Delta \vdash^{P,N} F \text{ at } z$$

where Γ is a global context and Δ is the local context we considered previously.

$$\begin{array}{llll} \textit{Global Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x : F \\ \textit{Local Contexts} & \Delta & ::= & \cdot \mid \Delta, x : F \text{ at } z \end{array}$$

Our extended logic contains two sorts of hypothesis rules, one for using each sort of assumption. L is identical to the hypothesis rule used in previous sections (modulo the unused global context Γ). G specifies how to use global hypotheses; they may be placed in any location and used there.

$$\frac{}{\Gamma; \Delta, x{:}F \text{ at } z \vdash F \text{ at } z}\ L \qquad \frac{}{\Gamma, x{:}F; \Delta \vdash F \text{ at } z}\ G$$

All rules from the previous sections are included in the new system unchanged aside from the fact that Γ is passed unused from conclusion to premises.

Internalizing Global Truth The modal connective $\Box\, F$ internalizes the notion that the formula F is true everywhere. If a formula may be proven true at a new place p, which by definition can contain no local assumptions, then that formula must be true everywhere:

$$\frac{\Gamma; \Delta \vdash^{P+p,N} F \text{ at } p \quad p \notin \mathsf{FP}(\Gamma) \cup \mathsf{FP}(\Delta) \cup \mathsf{FP}(F)}{\Gamma; \Delta \vdash^{P,N} \Box F \text{ at } z}\ \Box I$$

Here, we use $P + p$ to denote the disjoint union $P \cup \{p\}$. If $p \in P$, we consider $P + p$, and any judgment containing such notation, to be undefined. The side condition above ensures that p is truly new and that the conclusion of the rule is well formed in the absence of p.[2]

If we can prove $\Box\, F$, we can assume that F is globally true in the proof of any other formula F':

$$\frac{\Gamma; \Delta \vdash^{P,N} \Box F \text{ at } z \quad \Gamma, x : F; \Delta \vdash^{P,N} F' \text{ at } z'}{\Gamma; \Delta \vdash^{P,N} F' \text{ at } z'}\ \Box E$$

Returning to our printing example, suppose node E decides to allow *all* machines to send it PDF files. In order to avoid hiccups in the printing process, E intends to distribute software to all machines that allow them to convert local PDF files to files that will print properly on E's printer. We might represent this situation with a hypothesis $\mathsf{ToE} : \Box\,(\mathsf{pdf} \to \mathsf{pdf} \mathbin{@} E) \text{ at } E$. Now, given a PDF file at any other node q in the network (coded as the assumption $\mathsf{f}_q : \mathsf{pdf} \text{ at } q$), we can demonstrate that it is possible to send a pdf file to E using the following proof where Δ'' contains assumptions ToE and f_q. The global context Γ contains the single assumption $\mathsf{ToE}' : \mathsf{pdf} \to \mathsf{pdf} \mathbin{@} E$.

$$\dfrac{\mathcal{D} \quad \dfrac{\dfrac{\dfrac{}{\Gamma; \Delta'' \vdash \mathsf{pdf} \to \mathsf{pdf} \mathbin{@} E \text{ at } q}\ G \quad \dfrac{}{\Gamma; \Delta'' \vdash \mathsf{pdf} \text{ at } q}\ L}{\Gamma; \Delta'' \vdash \mathsf{pdf} \mathbin{@} E \text{ at } q}\ \to E}{\Gamma; \Delta'' \vdash \mathsf{pdf} \text{ at } E}\ @\, E}{\cdot; \Delta'' \vdash \mathsf{pdf} \text{ at } E}\ \Box E$$

where the derivation $\mathcal{D} =$

$$\frac{}{\cdot; \Delta'' \vdash \Box\,(\mathsf{pdf} \to \mathsf{pdf} \mathbin{@} E) \text{ at } E}\ L$$

[2] If we follow our convention about judgments being meaningless when they contain names or places not contained in the sets P and N that annotate the judgment, this side condition is unnecessary. Nevertheless, we leave it in for emphasis.

As another simple example, suppose the owner of A tells the system administrator how great Distiller is, and she installs it on all machines. We can represent the presence of software everywhere using a global context Γ with the following form.

$$\begin{array}{l}\mathsf{distill} : \mathsf{in}[\mathsf{ps}] \to \mathsf{out}[\mathsf{pdf}],\\ \mathsf{copyin} : \mathsf{ps} \to \mathsf{in}[\mathsf{ps}],\\ \mathsf{copyout} : \mathsf{out}[\mathsf{pdf}] \to \mathsf{pdf}\end{array}$$

When any machine q uses these globally available assumptions, it processes PDF files in its own local in and out directories (i.e., at $q.\mathsf{in}$ and $q.\mathsf{out}$), rather than in some global, communally decided upon places.

The truth is out there The dual notion of a globally true proposition F is a proposition that is true *somewhere*, although we may not necessarily know where. We already have all the judgmental apparatus to handle this new idea; we need only internalize it in a connective ($\Diamond F$). The introduction rule states that if the formula holds at any particular place z in the network, then it holds somewhere. The elimination rule explains how we use a formula F that holds somewhere: We introduce a new place p and assume F holds there.

$$\frac{\Gamma; \Delta \vdash^{P,N} F \ \mathsf{at}\ z}{\Gamma; \Delta \vdash^{P,N} \Diamond F \ \mathsf{at}\ z'}\ \Diamond I$$

$$\frac{\Gamma; \Delta \vdash^{P,N} \Diamond F \ \mathsf{at}\ z \qquad \Gamma; \Delta, x : F \ \mathsf{at}\ p \vdash^{P+p,N} F' \ \mathsf{at}\ z' \quad p \notin \mathsf{FP}(F') \cup \mathsf{FP}(z')}{\Gamma; \Delta \vdash^{P,N} F' \ \mathsf{at}\ z'}\ \Diamond E$$

Lemma 3
The modal connectives $\Box$ and $\Diamond$ are locally sound and complete.

Modal Axioms Possibility and necessity satisfy the following standard modal axioms (taken from Huth and Ryan [6, p. 284]).

$$\begin{array}{ll}\text{K:} & \cdot;\cdot \vdash \Box\,(F_1 \to F_2) \to (\Box\,F_1 \to \Box\,F_2) \ \mathsf{at}\ p\\ \text{B:} & \cdot;\cdot \vdash F \to \Box \Diamond F \ \mathsf{at}\ p\\ \text{D:} & \cdot;\cdot \vdash \Box\,F \to \Diamond F \ \mathsf{at}\ p\\ \text{T:} & \cdot;\cdot \vdash \Box\,F \to F \ \mathsf{at}\ p\\ \text{4:} & \cdot;\cdot \vdash \Box\,F \to \Box\,\Box\,F \ \mathsf{at}\ p\\ \text{5:} & \cdot;\cdot \vdash \Diamond F \to \Box \Diamond F \ \mathsf{at}\ p\end{array}$$

2.4 Properties

To summarize, our logic of places contains the familiar connectives from propositional intuitionistic logic as well as several modalities for reasoning about software and data distributed across a network:

$$F ::= \top \mid F_1 \to F_2 \mid F_1 \wedge F_2 \mid F\,@\,z \mid n[F] \mid \Box\,F \mid \Diamond F$$

A summary of the proof rules may be found in the Appendix.

Our proof system satisfies the following substitution properties.

Lemma 4 (Substitution)

1. *If* $\Gamma; \Delta \vdash^{P,N} F \text{ at } z$ *then for all* z', $\mathsf{FPN}(z') \subseteq (P \cup N)$ *implies* $\Gamma[z'/p]; \Delta[z'/p] \vdash^{P,N} F[z'/p] \text{ at } z[z'/p]$
2. *If* $\Gamma; \Delta \vdash^{P,N} F \text{ at } z$ *and* $\Gamma; \Delta, x : F \text{ at } z \vdash^{P,N} F' \text{ at } z'$ *then* $\Gamma; \Delta \vdash^{P,N} F' \text{ at } z'$
3. *If* $\Gamma; \Delta \vdash^{P+q,N} F \text{ at } q$ *and* $\Gamma, x : F; \Delta \vdash^{P,N} F' \text{ at } z'$ *then* $\Gamma; \Delta \vdash^{P,N} F' \text{ at } z'$

Sequent Calculus and Cut Elimination The local soundness and completeness properties we have presented earlier are a good aid when debugging initial definitions of connectives. However, they are no substitute for global properties of the logic. To ensure global consistency of our logic, we have defined a sequent calculus, proven cut elimination and shown that the sequent calculus is sound and complete with respect to our natural deduction formulation.

A sequent calculus judgment has the following form.

$$\Gamma; \Delta \overset{P,N}{\Longrightarrow} F \text{ at } z$$

It states that from global resources Γ and local resources Δ, we can reach the goal F at place z in a network with places P and edges labelled N. We have summarized the sequent calculus rules in the Appendix.

We let the judgment

$$\Gamma; \Delta \overset{P,N}{\Longrightarrow}^{-} F \text{ at } z$$

denote the valid sequent calculus judgments constructed without using the cut rule. Using Pfenning's structural cut elimination technique [10], we are able to prove that the sequent calculus without cut is just as powerful as the sequent calculus with cut.

Theorem 5 (Cut Elimination)
If $\Gamma; \Delta \overset{P,N}{\Longrightarrow} F \text{ at } z$ *then* $\Gamma; \Delta \overset{P,N}{\Longrightarrow}^{-} F \text{ at } z$.

Moreover, the sequent calculus with cut corresponds exactly to the natural deduction style presentation of the logic.

Theorem 6 (Sequent Soundness and Completeness)

1. *If* $\Gamma; \Delta \overset{P,N}{\Longrightarrow} F \text{ at } z$ *then* $\Gamma; \Delta \vdash^{P,N} F \text{ at } z$.
2. *If* $\Gamma; \Delta \vdash^{P,N} F \text{ at } z$ *then* $\Gamma; \Delta \overset{P,N}{\Longrightarrow} F \text{ at } z$.

Types $\tau ::=$

$$\mathbf{b} \mid \top \mid \tau_1 \to \tau_2 \mid \tau_1 \wedge \tau_2 \mid \tau \mathbin{@} z \mid n[\tau] \mid \Box\,\tau \mid \Cap\tau \mid \Diamond\tau$$

Proof Terms/Programs $e ::=$

$c \mid x \mid \mathbf{sync}\,(x) \mid \mathbf{run}\,(\,x\,[\,z\,]\,) \mid ()$	*const/var/*$\top$
$\mid \lambda x{:}\tau.e \mid e_1\ e_2$	*functions*$(\to)$
$\mid \langle\, e_1\,,\, e_2\,\rangle \mid \pi_i e$	*pairs* $(\wedge)$
$\mid \mathbf{ret}^{\mathsf{abs}}(e,z) \mid \mathbf{rpc}^{\mathsf{abs}}(e,z)$	*rpc abs.* $(\,@\,)$
$\mid \mathbf{ret}^{\mathsf{rel}}(e,n) \mid \mathbf{rpc}^{\mathsf{rel}}(e,z)$	*rpc rel.* $([\,])$
$\mid \mathbf{close}(\lambda p.\,e) \mid \mathbf{bc}\,e_1\ \mathsf{at}\ z\ \mathbf{as}\,x\,\mathbf{in}\,e_2$	*broadcast* $(\Box)$
$\mid \mathbf{port}(\lambda p.\,e) \mid \mathbf{pull}\,e_1\ \mathsf{at}\ z\ \mathbf{as}\,x\,\mathbf{in}\,e_2$	*portable* $(\Cap)$
$\mid \mathbf{agent}[e,z] \mid \mathbf{go}\,e_1\ \mathsf{at}\ z\ \mathbf{return}\,x,p\,\mathbf{in}\,e_2$	*agent* $(\Diamond)$

Figure 1: λ_{rpc} Syntax

3 λ_{rpc}: A Distributed Programming Language

The previous section developed an intuitionistic, modal logic capable of concisely expressing facts about the placement of various objects in a network. Here, we present the proof terms of logic and show how they may be given an operational interpretation as a distributed programming language that we call λ_{rpc}. The logical formulas serve as types that prevent distributed programs from "going wrong" by attempting to access resources that are unavailable at the place the program is currently operating.

3.1 Syntax and Typing

Figure 1 presents the syntax of programs and their types, and Figure 2 presents the typing rules for the language, which are the natural deduction-style proof rules for the logic.

Types and Typing Judgments The types correspond to the formulas of the logic; we use the meta variable τ rather than F to indicate a shift in interpretation. We also let $\mathbf{b}$ range over various base types we might interested in.

One other change between logic and language is that the language interprets $\Box$ in two different ways. To avoid confusion, the language has two distinct, but logically identical, types, $\Box\,\tau$ and $\Cap\tau$. Analogously, we separate the logical context Γ into two parts $\Gamma_{\Box}$ and $\Gamma_{\Cap}$ during type checking. Hence the overall type checking judgment has the following form.

$$\Gamma_{\Box}\,;\Gamma_{\Cap};\Delta \vdash^{P,N} e : \tau\ \mathsf{at}\ z$$

$$\frac{}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} c : \mathbf{b} \text{ at } z}\ const \qquad \frac{}{\Gamma_\Box;\Gamma_\boxplus;\Delta, x:\tau \text{ at } z \vdash^{P,N} x:\tau \text{ at } z}\ L$$

$$\frac{}{\Gamma_\Box, x:\tau;\Gamma_\boxplus;\Delta \vdash^{P,N} \mathbf{sync}\,(x) : \tau \text{ at } z}\ G\Box \qquad \frac{}{\Gamma_\Box;\Gamma_\boxplus, x:\tau;\Delta \vdash^{P,N} \mathbf{run}\,(\,x\,[\,z\,]\,) : \tau \text{ at } z}\ G\boxplus$$

$$\frac{}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} () : \top \text{ at } z}\ \top I$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta, x:\tau_1 \text{ at } z \vdash^{P,N} e:\tau_2 \text{ at } z}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \lambda x{:}\tau_1.e : \tau_1 \to \tau_2 \text{ at } z} \to I$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e_1 : \tau_1 \to \tau_2 \text{ at } z \quad \Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e_2 : \tau_1 \text{ at } z}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e_1\, e_2 : \tau_2 \text{ at } z} \to E$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e_1 : \tau_1 \text{ at } z \quad \Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e_2 : \tau_2 \text{ at } z}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \langle\, e_1\,,\, e_2\,\rangle : \tau_1 \times \tau_2 \text{ at } z} \wedge I \qquad \frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e : \tau_1 \times \tau_2 \text{ at } z}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \pi_i e : \tau_i \text{ at } z} \wedge E$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P+p,N} e : \tau \text{ at } z \quad p \notin FP(z) \cup FP(\Gamma_\Box) \cup FP(\Gamma_\boxplus) \cup FP(\Delta)}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \Lambda p{:}places\,.\,e : \forall p.\tau \text{ at } z} \forall I$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e : \forall p.\tau \text{ at } z \quad FPN(z') \subseteq P \cup N}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e\,[\,z'\,] : \tau\,[\,z'\,/\,p\,] \text{ at } z} \forall E$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e : \tau \text{ at } z}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \mathbf{ret}^{\mathsf{abs}}(e,z) : \tau \mathbin{@} z \text{ at } z'} @\, I \qquad \frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e : \tau \mathbin{@} z \text{ at } z'}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \mathbf{rpc}^{\mathsf{abs}}(e,z') : \tau \text{ at } z} @\, E$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e : \tau \text{ at } z.n}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \mathbf{ret}^{\mathsf{rel}}(e,n) : n[\tau] \text{ at } z} [\,]\, I \qquad \frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e : n[\tau] \text{ at } z}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \mathbf{rpc}^{\mathsf{rel}}(e,z) : \tau \text{ at } z.n} [\,]\, E$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P+p,N} e : \tau \text{ at } p \quad p \notin \mathsf{FP}(\Gamma_\Box) \cup \mathsf{FP}(\Gamma_\boxplus) \cup \mathsf{FP}(\Delta) \cup \mathsf{FP}(\tau)}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \mathbf{close}(\lambda p.\, e) : \Box\,\tau \text{ at } z} \Box\, I$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e_1 : \Box\,\tau \text{ at } z \quad \Gamma_\Box, x:\tau;\Gamma_\boxplus;\Delta \vdash^{P,N} e_2 : \tau' \text{ at } z'}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \mathbf{bc}\, e_1 \text{ at } z \,\mathbf{as}\, x \,\mathbf{in}\, e_2 : \tau' \text{ at } z'} \Box\, E$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P+p,N} e : \tau \text{ at } p \quad p \notin \mathsf{FP}(\Gamma_\Box) \cup \mathsf{FP}(\Gamma_\boxplus) \cup \mathsf{FP}(\Delta) \cup \mathsf{FP}(\tau)}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \mathbf{port}(\lambda p.\, e) : \boxplus\tau \text{ at } z} \boxplus I$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e_1 : \boxplus\tau \text{ at } z \quad \Gamma_\Box;\Gamma_\boxplus, x:\tau;\Delta \vdash^{P,N} e_2 : \tau' \text{ at } z'}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \mathbf{pull}\, e_1 \text{ at } z \,\mathbf{as}\, x \,\mathbf{in}\, e_2 : \tau' \text{ at } z'} \boxplus E$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e : \tau \text{ at } z}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \mathbf{agent}[e,z] : \Diamond\tau \text{ at } z'} \Diamond I$$

$$\frac{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} e_1 : \Diamond\tau \text{ at } z \quad \Gamma_\Box;\Gamma_\boxplus;\Delta, x:\tau \text{ at } p \vdash^{P+p,N} e_2 : \tau' \text{ at } z' \quad p \notin \mathsf{FP}(\tau') \cup \mathsf{FP}(z')}{\Gamma_\Box;\Gamma_\boxplus;\Delta \vdash^{P,N} \mathbf{go}\, e_1 \text{ at } z \,\mathbf{return}\, x, p \,\mathbf{in}\, e_2 : \tau' \text{ at } z'} \Diamond E$$

Figure 2: $\mathfrak{B}_{\mathsf{pc}}$ Typing

Programs The programs include an unspecified set of constants (c), and the standard introduction and elimination forms for unit, functions and pairs.

Variables from each different context are used in different ways. As a mnemonic for the different sorts of uses, we have added some syntactic sugar to the standard proof terms. Uses of local variables from Δ are just like ordinary uses of variables in your favorite (call-by-value) functional language so they are left undecorated. Variables in $\Gamma_{\Box}$ refer to computations that have been broadcast at some earlier point. In order to use such a variable, the program must *synchronize* with the concurrently executing computation. Hence, we write $\mathbf{sync}\,(x)$ for such uses. Variables in $\Gamma_{\boxplus}$ refer to portable closures. The use of a variable in this context corresponds to *running* the closure with the current place z as an argument. Hence, we write $\mathbf{run}\,(\,x\,[\,z\,]\,)$ for such uses.

Our first modalities $\tau\,@\,z$ has an operational interpretation as a remote procedure call. The introduction form ($\mathbf{ret}^{\mathsf{abs}}(e, z)$) constructs a "return value" for a remote procedure call. This "return value" can actually be an arbitrary expression e, which will be run at the place it is returned to. The elimination form ($\mathbf{rpc}^{\mathsf{abs}}(e, z)$) the remote procedure call itself. It sends the expression e to a remote site where e will be evaluated. If the expression is well-typed, it will eventually compute a return value that can be run safely at the caller's place.

The interpretation of $n[\tau]$ is quite similar to the interpretation of $\tau\,@\,z$. The main difference is that the return value $\mathbf{ret}^{\mathsf{rel}}(e, n)$ is constructed for a place that may be found by following the link named n from the the place where the remote procedure call executes. The elimination form ($\mathbf{rpc}^{\mathsf{rel}}(e, z)$) performs the remote procedure call by sending e to z, when the current place is $z.n$.

The introduction form for $\Box\,F$ is $\mathbf{close}(\lambda p.\,e)$. It creates a closure that may be *broadcast* by the elimination form ($\mathbf{bc}\,e_1\ \mathsf{at}\ z\ \mathbf{as}\,x\,\mathbf{in}\,e_2$) to every node in the network. More specifically, the elimination form executes e_1 at z, expecting e_1 to evaluate to $\mathbf{close}(\lambda p.\,e)$. When it does, the broadcast expression chooses a new universal reference for the closure, which is bound to x, and sends $\lambda p.e$ to every place in the network where it is applied to the current place and the resulting expression is associated with its universal reference. Remote procedure calls or broadcasts generated during evaluation of e_2 may refer to the universal reference bound to x, which is safe, since x has been broadcast everywhere.

Objects of type $\boxplus\tau$ are portable closures; they never contain local references and consequently may be run anywhere. The elimination form ($\mathbf{pull}\,e_1\ \mathsf{at}\ z\ \mathbf{as}\,x\,\mathbf{in}\,e_2$) takes advantage of this portability by first computing e_1 at z, which should result in a value with the form $\mathbf{port}(\lambda p.\,e)$. Next, it pulls the closure $\lambda p.e$ from z and substitutes it for x in e_2. The typing rules will allow x to appear anywhere, including in closures in e_2 that will eventually be broadcast or remotely executed. Once again, this is safe since e is portable and runs equally well everywhere.

Our last connective $\Diamond\tau$ is considered the type of a computational agent that is smart enough to know where it can go to produce a value with type τ. We introduce such an agent by packaging an expression with a place where the expression may successfully be run to completion. The elimination form ($\mathbf{go}\,e_1\ \mathsf{at}\ z\ \mathbf{return}\,x, p\,\mathbf{in}\,e_2$) first evaluates e_1 at z, producing an agent ($\mathbf{agent}[e, z']$).

Next, it commands the agent go to the hidden place z' and execute its encapsulated computation there. When the agent has completed its task, it synchronizes with the current computation and e_2 continues with p bound to z' and x bound to a value that is safe to use at z'.

Simple examples To gain a little more intuition about how to write programs in this language, consider computational interpretations of some of the proofs from the previous section. The context Δ referenced below contains the following assumptions.

$$\begin{array}{ll} \mathsf{f}_D : \mathsf{pdf}\ \mathsf{at}\ D & \mathsf{print} : \mathsf{pdf} \wedge \mathsf{pt} \to \mathsf{po}\ \mathsf{at}\ E \\ \mathsf{f}_E : \mathsf{pdf}\ \mathsf{at}\ E & \mathsf{DtoE} : \mathsf{pdf} \to \mathsf{pdf} \mathbin{@} E\ \mathsf{at}\ D \\ \mathsf{ptr}_E : \mathsf{pt}\ \mathsf{at}\ E & \mathsf{ToE} : \Box\,(\mathsf{pdf} \to \mathsf{pdf} \mathbin{@} E)\ \mathsf{at}\ E \end{array}$$

Printing a PDF file (involving local computation only):

$$\cdot; \Delta \vdash \mathsf{print}(\mathsf{f}_E, \mathsf{ptr}_E) : \mathsf{po}\ \mathsf{at}\ E$$

Fetching a PDF file (involving a remote procedure call in which the computation $\mathsf{DtoE}\ \mathsf{f}_D$ is executed at D):

$$\cdot; \Delta \vdash \mathbf{rpc}^{\mathsf{abs}}(\mathsf{DtoE}\ \mathsf{f}_D, D) : \mathsf{pdf}\ \mathsf{at}\ E$$

Fetching then printing:

$$\cdot; \Delta \vdash (\lambda x{:}\mathsf{pdf}.\mathsf{print}\ (x, \mathsf{ptr}_E))(\mathbf{rpc}^{\mathsf{abs}}(\mathsf{DtoE}\ \mathsf{f}_D, D)) : \mathsf{po}\ \mathsf{at}\ E$$

Broadcasting E's PDF conversion function to all nodes then fetching a PDF file from node q (recall that in general, uses of these global variables involves synchronizing with the broadcast expression; below the broadcast expression is a value, but we synchronize anyway):

$$\begin{array}{ll} \cdot; \Delta, \mathsf{f}_q : \mathsf{pdf}\ \mathsf{at}\ q \vdash & \mathbf{bc}\ \mathsf{ToE}\ \mathsf{at}\ E\ \mathbf{as}\ \mathsf{ToE}'\ \mathbf{in} \\ & \mathbf{rpc}^{\mathsf{abs}}(\mathbf{sync}\,(\mathsf{ToE}')\ \mathsf{f}_q, q) \quad : \mathsf{pdf}\ \mathsf{at}\ E \end{array}$$

Broadcasting E's PDF conversion function to all nodes then fetching a PDF file from multiple nodes ($\mathbf{let}\, x\ =\ e_1\ \mathbf{in}\ e_2$ is an abbreviation for the usual lambda abstraction and application):

$$\begin{array}{l} \cdot; \Delta, \mathsf{f}_C : \mathsf{pdf}\ \mathsf{at}\ C, \mathsf{f}_B : \mathsf{pdf}\ \mathsf{at}\ B \vdash \\ \quad \mathbf{bc}\ \mathsf{ToE}\ \mathsf{at}\ E\ \mathbf{as}\ \mathsf{ToE}'\ \mathbf{in} \\ \quad \mathbf{let}\ f_1\ =\ \mathbf{rpc}^{\mathsf{abs}}(\mathbf{sync}\,(\mathsf{ToE}')\ \mathsf{f}_D, D)\ \mathbf{in} \\ \quad \mathbf{let}\ f_2\ =\ \mathbf{rpc}^{\mathsf{abs}}(\mathbf{sync}\,(\mathsf{ToE}')\ \mathsf{f}_C, C)\ \mathbf{in} \\ \quad \mathbf{let}\ f_3\ =\ \mathbf{rpc}^{\mathsf{abs}}(\mathbf{sync}\,(\mathsf{ToE}')\ \mathsf{f}_B, B)\ \mathbf{in} \\ \quad \ldots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad : \tau\ \mathsf{at}\ E \end{array}$$

Another way to manage PDF files is to make them portable. For instance, if C and D contain portable PDF files, then E can pull these files from their resident

Syntax:

$$\begin{array}{rcl} \tau & ::= & \cdots \mid \tau \ \mathbf{ref} \\ e & ::= & \cdots \mid \mathbf{ref}\ e \mid !\, e \mid e_1 := e_2 \end{array}$$

Typing:

$$\frac{\Gamma_\square; \Gamma_\natural; \Delta \vdash^{P,N} e : \tau \ \mathsf{at}\ z}{\Gamma_\square; \Gamma_\natural; \Delta \vdash^{P,N} \mathbf{ref}\ e : \tau\ \mathbf{ref}\ \mathsf{at}\ z}\ \mathit{ref}$$

$$\frac{\Gamma_\square; \Gamma_\natural; \Delta \vdash^{P,N} e : \tau\ \mathbf{ref}\ \mathsf{at}\ z}{\Gamma_\square; \Gamma_\natural; \Delta \vdash^{P,N} !\, e : \tau\ \mathsf{at}\ z}\ !$$

$$\frac{\Gamma_\square; \Gamma_\natural; \Delta \vdash^{P,N} e_1 : \tau\ \mathbf{ref}\ \mathsf{at}\ z \quad \Gamma_\square; \Gamma_\natural; \Delta \vdash^{P,N} e_2 : \tau\ \mathsf{at}\ z}{\Gamma_\square; \Gamma_\natural; \Delta \vdash^{P,N} e_1 := e_2 : \top\ \mathsf{at}\ z}\ :=$$

Figure 3: Syntax and Typing for Effectful Operations

locations and print them on its local printer. Remember that portable values are polymorphic closures that are "run" when used. In this case, the closure simply returns the appropriate PDF file.

$$\begin{array}{l} \cdot; \Delta, \mathsf{f}_C : \natural\mathsf{pdf}\ \mathsf{at}\ C, \mathsf{f}_D : \natural\mathsf{pdf}\ \mathsf{at}\ D \vdash \\ \quad \mathbf{pull}\, \mathsf{f}_C\ \mathsf{at}\ C\ \mathbf{as}\, \mathsf{f}'_C\ \mathbf{in} \\ \quad \mathbf{pull}\, \mathsf{f}_D\ \mathsf{at}\ D\ \mathbf{as}\, \mathsf{f}'_D\ \mathbf{in} \\ \quad \mathbf{let}\ \_ = \mathsf{print}(\mathbf{run}\,(\,\mathsf{f}'_C\ [\,E\,]\,), \mathsf{ptr}_E)\ \mathbf{in} \\ \quad \mathbf{let}\ \_ = \mathsf{print}(\mathbf{run}\,(\,\mathsf{f}'_D\ [\,E\,]\,), \mathsf{ptr}_E)\ \mathbf{in} \\ \quad \ldots \qquad\qquad : \tau\ \mathsf{at}\ E \end{array}$$

3.2 Effectful Extensions

To begin to convince ourselves that our little lambda calculus can be scaled up to the point that it could serve as a practical distributed programming language, we have examined one of the trickiest but most useful effectful extensions to the language, mutable references. Our design introduces just the right type structure for the various modalities to ensure that assignment and dereference of mutable references can only happen locally. In other words, well-typed programs never attempt an assignment operation at one place, when the ref being assigned to is stored somewhere else.

The syntax and typing rules for this extension appears in Figure 3. The main point to note about the typing rules is that like expressions manipulating unit, pairs and functions, if the expressions manipulating references or recursive functions are placed at z, then the appropriate subexpressions must also be placed at z.

Suppose we have decided to use our network as a distributed database of technical reports and that every node supports a function $\mathsf{db} : \mathsf{key} \to \natural\mathsf{pdf}\ \mathsf{option}$.

We will let τ option be the usual disjoint union type with constructors `None` and `Some`, and destructor `case`. Now node E can write code for a distributed lookup using key k : ⌑key at E. For clarity in the code below, we do not annotate uses of the global variables with "`run`" or "`sync`;" it is trivial to reconstruct these annotations from the context.

```
let r : ⌑pdf option ref = ref None in
let search : □⊤ =
  close(λp.
    pull k at E as k' in
    case (db k') of
      None   => ()
    | Some f => pull f at p as f' in
                rpc(r := Some (port(λq.f'));
                    ret((),p),
                    E))
in bc search at E as __ in
...
```

3.3 Operational Semantics and Safety

To give an operational semantics for our programming language, we deviate from the Curry-Howard tradition that would suggest using proof simplification as program evaluation. An operational semantics based exclusively on proof simplification would fail to model the action of distributing resources across a network properly. Instead, we develop a relatively concrete notion of a network, and explicitly allocate processes (expressions) together at different places in the network. Figure 4 presents the various new syntactic objects we use to specify our operational model.

Run-time Structures Networks $\mathcal{N}$ are 5-tuples consisting of a set of places P, a set of names N, an edge function E, a distributed process environment $\mathcal{L}$ and a distributed storage system $\mathcal{M}$. We have seen places and names before. The edge function E is a total function from pairs of places and names to places; it is used to interpret relative addressing. If z is a path consisting of places and names from P and N, we can interpret it using the function E^*:

$$\frac{}{E^*(p) = p} \qquad \frac{E^*(z) = p \quad E(p, n) = q}{E^*(z.n) = q}$$

We define $z_1 \equiv_E z_2$ to be $E^*(z_1) = E^*(z_2)$.

The process environment $\mathcal{L}$ is a finite partial map from places p in P to process ids to expressions. The distributed storage system $\mathcal{M}$ is a finite partial map from places to mutable storage locations (m) to values v. We write these

$$
\begin{array}{llll}
\textit{Networks} & \mathcal{N} & ::= & (P, N, E, \mathcal{L}, \mathcal{M}) \\
\textit{Edge functions} & E & : & P \times N \to P \\
\textit{Process Envs.} & \mathcal{L} & ::= & \cdot \mid \mathcal{L}, \ell \to e \;\mathsf{at}\; p \\
\textit{Stores} & \mathcal{M} & ::= & \cdot \mid \mathcal{M}, m \to v \;\mathsf{at}\; p \\
\textit{Values} & v & ::= & c \mid m \mid \lambda x{:}\tau.e \mid \langle\, v_1 \,,\, v_2 \,\rangle \\
 & & & \mid \mathbf{ret}^{\mathsf{abs}}(e, z) \mid \mathbf{ret}^{\mathsf{rel}}(e, n) \\
 & & & \mid \mathbf{close}(\lambda p.\, e) \mid \mathbf{port}(\lambda p.\, e) \\
 & & & \mid \mathbf{agent}[e, z] \\
\textit{RT Terms} & e & ::= & \cdots \mid \mathbf{sync}\,(\ell) \mid \mathbf{run}\,(\,\lambda p.e\,[\,z\,]\,) \mid m \\
 & & & \mid \mathbf{sync}\,(\mathbf{rpc}^{\mathsf{abs}}(\ell, z)) \\
 & & & \mid \mathbf{sync}\,(\mathbf{rpc}^{\mathsf{rel}}(\ell, z)) \\
 & & & \mid \mathbf{sync}\,(\mathbf{bc}\,\ell \;\mathsf{at}\; z \;\mathbf{as}\, x \,\mathbf{in}\, e_2) \\
 & & & \mid \mathbf{sync}\,(\mathbf{pull}\,\ell \;\mathsf{at}\; z \;\mathbf{as}\, x \,\mathbf{in}\, e_2) \\
 & & & \mid \mathbf{sync}_1(\mathbf{go}\,\ell \;\mathsf{at}\; z \;\mathbf{return}\, x, q \,\mathbf{in}\, e) \\
 & & & \mid \mathbf{sync}_2(\mathbf{go}\,\ell \;\mathsf{at}\; z \;\mathbf{return}\, x, q \,\mathbf{in}\, e) \\
\textit{Contexts} & C & ::= & [\,] \mid C\, e_2 \mid v_1\, C \\
 & & & \mid \langle\, C \,,\, e_2 \,\rangle \mid \langle\, v_1 \,,\, C \,\rangle \mid \pi_i\; C \\
 & & & \mid \mathbf{ref}\; C \mid C := e_1 \mid v := C \mid !\; C
\end{array}
$$

Figure 4: Syntax of Run-time Structures

partial maps as lists of elements with the form $\ell \to e \;\mathsf{at}\; p$ or $m \to v \;\mathsf{at}\; p$. Whenever we write such a map, we assume that no pair of place and location (p and ℓ, or p and m) appears in two different components of the map. We do not distinguish between maps that differ only in the ordering of their elements. $\mathcal{L}(p)(\ell)$ denotes e when $\mathcal{L} = \mathcal{L}', \ell \to e \;\mathsf{at}\; p$, and similarly with $\mathcal{M}(p)(m)$. We use the notation $\mathcal{L} \setminus \ell$ to denote the mapping $\mathcal{L}$ with all elements of the form $\ell \to e \;\mathsf{at}\; p$ removed. We emphasize that *all* elements are removed as there may be one such element at every place in the network.

In order to give an operational semantics to our programs, a few of the constructs require that we introduce new expressions (the RT terms in Figure 4) that only occur at run time. For instance, mutable locations can appear in expressions at run time, but should not appear in unevaluated programs. Other run-time terms are used to represent expressions, such as the elimination forms for $\Box$ and $\Diamond$, that are suspended partway through evaluation and are waiting to synchronize with remotely executing expressions.

The last new bit of notation that we need involves the definition of evaluation contexts C (see Figure 4). These contexts specify the order of evaluation, which is left to right and call by value. Notice that there are no contexts for the introduction forms for $@$, $n[\,]$, $\Box$, $\boxplus$ or $\Diamond$, and consequently evaluation does not proceed under these constructors.

Operational Rules The state of a network $\mathcal{N} = (P, N, E, \mathcal{L}, \mathcal{M})$ evolves according to the operational rules listed in Figure 3.3. These rules specify a relation with the form $\mathcal{L}, \mathcal{M} \longmapsto \mathcal{L}', \mathcal{M}'$. At any given time $\mathcal{L}$ may contain several concurrently executing expressions at various stages of evaluation. Any of these expressions may be selected for a single step of evaluation, so the judgment gives rise to a nondeterministic operational semantics. The network evolves ($\mathcal{N} \longmapsto \mathcal{N}'$) if and only if $\mathcal{N} = (P, N, E, \mathcal{L}, \mathcal{M})$, and $\mathcal{N}' = (P, N, E, \mathcal{L}', \mathcal{M}')$, and $(\mathcal{L}, \mathcal{M}) \longmapsto (\mathcal{L}', \mathcal{M}')$.

Most of the formal rules have been described informally in the previous subsection, so we will only make a few points here. First, notice that we have two special rules **sync** and **run** to handle synchronization with broadcast expressions and execution of portable code. Second, the ordinary sequential programming language constructs (functions, pairs, references, etc.) operate identically to the way they operate in ordinary sequential programming languages. Third, each of the modalities has two or more rules to describe their evaluation. Typically, the first rule causes the elimination form for a modality to spawn an expression at a remote node and then to move into a state in which it waits to synchronize with that expression. The second (or third) rule performs the appropriate synchronization and allows evaluation to continue.

Typing for Run-time Structures To carry out our safety proof for the language, we use the standard Progress and Preservation techniques, which require that we be able to show that the network is well-typed at every step in evaluation. In order to do so, we need to generalize the form of the typing judgment to take acount of the way that relative naming is interpreted by the network's edge function E. The generalized judgment has the form $\Gamma_{\square}; \Gamma_{\boxplus}; \Delta \vdash^{P,N,E} e : \tau \ \mathsf{at}\ z$. None of the typing rules in the previous sections change, aside from propagating E from conclusions to premises. However, we add the rules from Figure 5 to give types to the intermediate forms of expression. Figure 6 gives the well-formedness conditions for the network as a whole.

Type Safety The type system is sound with respect to our operational semantics for distributed program evaluation. The proofs of Preservation and Progress theorems, stated below, follow the usual strategy.

Theorem 7 (Preservation)
If $\vdash \mathcal{N} : \Gamma_{\square}; \cdot; \Delta; P; N$ *and* $\mathcal{N} \longmapsto \mathcal{N}'$ *then there exists* $\Gamma'_{\square}$ *and* Δ' *such that* $\vdash \mathcal{N}' : \Gamma'_{\square}; \cdot; \Delta'; P; N$.

Theorem 8 (Progress)
If $\vdash \mathcal{N} : \Gamma_{\square}; \cdot; \Delta; P; N$ *then either*

- $\mathcal{N} \longmapsto \mathcal{N}'$, *or*
- $\mathcal{N} = (P, N, E, \mathcal{L}, \mathcal{M})$ *and for all places* p *in* P, *and for all* ℓ *in* $\mathsf{Dom}(\mathcal{L}(p))$, $\mathcal{L}(p)(\ell)$ *is a value.*

4 Discussion

Extensions and Variations This paper presents a solid foundation on which to build a distributed functional programming language. However, in terms of language design, it is only the beginning. A few interesting extensions and variations are discussed briefly below.

- Values everywhere. Our interpretation of $\Box$ involves either broadcasting a closure or substituting a closure into local code. In each case, there is some computational overhead to manage the closure: Either we synchronize or we run the closure when it gets used. To avoid this overhead, we could place a value restriction on the expression in the introduction form for $\Box$. One possibility that is definitely not an option is evaluating eagerly under $\Box$ before broadcasting or substitution: In the presence of references (and almost certainly other effects), this evaluation strategy is unsound.

- Dynamic network evolution. The current work assumes that the set of network places and the network topology is fixed. While this is a reasonable assumption for some distributed programming environments, others allow the topology to evolve. An interesting challenge for future work is to extend our logic and language with features that express evolution. We believe that the new name connectives developed in the context of nominal logics [13, 9] may be of help here.

- Synchronous and asynchronous variations. Just as ordinary sequential programming languages may be defined with different evaluation strategies (call-by-value, call-by-name, call-by-need), it appears possible to develop different operational interpretations of the modal connectives in which execution is more or less synchronized. For instance, when defining the operation of the $\Box$-connective, we could wait until all broadcast expressions have completed evaluation before proceeding with the evaluation of the second expression e_2. Likewise, remote procedure calls are synchronized: Evaluation does not proceed until they have received the return value, even though the following computation does not necessarily require the value immediately. In the future, we plan to explore these nuances in greater detail.

Related Work Hybrid logics are an old breed of logic that date back to Arthur Prior's work in the 1960s [14]. As in our logic, they mix modal necessity and possibility with formulas such as $F \mathbin{@} z$ that are built from pure names. More recently, researchers have developed a rich semantic theory for these logics and studied both tableau proofs and sequents; many resources on these topics and others are available off the hybrid logics web page.[3] However, work on hybrid logics is usually carried out in a classical setting and we have not found an

[3] See `http://www.hylo.net`.

intuitionistic, natural deduction style proof theory like ours that can serve as a foundation for distributed functional programming languages.

Cardelli and Gordon's ambient logic [2] highlights the idea that modalities for possibility and necessity need not only be interpreted *temporally*, but can also be interpreted *spatially*, and this abstract idea was a central influence in our work. However, at a more technical level, the ambient logic is entirely different from the logic we develop here: The ambient logic has a widely different set of connectives, is classical as opposed to intuitionistic, and is defined exclusively by a sequent calculus rather than by natural deduction. Moreover, it does not serve as a type system for ambient programs; rather, it is a tool for reasoning about them.

Another major influence on our work is Pfenning and Davies' judgmental reconstruction of modal logic [12], which is developed in accordance with Martin Löf's design patterns for type theory [8]. Pfenning and Davies go on to interpret modal necessity temporally (as opposed to spatially) in their work on staged computation [4]. One obvious technical difference between our logic and theirs is that our logic is founded on local judgments that include the specific place where a proposition is true whereas theirs do not.

The judgments $\vdash F \ \mathsf{at} \ z$ also appear in our own recent work with Ahmed [1], where we combine the modal connective $n[\tau]$ with substructural connectives to reason about region-based memory management in a proof-carrying code setting.

Concurrently with this research, members of the CMU Concert Project have begun to build a programming language for grid computing, and Harper, Moody and Pfenning have had similar insights as us with respect to the roll that modal logics may play in developing a foundation for distributed computing. More specifically, they interpret objects with type $\Box\,\tau$ as jobs that may be injected into the grid and run anywhere; objects with type $\Diamond\tau$ also have computational significance in their language.[4] In their application domain, every node is assumed to contain identical resources, so they are investigating type systems derived from pure modal logics rather than hybrid logics like the one we have presented here.

References

[1] A. Ahmed, L. Jia, and D. Walker. Reasoning about hierarchical storage. In *IEEE Symposium on Logic in Computer Science*, pages 33–44, Ottawa, Canada, June 2003.

[2] L. Cardelli and A. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Twenty-Seventh ACM Symposium on Principles of Programming Languages*, pages 365–377. ACM Press, Jan. 2000.

[3] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, June 2000.

[4] Personal communication, Robert Harper and Frank Pfenning, June 2003.

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

[5] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421, Pisa, Italy, Aug. 1996. Springer.

[6] M. R. A. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, England, 2000.

[7] L. Jia and D. Walker. Modal proofs as distributed programs. Technical Report TR-671-03, Princeton University, 2003. Forthcoming.

[8] M. Löf. On the meanings of the logical constants and the justifications of the logical laws. Technical Report 2, University of Siena, 1985.

[9] D. Miller and A. Tiu. A proof theory for generic judgments. In *IEEE Symposium on Logic in Computer Science*, pages 118–127, Ottawa, Canada, June 2003.

[10] F. Pfenning. Structural cut elimination I. intuitionistic and classical logic. *Information and Computation*, 157(1/2):84–141, Mar. 2000.

[11] F. Pfenning. Logical frameworks. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 16, pages 977–1061. Elsevier Science and MIT Press, 2001.

[12] F. Pfenning and R. Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11:511–540, 2001.

[13] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and computation*, 2003. To appear.

[14] A. Prior. *Past, present and future*. Oxford University press, 1967.

A Natural deduction rules

$$\frac{}{\Gamma; \Delta, F \text{ at } z \vdash^{P,N} F \text{ at } z}\ L \qquad \frac{}{\Gamma, F; \Delta \vdash^{P,N} F \text{ at } z}\ G$$

$$\frac{}{\Gamma; \Delta \vdash^{P,N} () : \top \text{ at } z}\ \top I$$

$$\frac{\Gamma; \Delta, F_1 \text{ at } z \vdash^{P,N} F_2 \text{ at } z}{\Gamma; \Delta \vdash^{P,N} F_1 \to F_2 \text{ at } z} \to I$$

$$\frac{\Gamma; \Delta \vdash^{P,N} F_1 \to F_2 \text{ at } z \quad \Gamma; \Delta \vdash^{P,N} F_1 \text{ at } z}{\Gamma; \Delta \vdash^{P,N} F_2 \text{ at } z} \to E$$

$$\frac{\Gamma;\Delta \vdash^{P,N} F_1 \text{ at } z \quad \Gamma;\Delta \vdash^{P,N} F_2 \text{ at } z}{\Gamma;\Delta \vdash^{P,N} F_1 \wedge F_2 \text{ at } z} \wedge I$$

$$\frac{\Gamma;\Delta \vdash^{P,N} F_1 \wedge F_2 \text{ at } z}{\Gamma;\Delta \vdash^{P,N} F_1 \text{ at } z} \wedge E1 \qquad \frac{\Gamma;\Delta \vdash^{P,N} F_1 \wedge F_2 \text{ at } z}{\Gamma;\Delta \vdash^{P,N} F_2 \text{ at } z} \wedge E2$$

$$\frac{\Gamma;\Delta \vdash^{P+p,N} F \text{ at } z \quad p \notin \mathsf{FP}(\Gamma) \cup \mathsf{FP}(\Delta) \cup \mathsf{FP}(z)}{\Gamma;\Delta \vdash^{P,N} \forall p.F \text{ at } z} \forall I$$

$$\frac{\Gamma;\Delta \vdash^{P,N} \forall p.F \text{ at } z \quad FPN(z') \subseteq P \cup N}{\Gamma;\Delta \vdash^{P,N} F\,[\,z'\,/\,p\,] \text{ at } z} \forall E$$

$$\frac{\Gamma;\Delta \vdash^{P,N} F \text{ at } z.n}{\Gamma;\Delta \vdash^{P,N} n[F] \text{ at } z}\,[\,]I \qquad \frac{\Gamma;\Delta \vdash^{P,N} n[F] \text{ at } z}{\Gamma;\Delta \vdash^{P,N} F \text{ at } z.n}\,[\,]E$$

$$\frac{\Gamma;\Delta \vdash^{P,N} F \text{ at } z}{\Gamma;\Delta \vdash^{P,N} F\,@\,z \text{ at } z'}\,@\,I \qquad \frac{\Gamma;\Delta \vdash^{P,N} F\,@\,z \text{ at } z'}{\Gamma;\Delta \vdash^{P,N} F \text{ at } z}\,@\,E$$

$$\frac{\Gamma;\Delta \vdash^{P+p,N} F \text{ at } p \quad p \notin \mathsf{FP}(\Gamma) \cup \mathsf{FP}(\Delta) \cup \mathsf{FP}(F)}{\Gamma;\Delta \vdash^{P,N} \Box F \text{ at } z} \Box I$$

$$\frac{\Gamma;\Delta \vdash^{P,N} \Box F \text{ at } z \quad \Gamma, F;\Delta \vdash^{P,N} F' \text{ at } z'}{\Gamma;\Delta \vdash^{P,N} F' \text{ at } z'} \Box E$$

$$\frac{\Gamma;\Delta \vdash^{P,N} F \text{ at } z}{\Gamma;\Delta \vdash^{P,N} \Diamond F \text{ at } z'} \Diamond I$$

$$\frac{\Gamma;\Delta \vdash^{P,N} \Diamond F \text{ at } z \qquad \Gamma;\Delta, F \text{ at } p \vdash^{P+p,N} F' \text{ at } z' \quad p \notin \mathsf{FP}(F') \cup \mathsf{FP}(z')}{\Gamma;\Delta \vdash^{P,N} F' \text{ at } z'} \Diamond E$$

B Sequent caculus rules

$$\frac{}{\Gamma;\Delta, F \text{ at } z \overset{P,N}{\Longrightarrow} F \text{ at } z}\ \textit{L-INIT}$$

$$\frac{}{\Gamma, F;\Delta \overset{P,N}{\Longrightarrow} F \text{ at } z}\ \textit{G-INIT}$$

$$\frac{\Gamma, F;\Delta, F \text{ at } z \overset{P,N}{\Longrightarrow} F' \text{ at } z'}{\Gamma, F;\Delta \overset{P,N}{\Longrightarrow} F' \text{ at } z'}\ \textit{COPY}$$

$$\frac{}{\Gamma;\Delta \overset{P,N}{\Longrightarrow} \top \text{ at } z}\ \top R$$

$$\frac{\Gamma; \Delta, F_1 \text{ at } z \stackrel{P,N}{\Longrightarrow} F_2 \text{ at } z}{\Gamma; \Delta \stackrel{P,N}{\Longrightarrow} F_1 \to F_2 \text{ at } z} \to R$$

$$\frac{\Gamma; \Delta \stackrel{P,N}{\Longrightarrow} F_1 \text{ at } z \quad \Gamma; \Delta, F_2 \text{ at } z \stackrel{P,N}{\Longrightarrow} F' \text{ at } z'}{\Gamma; \Delta, F_1 \to F_2 \text{ at } z \stackrel{P,N}{\Longrightarrow} F \text{ at } z'} \to L$$

$$\frac{\Gamma; \Delta \stackrel{P,N}{\Longrightarrow} F_1 \text{ at } z \quad \Gamma; \Delta \stackrel{P,N}{\Longrightarrow} F_2 \text{ at } z}{\Gamma; \Delta \stackrel{P,N}{\Longrightarrow} F_1 \wedge F_2 \text{ at } z} \wedge R$$

$$\frac{\Gamma; \Delta, F_1 \text{ at } z \stackrel{P,N}{\Longrightarrow} F \text{ at } z'}{\Gamma; \Delta, F_1 \wedge F_2 \text{ at } z \stackrel{P,N}{\Longrightarrow} F \text{ at } z'} \wedge L1$$

$$\frac{\Gamma; \Delta, F_2 \text{ at } z \stackrel{P,N}{\Longrightarrow} F \text{ at } z'}{\Gamma; \Delta, F_1 \wedge F_2 \text{ at } z \stackrel{P,N}{\Longrightarrow} F \text{ at } z'} \wedge L2$$

$$\frac{\Gamma; \Delta \stackrel{P+p,N}{\Longrightarrow} F \text{ at } z \quad p \notin \mathsf{FP}(\Gamma) \cup \mathsf{FP}(\Delta) \cup \mathsf{FP}(z)}{\Gamma; \Delta \stackrel{P,N}{\Longrightarrow} \forall p.F \text{ at } z} \forall R$$

$$\frac{\Gamma; \Delta, F\,[\,z_1 \,/\, p\,] \text{ at } z \stackrel{P,N}{\Longrightarrow} F' \text{ at } z'}{\Gamma; \Delta, \forall p.F \text{ at } z \stackrel{P,N}{\Longrightarrow} F' \text{ at } z'} \forall L$$

$$\frac{\Gamma; \Delta \stackrel{P,N}{\Longrightarrow} F \text{ at } z.n}{\Gamma; \Delta \stackrel{P,N}{\Longrightarrow} n[F] \text{ at } z} [\,] R$$

$$\frac{\Gamma; \Delta, F \text{ at } z.n \stackrel{P,N}{\Longrightarrow} F' \text{ at } z'}{\Gamma; \Delta, n[F] \text{ at } z \stackrel{P,N}{\Longrightarrow} F' \text{ at } z'} [\,] L$$

$$\frac{\Gamma; \Delta \stackrel{P,N}{\Longrightarrow} F \text{ at } z}{\Gamma; \Delta \stackrel{P,N}{\Longrightarrow} F \mathbin{@} z \text{ at } z'} @\, R$$

$$\frac{\Gamma; \Delta, F \text{ at } z \stackrel{P,N}{\Longrightarrow} F' \text{ at } z''}{\Gamma; \Delta, F \mathbin{@} z \text{ at } z' \stackrel{P,N}{\Longrightarrow} F' \text{ at } z''} @\, L$$

$$\frac{\Gamma; \Delta \stackrel{P+p,N}{\Longrightarrow} F \text{ at } p \quad p \notin \mathsf{FP}(\Gamma) \cup \mathsf{FP}(\Delta) \cup \mathsf{FP}(F)}{\Gamma; \Delta \stackrel{P,N}{\Longrightarrow} \Box F \text{ at } z} \Box R$$

$$\frac{\Gamma, F; \Delta \stackrel{P,N}{\Longrightarrow} F' \text{ at } z'}{\Gamma; \Delta, \Box F \text{ at } z \stackrel{P,N}{\Longrightarrow} F' \text{ at } z'} \Box L$$

$$\frac{\Gamma; \Delta \overset{P,N}{\Longrightarrow} F \text{ at } z}{\Gamma; \Delta \overset{P,N}{\Longrightarrow} \Diamond F \text{ at } z'} \Diamond R$$

$$\frac{\Gamma; \Delta, F \text{ at } p \overset{P+p;N}{\Longrightarrow} F' \text{ at } z' \quad p \notin \mathsf{FP}(F') \cup \mathsf{FP}(z')}{\Gamma; \Delta, \Diamond F \text{ at } z \overset{P,N}{\Longrightarrow} F' \text{ at } z'} \Diamond L$$

$$\frac{\Gamma; \Delta \overset{P,N}{\Longrightarrow} F \text{ at } z \quad \Gamma; \Delta, F \text{ at } z \overset{P,N}{\Longrightarrow} F' \text{ at } z'}{\Gamma; \Delta \overset{P,N}{\Longrightarrow} F' \text{ at } z'} \textit{ L-Cut}$$

$$\frac{\Gamma; \Delta \overset{P+q;N}{\Longrightarrow} F \text{ at } q \quad \Gamma, F; \Delta \overset{P,N}{\Longrightarrow} F' \text{ at } z'}{\Gamma; \Delta \overset{P,N}{\Longrightarrow} F' \text{ at } z'} \textit{ G-Cut}$$

$$\frac{\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} e : \tau \text{ at } z' \quad z' \equiv_E z}{\Gamma_\Box; \Gamma_\sharp; \Delta \vdash^{P,N,E} e : \tau \text{ at } z} \; Equiv$$

$$\frac{}{\Gamma_\Box; \cdot; \Delta, \ell : \tau \text{ at } z \vdash^{P,N,E} \ell : \tau \text{ at } z} \; LRT$$

$$\frac{}{\Gamma_\Box, \ell : \tau; \cdot; \Delta \vdash^{P,N,E} \mathbf{sync}\,(\ell) : \tau \text{ at } z} \; G_\Box RT$$

$$\frac{\Gamma_\Box; \cdot; \Delta \vdash^{P+p,N,E} e : \tau \text{ at } p}{\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \mathbf{run}\,(\,\lambda p.e\,[\,z\,]\,) : \tau \text{ at } z} \; G_\sharp RT$$

$$\frac{}{\Gamma_\Box; \cdot; \Delta, m : \tau \text{ at } z \vdash^{P,N,E} m : \tau \; \mathbf{ref} \text{ at } z} \; Lref$$

$$\frac{\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \ell : n[\tau] \text{ at } z}{\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \mathbf{sync}\,(\mathbf{rpc}^{\mathsf{rel}}(\ell, z)) : \tau \text{ at } z.n} \; [\,]RT$$

$$\frac{\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \ell : \tau \,@\, z \text{ at } z'}{\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \mathbf{sync}\,(\mathbf{rpc}^{\mathsf{abs}}(\ell, z')) : \tau \text{ at } z} \; @\,RT$$

$$\frac{\begin{array}{c}\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \ell : \Box\,\tau \text{ at } z \\ \Gamma_\Box, x : \tau; \cdot; \Delta \vdash^{P,N,E} e_2 : \tau' \text{ at } z'\end{array}}{\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \mathbf{sync}\,(\mathbf{bc}\,\ell \text{ at } z \;\mathbf{as}\, x \,\mathbf{in}\, e_2) : \tau' \text{ at } z'} \; \Box\,RT$$

$$\frac{\begin{array}{c}\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \ell : \sharp\tau \text{ at } z \\ \Gamma_\Box; x : \tau; \Delta \vdash^{P,N,E} e_2 : \tau' \text{ at } z'\end{array}}{\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \mathbf{sync}\,(\mathbf{pull}\,\ell \text{ at } z \;\mathbf{as}\, x \,\mathbf{in}\, e_2) : \tau' \text{ at } z'} \; \sharp\,RT$$

$$\frac{\begin{array}{c}\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \ell : \Diamond\tau \text{ at } z \\ \Gamma_\Box; \cdot; \Delta, x : \tau \text{ at } q \vdash^{P+q,N,E} e_2 : \tau' \text{ at } z'\end{array}}{\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \mathbf{sync}_1(\mathbf{go}\,\ell \text{ at } z \;\mathbf{return}\, x, q \,\mathbf{in}\, e_2) : \tau' \text{ at } z'} \; \Diamond RT1$$

$$\frac{\begin{array}{c}\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \ell : \tau \text{ at } z \\ \Gamma_\Box; \cdot; \Delta, x : \tau \text{ at } q \vdash^{P+q,N,E} e_2 : \tau' \text{ at } z'\end{array}}{\Gamma_\Box; \cdot; \Delta \vdash^{P,N,E} \mathbf{sync}_2(\mathbf{go}\,\ell \text{ at } z \;\mathbf{return}\, x, q \,\mathbf{in}\, e_2) : \tau' \text{ at } z'} \; \Diamond RT2$$

Figure 5: λ_{rpc} Runtime Typing Rules

$\boxed{\Delta_m \vdash^{P,N,E} \mathcal{L} : \Gamma_\square; \cdot; \Delta_i}$

$$\frac{}{\Delta_m \vdash^{P,N,E} \cdot : \cdot; \cdot; \cdot} \quad \text{(Empty-L)}$$

$$\frac{\Delta_m \vdash^{P,N,E} \mathcal{L} : \Gamma_\square; \cdot; \Delta_i \quad \Gamma_\square; \cdot; \Delta_i, \Delta_m \vdash^{P,N,E} e : \tau \text{ at } p}{\Delta_m \vdash^{P,N,E} \mathcal{L}, \ell \to e \text{ at } p : \Gamma_\square; \cdot; \Delta_i, \ell : \tau \text{ at } p} \quad \text{(Local-L)}$$

$$\frac{\Delta_m \vdash^{P,N,E} \mathcal{L}\backslash\ell : \Gamma_\square; \cdot; \Delta_i \quad \Gamma_\square; \cdot; \Delta_i, \Delta_m \vdash^{P,N,E} \mathcal{L}(p)(\ell) : \tau \text{ at } p \text{ for all } p \in P}{\Delta_m \vdash^{P,N,E} \mathcal{L} : \Gamma_\square, \ell : \tau; \cdot; \Delta_i} \quad \text{(Global-L)}$$

$\boxed{\Gamma_\square; \cdot; \Delta \vdash^{P,N,E} \mathcal{M} : \Delta_m}$

$$\frac{}{\Gamma_\square; \cdot; \Delta \vdash^{P,N,E} \cdot : \cdot} \quad \text{(Empty-M)}$$

$$\frac{\Gamma_\square; \cdot; \Delta \vdash^{P,N,E} \mathcal{M} : \Delta'_m \quad \Gamma_\square; \cdot; \Delta \vdash^{P,N,E} v : \tau \text{ at } p}{\Gamma_\square; \cdot; \Delta \vdash^{P,N,E} \mathcal{M}, m \to v \text{ at } p : \Delta'_m, m : \tau \text{ at } p} \quad \text{(Local-M)}$$

$\boxed{\vdash \mathcal{N} : \Gamma_\square; \cdot; \Delta; P; N}$

$$\frac{\Delta_m \vdash^{P,N,E} \mathcal{L} : \Gamma_\square; \cdot; \Delta_i \quad \Gamma_\square; \cdot; \Delta_i, \Delta_m \vdash^{P,N,E} \mathcal{M} : \Delta_m}{\vdash (P, N, E, \mathcal{L}, \mathcal{M}) : \Gamma_\square; \cdot; \Delta_i, \Delta_m; P; N} \quad \text{(Network)}$$

Figure 6: λ_{rpc} Network Typing

$\mathcal{L}, \mathcal{M} \longmapsto \mathcal{L}', \mathcal{M}'$	
sync RT	$\mathcal{L}, \ell' \to C[\,\mathbf{sync}\,(\ell)\,]$ at p, $\ell \to v$ at $p, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell' \to C[\,v\,]$ at p, $\ell \to v$ at $p, \mathcal{M}$
$run\,RT$	$\mathcal{L}, \ell \to C[\,\mathbf{run}\,(\,\lambda p.e\,[\,z\,]\,)\,]$ at $q, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,e\,[\,z\,/\,p\,]\,]$ at $q, \mathcal{M}$
$\to RT$	$\mathcal{L}, \ell \to C[\,(\lambda x{:}\tau.e)\,v\,]$ at $p, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,e\,[\,v\,/\,x\,]\,]$ at $p, \mathcal{M}$
$\wedge RT$	$\mathcal{L}, \ell \to C[\,\pi_i\langle\, v_1\,,\, v_2\,\rangle\,]$ at $p, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,v_i\,]$ at $p, \mathcal{M}$
@ $RT1$	$\mathcal{L}, \ell \to C[\,\mathbf{rpc}^{\mathsf{abs}}(e, z)\,]$ at $p, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,\mathbf{sync}\,(\mathbf{rpc}^{\mathsf{abs}}(\ell_1, z))\,]$ at p, $\ell_1 \to e$ at $p_1, \mathcal{M}$ where $E^*(z) = p_1$
@ $RT2$	$\mathcal{L}, \ell \to C[\,\mathbf{sync}\,(\mathbf{rpc}^{\mathsf{abs}}(\ell_1, z))\,]$ at p, $\ell_1 \to \mathbf{ret}^{\mathsf{abs}}(e, z_1)$ at $p_1, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,e\,]$ at p, $\ell_1 \to \mathbf{ret}^{\mathsf{abs}}(e, z_1)$ at $p_1, \mathcal{M}$ where $E^*(z) = p_1 \quad E^*(z_1) = p$
$[\,]\,RT1$	$\mathcal{L}, \ell \to C[\,\mathbf{rpc}^{\mathsf{rel}}(e, z)\,]$ at $p, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,\mathbf{sync}\,(\mathbf{rpc}^{\mathsf{rel}}(\ell_1, z))\,]$ at p, $\ell_1 \to e$ at $q, \mathcal{M}$ where $E^*(z) = q$
$[\,]\,RT2$	$\mathcal{L}, \ell \to C[\,\mathbf{sync}\,(\mathbf{rpc}^{\mathsf{rel}}(\ell_1, z))\,]$ at p, $\ell_1 \to \mathbf{ret}^{\mathsf{rel}}(e, n)$ at $q, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,e\,]$ at p, $\ell_1 \to \mathbf{ret}^{\mathsf{rel}}(e, n)$ at $q, \mathcal{M}$ where $E^*(z) = q$
$\Box\,RT1$	$\mathcal{L}, \ell \to C[\,\mathbf{bc}\,e_1$ at z **as** x **in** $e_2\,]$ at $p_0, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,\mathbf{sync}\,(\mathbf{bc}\,\ell_1$ at z **as** x **in** $e_2)\,]$ at p_0, $\ell_1 \to e_1$ at $p_1, \mathcal{M}$ where $E^*(z) = p_1$
$\Box\,RT2$	$\mathcal{L}, \ell \to C[\,\mathbf{sync}\,(\mathbf{bc}\,\ell_1$ at z **as** x **in** $e_2)\,]$ at p_0, $\ell_1 \to \mathbf{close}(\lambda p.\,e)$ at $p_1, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,e_2\,[\,\ell_2\,/\,x\,]\,]$ at p_0, $\ell_1 \to \mathbf{close}(\lambda p.\,e)$ at p_1, $\{\,\ell_2 \to e\,[\,q\,/\,p\,]$ at $q\,\}\ (\forall q \in P), \mathcal{M}$ where $E^*(z) = p_1$
$\Join\,RT1$	$\mathcal{L}, \ell \to C[\,\mathbf{pull}\,e_1$ at z **as** x **in** $e_2\,]$ at $p, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,\mathbf{sync}\,(\mathbf{pull}\,\ell_1$ at z **as** x **in** $e_2)\,]$ at p, $\ell_1 \to e_1$ at $p_1, \mathcal{M}$ where $E^*(z) = p_1$
$\Join\,RT2$	$\mathcal{L}, \ell \to C[\,\mathbf{sync}\,(\mathbf{pull}\,\ell_1$ at z **as** x **in** $e_2)\,]$ at p, $\ell_1 \to \mathbf{port}(\lambda p.\,e)$ at $p_1, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,e_2\,[\,\lambda p.e\,/\,x\,]\,]$ at p, $\ell_1 \to \mathbf{port}(\lambda p.\,e)$ at $p_1, \mathcal{M}$ where $E^*(z) = p_1$
$\Diamond\,RT1$	$\mathcal{L}, \ell \to C[\,\mathbf{go}\,e_1$ at z **return** x, q **in** $e_2\,]$ at $p_0, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,\mathbf{sync}_1(\mathbf{go}\,\ell_1$ at z **return** x, q **in** $e_2)\,]$ at p_0, $\ell_1 \to e_1$ at $p_1, \mathcal{M}$ where $E^*(z) = p_1$
$\Diamond\,RT2$	$\mathcal{L}, \ell \to C[\,\mathbf{sync}_1(\mathbf{go}\,\ell_1$ at z **return** x, q **in** $e_2)\,]$ at p_0, $\ell_1 \to \mathbf{agent}[e, z_1]$ at $p_1, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,\mathbf{sync}_2(\mathbf{go}\,\ell_2$ at z_1 **return** x, q **in** $e_2)\,]$ at p_0, $\ell_1 \to \mathbf{agent}[e, z_1]$ at p_1, $\ell_2 \to e$ at $p_2, \mathcal{M}$ where $E^*(z) = p_1$, $E^*(z_1) = p_2$
$\Diamond\,RT3$	$\mathcal{L}, \ell \to C[\,\mathbf{sync}_2(\mathbf{go}\,\ell_1$ at z **return** x, q **in** $e_2)\,]$ at p_0, $\ell_1 \to v$ at $p_1, \mathcal{M}$ $\longmapsto \ell \to C[\,e_2\,[\,z\,/\,q\,]\,[\,v\,/\,x\,]\,]$ at p_0, $\ell_1 \to v$ at $p_1, \mathcal{M}$ where $E^*(z) = p_1$
ref RT	$\mathcal{L}, \ell \to C[\,\mathbf{ref}\,v\,]$ at $p, \mathcal{M}$ $\longmapsto \mathcal{L}, \ell \to C[\,m\,]$ at $p, \mathcal{M}$, $m \to v$ at p $(m \notin \mathsf{Dom}(\mathcal{M}))$
$!\,RT$	$\mathcal{L}, \ell \to C[\,!\,m\,]$ at $p, \mathcal{M}, m \to v$ at p $\longmapsto \mathcal{L}, \ell \to C[\,v\,]$ at $p, \mathcal{M}$, $m \to v$ at p
$:=\ RT$	$\mathcal{L}, \ell \to C[\,m := v\,]$ at $p, \mathcal{M}$, $m \to v'$ at p $\longmapsto \mathcal{L}, \ell \to C[\,()\,]$ at $p, \mathcal{M}$, $m \to v$ at p

Figure 7: λ_{rpc} Operational Semantics