Distributed Control Flow with Classical Modal Logic*

Tom Murphy VII, Karl Crary, and Robert Harper

Carnegie Mellon University (tom7,crary,rwh@cs.cmu.edu)

Abstract. In previous work we presented a foundational calculus for spatially distributed computing based on intuitionistic modal logic. With the modalities $\Box$ and $\Diamond$ we were able to capture two key invariants: the mobility of portable code and the locality of fixed resources. This work investigates issues in distributed control flow through a similar propositions-as-types interpretation of *classical* modal logic. The resulting programming language is enhanced with the notion of a network-wide continuation, through which we can give computational interpretation of classical theorems (such as $\Box A \equiv \neg\Diamond\neg A$). Such continuations are also useful primitives for building higher-level constructs of distributed computing. The resulting system is elegant, logically faithful, and computationally reasonable.

1 Introduction

This paper is an exploration of distributed control flow using a propositions-as-types interpretation of classical modal logic. We build on our previous intuitionistic calculus, Lambda 5 [8], which is a simple programming language (and associated logic) for distributed computing. Lambda 5 focuses particularly on the spatial distribution of programs, and allows the programmer to express the *place* in which computation occurs using modal typing judgments. Through the modal operators $\Box$ and $\Diamond$ we are then able to express invariants about mobility and locality of resources. Our new calculus, C5, extends Lambda 5 with network-wide continuations, which arise naturally from the underlying classical logic. These continuations create a new relationship between the modalities $\Box$ and $\Diamond$, which we see with several examples, and serve as building blocks for other useful primitives. Before we introduce C5, we begin with a short reprise of Lambda 5.

Lambda 5. The Lambda 5 programming model is a network with many different *places*, or *nodes*. In order to be faithful to this model, we use a style of logic that has the ability to reason simultaneously from multiple perspectives, namely, modal logic. Compared to propositional logic, which is concerned with

* The ConCert Project is supported by the National Science Foundation under grant ITR/SY+SI 0121633: "Language Technology for Trustless Software Dissemination".

truth, modal logic deals with truth from the perspective of different *worlds*. These worlds are related by an *accessibility relation*, which affects the strength of the modal connectives; different assumptions about accessibility give rise to different modal logics. For modeling a network where the worlds are nodes, we choose Intuitionistic S5 [14], whose relation is reflexive, symmetric, and transitive—every world is related to every other world. Therefore, except when comparing it to other systems, we essentially dispense with the accessibility relation altogether. This leads to a simpler explanation of the judgments and connectives.

$A\,\mathsf{true}\,@\omega$ is the basic judgment, meaning that the proposition A is true at the world ω (we abbreviate this to $A@\omega$). There are two new proposition forms for quantifying over worlds. $\Box A$ is the statement that A is true at every world. $\Diamond A$ means that A is true at *some* world. Because we think of these worlds as places in the network, operationally we interpret type $\Box A$ as representing mobile code or data of type A, and the type $\Diamond A$ as an address of a value of type A.

Propositions must be situated at a world in order to be judged true, so it is important to distinguish between the proposition $\Box A$ and the judgment $\Box A@\omega$, the latter meaning that A is true in every world *from the perspective of* ω. In S5, every world has the same perspective with regard to statements about *all* or *some* world(s). But operationally this will be significant, as there is no true "global" code, only mobile code that currently exists at some world.

Though the logic distinguishes between $\Box A@\omega$ and $\Box A@\omega'$, both have precisely the same immediate consequences. The typical rule for eliminating $\Box$, for instance as given by Simpson [14] is

$$\frac{\Box A@\omega}{A@\omega'}\ \Box\mathrm{E}_{\ (\text{Simpson})}$$

With this rule, it *never really matters* where $\Box A$ exists, since we can eliminate it instantly to any world. However, we do care operationally where mobile code resides, and so we adjust the natural deduction rules to reflect this bias. The logic features a novel *decomposition* into locally-acting introduction and elimination rules as well as *motion* rules for moving between worlds, i.e.

$$\frac{\Box A@\omega}{A@\omega}\ \Box\mathrm{Elim} \qquad\qquad \frac{\Box A@\omega}{\Box A@\omega'}\ \Box\mathrm{Move}$$

We argue [8] that this results in a more appropriate operational interpretation. Our classical system also features this decomposition, and like Lambda 5, we are able to retain a crisp connection to the underlying logic.

Although distributed computing problems are often thought of as being concurrent, both Lambda 5 and our new calculus are sequential. We consider concurrency an orthogonal issue, although we give remarks on how it can be accomplished in Section 5.

Classical Control Flow. The notion that control operators such as Scheme's `call/cc` or Felleisen's $\mathcal{C}$ can be given logical meaning via classical logic is well known. Essentially, if we interpret the type $\neg A$ as a continuation expecting

a value of type A, then the types of these operators are classical tautologies. Griffin first proposed this in 1990 [4] with later refinements by (for example) Murthy [9]. Parigot's $\lambda\mu$-calculus [10] takes this idea and develops it into a full-fledged natural deduction system for classical logic.[1] It soon became clear that this was no accident—classical logic *is* the logic of control flow.

Therefore, a natural next step is to look at *classical* S5 to see what kind of programming language it gives us, which is the topic of this paper. We find that the notion of a network-wide continuation arises naturally, giving a computational explanation to (intuitionistically ridiculous) classical theorems such as $\Box A \equiv \neg\Diamond\neg A$. We also believe that such primitives are useful for building distributed computing mechanisms such as asynchronous message passing.

The paper proceeds as follows. We first present classical S5 judgmentally, giving a natural deduction system and intuition for its operational behavior. Next we give proof terms for some classical theorems, to elucidate the new connection between $\Box$ and $\Diamond$ made possible by network-wide continuations. In order to make these intuitions concrete, we then give an operational semantics based on an abstract network. We follow with some ideas about concurrency and how network-wide continuations can be used by distributed applications, and conclude with a discussion of related work. The appendix contains a proof that C5 really is classical S5 (along with establishing the existence of normal forms), by relating it to a sequent calculus that admits cut.

All of the proofs in this paper have been formalized in the Twelf system [11] and mechanically verified by its metatheorem checker [13].[2] Extended discussion of some of the proofs can be found in the accompanying technical report [7].

2 Classical S5

We wish to take a propositions-as-types interpretation of modal logic, so a judgmental proof theory for our logic is critical. In this section we give such a presentation of Classical S5.

Because modal logic is concerned with truth relativized to worlds, our judgments must reflect that. We have two main judgments in our proof theory.

$$A\ \mathsf{true} \mathbin{@} \omega \qquad\qquad A\ \mathsf{false} \star \omega$$

The first simply states that the proposition A is true at the world ω, as we had in Lambda 5. The second, which is new, says that the proposition A is *false* at the world ω. Although these two judgments are dual, the natural deduction system is deliberately biased towards deducing that propositions are true. We will only make assumptions about falsehood for the purpose of deriving a contradiction. As is standard, we reify the hypotheses about truth and falsehood into contexts (eliding true and false), and the central judgment of our proof theory becomes

$$\Gamma; \Delta \vdash A @ \omega$$

[1] Our calculus is quite similar to his (extended to the modal case!), although we prefer to present it with an emphasis on *truth* and *falsehood* judgments.

[2] They can be found at `http://www.cs.cmu.edu/~concert/`.

$$\frac{\Gamma, \omega'; \Delta \vdash M : A @ \omega'}{\Gamma; \Delta \vdash \mathtt{box}\ \omega'.M : \Box A @ \omega}\ \Box I \qquad \frac{\Gamma; \Delta \vdash M : \Box A @ \omega}{\Gamma; \Delta \vdash \mathtt{unbox}\, M : A @ \omega}\ \Box E$$

$$\frac{}{\Gamma, x{:}A @ \omega, \Gamma'; \Delta \vdash x : A @ \omega}\ \text{hyp} \qquad \frac{\Gamma; \Delta \vdash M : \Box A @ \omega' \quad \Gamma \vdash \omega'}{\Gamma; \Delta \vdash \mathtt{get}_{\Box}[\omega']M : \Box A @ \omega}\ \Box M$$

$$\frac{\Gamma; \Delta \vdash M : A @ \omega}{\Gamma; \Delta \vdash \mathtt{here}\, M : \Diamond A @ \omega}\ \Diamond I \qquad \frac{\Gamma; \Delta \vdash M : \Diamond A @ \omega' \quad \Gamma \vdash \omega'}{\Gamma; \Delta \vdash \mathtt{get}_{\Diamond}[\omega']M : \Diamond A @ \omega}\ \Diamond M$$

$$\frac{\Gamma; \Delta \vdash M : \Diamond A @ \omega \quad \Gamma, \omega', x{:}A @ \omega'; \Delta \vdash N : B @ \omega}{\Gamma; \Delta \vdash \mathtt{letd}\, \omega'.x = M\ \mathtt{in}\ N : B @ \omega}\ \Diamond E \qquad \frac{\Gamma; \Delta \vdash N : A @ \omega \quad \Gamma; \Delta \vdash M : A \supset B @ \omega}{\Gamma; \Delta \vdash MN : B @ \omega}\ \supset E$$

$$\frac{\Gamma, x{:}A @ \omega; \Delta \vdash M : B @ \omega}{\Gamma; \Delta \vdash \lambda x.M : A \supset B @ \omega}\ \supset I \qquad \frac{\Gamma; \Delta, u{:}A \star \omega \vdash M : A @ \omega}{\Gamma; \Delta \vdash \mathtt{letcc}\, u\, \mathtt{in}\, M : A @ \omega}\ \text{bc}$$

$$\frac{\Gamma; \Delta, u{:}A \star \omega, \Delta' \vdash M : A @ \omega}{\Gamma; \Delta, u{:}A \star \omega, \Delta' \vdash \mathtt{throw}\, M\, \mathtt{to}\, u : C @ \omega'}\ \# \qquad \frac{\Gamma; \Delta \vdash M : \bot @ \omega' \quad \Gamma \vdash \omega'}{\Gamma; \Delta \vdash \mathtt{go}[\omega']M : C @ \omega}\ \bot E$$

$$\frac{\Gamma; \Delta \vdash M : A @ \omega \quad \Gamma; \Delta \vdash N : B @ \omega}{\Gamma; \Delta \vdash \langle M, N \rangle : A \wedge B @ \omega}\ \wedge I \qquad \frac{\Gamma; \Delta \vdash M : A_1 \wedge A_2 @ \omega}{\Gamma; \Delta \vdash \pi_i M : A_i @ \omega}\ \wedge E_i$$

Fig. 1. Classical S5 natural deduction ("C5")

where we deduce that A is true at world ω under truth assumptions of the form $B @ \omega'$ appearing in Γ and falsehood assumptions of the form $C \star \omega''$ appearing in Δ. We also have hypotheses about the existence of worlds. It is cumbersome to write a separate context of world hypotheses, so these assumptions (written merely as ω) appear in Γ as well. We also take the common shortcut of only permitting mention of worlds that exist. Therefore, all judgments are hypothetical in at least *some* world (the world at which the conclusion is formed), until we introduce world constants in Section 4.

Operationally, we will think of a falsehood assumption $A \star \omega$ as a continuation, living at world ω, that expects something of type A.

Our natural deduction system appears in Fig. 1. These rules include proof terms, which we will explain shortly. Aside from the falsehood context, the rules for $\Box$, $\Diamond$ and $\supset$ are the same as in Lambda 5. The new connectives $\bot$ (discussed below) and $\wedge$ are treated as they would be in the intuitionistic case. The major additions are the structural rules *bc* (by contradiction) and # (contradict), which enable classical reasoning.

The *bc* rule is read as follows: In order to prove $A @ \omega$, we can assume that A is false at ω. This corresponds directly to the classical axiom $(\neg A \supset A) \supset A$. Operationally, this names the current continuation—we use a distinct class of "falsehood" or "continuation" variables u for this. The # rule may be alarming at first glance, because it requires the assumption $A \star \omega$ to appear in the conclusion. This is because the # rule is actually the *hypothesis* rule for false-

hood assumptions, and will have a corresponding substitution principle.[3] The rule simply states that if we have the assumption that A is false and are able to prove that A is true (at the same world), then we can deduce a contradiction and thus any proposition. The # rule is realized operationally as a `throw` of an expression (not a value, even though this is a call-by-value language) to a matching continuation. Note that continuations are *global*—we can throw from any world to a remote continuation $A \star \omega$, provided that we are able to construct a proof of $A @ \omega$.

The rules for $\Box$ and $\Diamond$ are key to the system. $\Box$ elimination is the easiest to understand: If we know that $\Box A$ is true at some world, then we know A is true at the same world. To prove $\Box A$, we must prove A at a hypothetical world about which nothing is known (rule $\Box I$). Operationally, we realize $\Box A$ as a piece of suspended code, with the hypothetical world ω' bound within it. Introduction of $\Diamond$ is simple; if we know A then we know that A is true *somewhere* (namely here). Operationally this will record the value in a table and return an address that witnesses its existence. Elimination of $\Diamond$ is as follows: if we know $\Diamond A$, then we know there is some world where A is true (but we don't know anything else about it). Call this world ω' and assume $A @ \omega'$ in order to continue reasoning. Finally, we provide *motion* rules (as per our decomposition) $\Box M$ and $\Diamond M$. Both simply allow knowledge of $\Box A$ or $\Diamond A$ at one world to be transported to another. Operationally these move the values between worlds.

Bottom has no introduction form, but we allow the *remote elimination* of it (rule $\bot E$). This is similar to the motion rules for $\Box$ and $\Diamond$, but is called `go` to indicate a transfer of control with no return.[4]

Despite the fact that our proof theory is specially constructed to give rise to a good operational semantics, it really embodies classical S5. To see this, we observe that it is equivalent to a symmetric multiple-conclusion sequent calculus that is more straightforwardly classical S5. The sequent calculus has the subformula property and admits (a dual form of) cut, which also establishes the existence of normal forms for our proof terms. The argument is mostly similar to the one used for our previous calculus, and is not the focus of this paper. Interested readers can find this material in the Appendix; otherwise, we'll begin to motivate the operational semantics of our calculus with some examples.

3 Examples

In this section we give proof terms showing the new connection between $\Box$ and $\Diamond$ made possible by network-wide continuations. A full operational semantics is forthcoming in Section 4.1, but let us review our informal interpretation of the modal connectives now.

[3] A theory of *hypothetical hypotheticals* would be able to express this in a less awkward—but perhaps no less alarming—way. Abel [1] for instance gives such a third-order encoding of the $\lambda\mu$-calculus.

[4] We could have equivalently had a $\texttt{get}_{\bot}$ and a local `abort`, but there appears to be no practical use to this decomposition.

A value of type $\Box A$ is a suspended expression that makes sense anywhere. We call such values *boxes*, and we can open them at any world using the `unbox` primitive, which begins evaluating the expression. A value of type $\Diamond A$ is an address of a value that has been published in a table at some world. In order to make addresses, we use the `here` construct to publish a value in the local table and generate a new address for it. We have the ability to travel and move certain data between worlds by using the `get` and `go` constructs.

Finally, because our examples involve negation ($\neg A$), we first briefly explain how we treat it.

Negation. Although we have not given the rules for the negation connective, it is easily added to the system. Here we take the standard shortcut of treating $\neg A$ as an abbreviation for $A \supset \bot$. We computationally read $\neg A @ \omega$ as a continuation expecting A, although this should be distinguished from a primitive continuation assumption $u{:}A \star \omega$: the former is introduced by lambda abstraction and eliminated by application, while the latter is formed with `letcc` and eliminated by a `throw` to it. The two are related in that we can reify a falsehood assumption $u{:}A \star \omega$ as a negated formula $\neg A @ \omega$ by forming a function that throws to it: $\lambda a.\, \texttt{throw}\, a\, \texttt{to}\, u$. Likewise, we can create a falsehood assumption from a term $M : \neg A @ \omega$, namely $M(\texttt{letcc}\, u\, \texttt{in} \ldots)$.

Classical Axioms. As examples, we give proof terms for several classical axioms. To implement one of these axioms, the programmer engages in a little theorem proving puzzle. Because we are dealing with classical logic, we have two sorts of resources in solving the puzzle: *values* of type A, as in intuitionistic logic, but also *contexts expecting* terms of type A. We can capture such contexts with `letcc`, so sometimes we go out of our way to create them; thus the the *need for* a value of some type can be as useful as the *presence of* one.

Our first example comes from the standard practice in classical modal logic of defining $\Box$ in terms of $\Diamond$ through the equivalence $\Box A \equiv \neg\Diamond\neg A$. From left to right the implication is intuitionistically valid, so we'll look at the proof of the implication right to left. In C5, the proof term tells an interesting story:

$$\begin{array}{ll}
\lambda d.\, \texttt{box}\ \omega'. & (d : (\Diamond\neg A) \supset \bot @ \omega; \text{need to return } A @ \omega') \\
\quad \texttt{letcc}\, u\, \texttt{in}\, \texttt{go}[\omega] & (\text{applying } d \text{ will yield } \bot) \\
\quad\quad d(\texttt{get}_\Diamond[\omega'](\texttt{here}(\lambda a.\, \texttt{throw}\, a\, \texttt{to}\, u)))
\end{array}$$

In each example, we'll assume that the whole term lives at the world ω. Operationally, the reading of $\neg\Diamond\neg A \supset \Box A$ is that given a continuation d (expecting the address of an A continuation), we will return a boxed A that is well-formed anywhere. It is easiest to understand this term from the perspective of the consumer of the resulting $\Box A$. When it is unboxed at some world ω', it grabs the current continuation u, which expects an A. It then publishes this continuation (reified as a function); the address is what we require as an argument for d. (What happens next depends on what d does with its argument!) The intervening `go` and $\texttt{get}_\Diamond$ accomplish the transfer of control between the two worlds.

Dually we can define $\Diamond$ in terms of $\Box$. Again, one direction is intuitionistically valid. The other, $\neg\Box\neg A \supset \Diamond A$, is asked to conjure up an address of an arbitrary A given a continuation (that expects a boxed A continuation). It is implemented by the following proof term:

$$\begin{array}{ll} \lambda b.\, \texttt{letcc}\, u\, \texttt{in} & (b : (\Box\neg A) \supset \bot @ \omega; u : \Diamond A \star \omega) \\ \quad \texttt{go}[\omega]\ b(\texttt{box}\ \omega'.\lambda a. & (a : A @ \omega') \\ \qquad \texttt{throw}(\texttt{get}_\Diamond[\omega'](\texttt{here}\, a))\, \texttt{to}\, u) & \end{array}$$

Here, we immediately grab the $\Diamond A$ continuation with `letcc`. Since we will be calling b (proving $\bot$ and never returning), we "`go`" to the current world. We then form a `box` to pass to the function b. It contains a function of type $A \supset \bot$, which takes the address of its argument and throws it to the saved continuation u. Thus the location of A that we ultimately return is any world that calls the $\neg A$ function that we've boxed up.

Excluded "Modal." The following example uses disjunction, which we've left out of our calculus so far. A description of some ways it can be added is given in Section 6, but for now we will be somewhat less formal and simply assume that we have constructors `inl` and `inr` for forming proofs of $A \vee B$.

Our example is a modal version of the excluded middle axiom: $\Box A \vee \Diamond\neg A$. We will again return a box that does something when opened.

$$\begin{array}{ll} \texttt{letcc}\, u_o\, \texttt{in} & (u_o : \Box A \vee \Diamond\neg A \star \omega) \\ \quad \texttt{inl}(\texttt{box}\ \omega'.\, \texttt{letcc}\, u\, \texttt{in} & (u : A \star \omega') \\ \qquad \texttt{throw}(\texttt{inr}(\texttt{get}_\Diamond[\omega']\, \texttt{here}(\lambda a.\, \texttt{throw}\, a\, \texttt{to}\, u))) & \\ \qquad \texttt{to}\, u_o) & \end{array}$$

First, we save the current continuation as u_o, since we will need to "change our minds" and return multiple different disjuncts. When asked for $\Box A \vee \Diamond\neg A$, the program initially says $\Box A$.If the box is opened, the program uses context expecting an A to produce a $\Diamond\neg A$, *time travels* back to when it was asked about the disjunction, and returns this different answer.If that $\neg A$ continuation is ever invoked, the program goes back and uses the A to fulfill the outstanding request for an A at the world where the box was opened.

In the style of sci-fi storytelling popular when describing such things, we conclude our examples with the following fable (with apologies to Wadler [15]):

A magician who purports to be from the future is making bold claims. Asking for a volunteer, he offers the following prize to anyone who comes on stage:

"I'm going to hand you a box that has *you* inside it! Either that, or I'll give you the address of a place with a magical time travelling portal."

Being questionably brave, you volunteer and walk onto the stage. The magician hands you your prize—a large cardboard box. Noting your skepticism, he adds, "You can open it anywhere, and you'll be inside."

You decide to take the box home. It's much too light to have anything in it, let alone yourself! You open the box and look inside, wondering what sort

$$
\begin{array}{llll}
\text{world vars} & \omega & \text{world names} & \mathbf{w} \qquad \text{labels} \quad \ell \\
\text{value vars} & x, y & \text{cont labs} & \mathbf{k} \qquad \text{cont vars} \quad u \\
\text{types} & A, B ::= A \supset B \mid \Box A \mid \Diamond A \mid A \wedge B \mid \bot & & \\
\text{networks} & \mathbb{N} ::= \mathbb{W}; R & \text{world exps} & \mathrm{w} ::= \omega \mid \mathbf{w} \\
\text{configs} & \mathbb{W} ::= \{\mathbf{w}_1 : \langle \chi_1, b_1 \rangle, \cdots\} & & \\
\text{cursors} & R ::= \mathbf{w} : [k \prec v] \mid \mathbf{w} : [k \succ M] & & \\
\text{tables} & b ::= \bullet \mid b, \ell = v & \text{cont tables} & \chi ::= \bullet \mid \chi, \mathbf{k} = k \\
\text{config types} & \Sigma ::= \{\mathbf{w}_1 : \langle X_1, \beta_1 \rangle, \cdots\} & & \\
\text{table types} & \beta ::= \bullet \mid \beta, \ell : A & \text{ctable types} & X ::= \bullet \mid X, \mathbf{k} : A \\
\text{cont exps} & Z ::= \mathbf{w}.\mathbf{k} \mid u & & \\
\text{conts} & k ::= \mathtt{return}\, Z \mid \mathtt{finish} \mid \mathtt{abort} \mid k \lhd f & & \\
\text{values} & v ::= \lambda x.M \mid \mathtt{box}\, \omega.M \mid \mathbf{w}.\ell \mid \langle v, v' \rangle & & \\
\text{frames} & f ::= \circ\, N \mid v\ \circ \mid \mathtt{here} \circ \mid \mathtt{unbox} \circ & & \\
 & \quad \mid \mathtt{letd}\, \omega.x = \circ\, \mathtt{in}\, N \mid \pi_n \circ \mid \langle \circ, N \rangle \mid \langle v, \circ \rangle & & \\
\text{exps} & M, N ::= v \mid MN \mid x \mid \ell \mid \mathtt{get}_{\Box}[\mathrm{w}]M \mid \mathtt{here}\, M \mid \mathtt{get}_{\Diamond}[\mathrm{w}]M & & \\
 & \quad \mid \mathtt{unbox}\, M \mid \mathtt{letd}\, \omega.x = M\, \mathtt{in}\, N \mid \mathtt{throw}\, M\, \mathtt{to}\, Z & & \\
 & \quad \mid \mathtt{go}[\mathrm{w}]M \mid \mathtt{letcc}\, u\, \mathtt{in}\, M \mid \langle M, N \rangle \mid \pi_n M & &
\end{array}
$$

Fig. 2. Syntax of type system

of gag he has planned. But suddenly you find that the box has vanished, and you're standing on stage waiting for him to tell you what you've won, again.

"The address of the time-travelling portal is," he begins, rattling off your home address. You are startled that he could have known your address, but when you later arrive home, you see an open cardboard box waiting. Is this supposed to be the portal? Knowing it to be harmless, but insisting on proving the magician to be a fraud, you step into it.

A hot flash of embarrassment passes over you as you realize that you are now standing in a cardboard box, in your house, as promised.

4 Type System and Operational Semantics

Our deductive proof theory corresponds to a natural programming language whose syntax is the proof terms from Fig. 1. In order to give this language an operational interpretation, we need to introduce a number of syntactic constructs, which are given in Fig. 2.

As in Lambda 5, the behavior of a program is specified in terms of an abstract network that steps from state to state. The network is built out of a fixed number of worlds, whose names we write as bold $\mathbf{w}$. Because we can now mention specific worlds in addition to hypothetical worlds ω, we introduce world expressions, which are written with a Roman w. A network state $\mathbb{N}$ has two parts. First is a world configuration $\mathbb{W}$ which identifies two tables with each world $\mathbf{w}_i$ present. The first table χ_i stores network-wide continuations by mapping continuation labels $\mathbf{k}$ to literal continuations k. The second table b_i maps value labels ℓ to values in order to store values whose address we have published. These tables

$\Sigma; \Gamma; \Delta \vdash M : A @ \mathrm{w}$	The expression M has type A at world w
$\Sigma \vdash k : A \star \mathrm{w}$	The continuation k expects a value of type A at world w
$\Sigma; \Delta \vdash Z : A \star \mathrm{w}$	The continuation expression Z is well-formed with type A at w
$\Sigma \vdash b @ \mathbf{w}$	The value table b is well-formed at the world named $\mathbf{w}$
$\Sigma \vdash \chi \star \mathbf{w}$	The continuation table χ is well-formed at the world named $\mathbf{w}$
$\Sigma \vdash R$	The cursor is well-formed
$\Sigma \vdash \mathbb{N}$	The network is well-formed

Fig. 3. Index of judgments. In each judgment Σ is a configuration typing, Γ is a context of truth hypotheses, and Δ is a context of falsehood hypotheses

have types X and β respectively (which map labels $\mathbf{k}$ and ℓ to types), and so we can likewise construct the type of an entire configuration, written Σ.

Aside from the current world configuration, a network state also contains a *cursor* denoting the current focus of computation. The cursor either takes the form $\mathbf{w} : [k \prec v]$ (returning the value v to the continuation k) or $\mathbf{w} : [k \succ M]$ (evaluating the expression M in continuation k). In either case it selects a world $\mathbf{w}$ where the computation is taking place.

Continuations themselves are stacks of frames (expressions with a "hole," written $\circ$) with a bottommost `return`, `finish` or `abort`. The `finish` continuation represents the end of computation, so a network state whose cursor is returning a value to `finish` is called *terminal*. The `abort` continuation will be unreachable, and `return` will send the received value to a remote continuation.

Most of the expressions and values are straightforward. As in Lambda 5, the canonical value for $\Box$ abstracts over the hypothetical world and leaves its body unevaluated (`box` $\omega'.M$). The canonical form for $\Diamond$ is a pair of a world name and a label $\mathbf{w}.\ell$, which addresses a table entry at that world. Such an address is well-formed anywhere (assuming that $\mathbf{w}$'s table has a label ℓ containing a value of type A) and has type $\Diamond A @ \mathrm{w}'$. On the other hand we have another sort of label, written just ℓ, which is disembodied from its world. These labels arise from the `letd` construct, which deconstructs an address $\mathbf{w}.\ell$ into its components $\mathbf{w}$ and ℓ (see the $\Diamond E$ rule from Fig. 1). Disembodied labels only make sense at a single world—here ℓ would have type $A @ \mathbf{w}$.

Although the external language only allows a `throw` to a continuation variable, intermediate states of evaluation require that these be replaced with the continuation value $\mathbf{w}.\mathbf{k}$, which pairs a continuation label with the world at which it lives. These continuation values are filled in by `letcc`.

The type system is given in Fig. 4 (we omit for space the rules that are the same as in Fig. 1 except for the configuration typing Σ). The index of judgments in Fig. 3 may be a useful reference in understanding them.

The rules *addr* and *lab* are used to type run-time artifacts of address publishing. In either case, we look up the type in the appropriate table typing β. As mentioned, *throw* allows a continuation expression Z, which is either a variable (typed with $hyp^\star$, as in the logic) or an address into a continuation table.

Typing of literal continuations k is fairly unsurprising. Note that the judgment $\Sigma \vdash k : A \star \mathrm{w}$ means that the continuation k *expects* a value of type A at

$$\frac{\Sigma(\mathbf{w}) = \langle X, \beta\rangle \quad \beta(\ell) = A}{\Sigma; \Gamma; \Delta \vdash \mathbf{w}.\ell : \Diamond A @ \mathrm{w}'}\ \text{addr} \qquad \frac{\Sigma(\mathbf{w}) = \langle X, \beta\rangle \quad \beta(\ell) = A}{\Sigma; \Gamma; \Delta \vdash \ell : A @ \mathbf{w}}\ \text{lab}$$

$$\frac{\Sigma; \Gamma; \Delta \vdash M : A @ \mathrm{w} \quad \Sigma; \Delta \vdash Z : A \star \mathrm{w}}{\Sigma; \Gamma; \Delta \vdash \mathtt{throw}\, M\, \mathtt{to}\, Z : C @ \mathrm{w}'}\ \text{throw} \qquad \frac{\Sigma; \Gamma; \Delta, u : A \star \mathrm{w} \vdash M : A @ \mathrm{w}}{\Sigma; \Gamma; \Delta \vdash \mathtt{letcc}\, u\, \mathtt{in}\, M : A @ \mathrm{w}}\ \text{letcc}$$

$$\frac{\Sigma(\mathbf{w}) = \langle X, \beta\rangle \quad X(\mathbf{k}) = A}{\Sigma; \Delta \vdash \mathbf{w}.\mathbf{k} : A \star \mathbf{w}}\ \text{addr}^\star \qquad \frac{}{\Sigma; \Delta, u : A \star \mathrm{w} \vdash u : A \star \mathrm{w}}\ \text{hyp}^\star$$

$$\frac{\Sigma \vdash k : B \star \mathrm{w} \quad \Sigma; \cdot; \cdot \vdash N : A @ \mathrm{w}}{\Sigma \vdash k \lhd \circ\ N : A \supset B \star \mathrm{w}}\ \text{kapp}_1 \qquad \frac{}{\Sigma \vdash \mathtt{finish} : A \star \mathrm{w}}\ \text{kfinish}$$

$$\frac{\Sigma \vdash k : B \star \mathrm{w} \quad \Sigma; \cdot; \cdot \vdash v : A \supset B @ \mathrm{w}}{\Sigma \vdash k \lhd v\ \circ : A \star \mathrm{w}}\ \text{kapp}_2 \qquad \frac{}{\Sigma \vdash \mathtt{abort} : \bot \star \mathrm{w}}\ \text{kabort}$$

$$\frac{\begin{array}{c}\Sigma \vdash k : C \star \mathrm{w} \\ \Sigma; \omega, x : A @ \omega; \cdot \vdash N : C @ \mathrm{w}\end{array}}{\Sigma \vdash k \lhd \mathtt{letd}\, \omega.x = \circ\, \mathtt{in}\, N : \Diamond A \star \mathrm{w}}\ \text{kletd} \qquad \frac{\Sigma \vdash k : \Diamond A \star \mathrm{w}}{\Sigma \vdash k \lhd \mathtt{here}\, \circ : A \star \mathrm{w}}\ \text{khere}$$

$$\frac{\Sigma \vdash k : A \wedge B \star \mathrm{w} \quad \Sigma; \cdot; \cdot \vdash N : B @ \mathrm{w}}{\Sigma \vdash k \lhd \langle \circ, N\rangle : A \star \mathrm{w}}\ \text{k}\wedge_1 \qquad \frac{\Sigma \vdash k : A \star \mathrm{w}}{\Sigma \vdash k \lhd \mathtt{unbox}\, \circ : \Box A \star \mathrm{w}}\ \text{kunbox}$$

$$\frac{\Sigma \vdash k : A \wedge B \star \mathrm{w} \quad \Sigma; \cdot; \cdot \vdash v : A @ \mathrm{w}}{\Sigma \vdash k \lhd \langle v, \circ\rangle : B \star \mathrm{w}}\ \text{k}\wedge_2 \qquad \frac{A = \Box A' \text{ or } \Diamond A' \quad \Sigma; \cdot \vdash Z : A \star \mathrm{w}'}{\Sigma \vdash \mathtt{return}\, Z : A \star \mathrm{w}}\ \text{kret}$$

$$\frac{\beta = (\ell_1 : A_1, \ldots) \quad \Sigma; \cdot; \cdot \vdash v_1 : A_1 @ \mathbf{w} \quad \ldots}{\underbrace{\{\cdots, \mathbf{w} : \langle X, \beta\rangle, \cdots\}}_{\Sigma} \vdash \underbrace{\ell_1 = v_1, \ldots}_{b} @ \mathbf{w}}\ b \qquad \frac{\mathbf{w} \in \mathrm{dom}(\Sigma) \quad \Sigma; \cdot; \cdot \vdash v : A @ \mathbf{w}\ \Sigma \vdash k : A \star \mathbf{w}}{\Sigma \vdash \mathbf{w} : [k \prec v]}\ \text{ret}$$

$$\frac{X = (\mathbf{k}_1 : A_1, \ldots) \quad \Sigma \vdash k_1 : A_1 \star \mathbf{w} \quad \ldots}{\underbrace{\{\cdots, \mathbf{w} : \langle X, \beta\rangle, \cdots\}}_{\Sigma} \vdash \underbrace{\mathbf{k}_1 = k_1, \ldots}_{\chi} \star \mathbf{w}}\ \chi \qquad \frac{\mathbf{w} \in \mathrm{dom}(\Sigma) \quad \Sigma; \cdot; \cdot \vdash M : A @ \mathbf{w}\ \Sigma \vdash k : A \star \mathbf{w}}{\Sigma \vdash \mathbf{w} : [k \succ M]}\ \text{eval}$$

$$\frac{\Sigma \vdash R \quad \Sigma \vdash \chi_i @ \mathbf{w}_i \ \ldots \quad \Sigma \vdash b_i @ \mathbf{w}_i \ \ldots}{\Sigma \vdash \{\mathbf{w}_1 : \langle \chi_1, b_1\rangle, \cdots, \mathbf{w}_m : \langle \chi_m, b_m\rangle\}; R}\ \text{net}$$

Fig. 4. Type system

w. The `return` continuation arises only from a $\mathtt{get}_\Diamond$ or $\mathtt{get}_\Box$, and so it allows only values of type $\Diamond A$ or $\Box A$. We use the network continuation mechanism to name the the outstanding $\mathtt{get}_\Diamond$ or $\mathtt{get}_\Box$ request on the remote machine.

For an entire network to be well-formed (rule *net*), all of the tables must have the type indicated by the configuration type Σ, which means that they must have exactly the same labels, and the values or continuations must be well-typed at the specified types (rules b and χ). Finally, the cursor must be well-formed: it must select a world that exists in the network, and there must exist a type A such that its continuation and value or expression both have type A and are closed.

Having set up the syntax and type system, we can now give the operational semantics and type safety theorem. After the following section we remark on how

$$
\begin{array}{lll}
\supset_e\text{-p} & \mathbb{W}; \mathbf{w} : [k \succ MN] & \mapsto \mathbb{W}; \mathbf{w} : [k \lhd \circ N \succ M] \\
\supset_e\text{-s} & \mathbb{W}; \mathbf{w} : [k \lhd \circ N \prec v] & \mapsto \mathbb{W}; \mathbf{w} : [k \lhd v \circ \succ N] \\
\supset_e\text{-r} & \mathbb{W}; \mathbf{w} : [k \lhd (\lambda x.M) \circ \prec v] & \mapsto \mathbb{W}; \mathbf{w} : [k \succ [v/x]M] \\
\text{value} & \mathbb{W}; \mathbf{w} : [k \succ v] & \mapsto \mathbb{W}; \mathbf{w} : [k \prec v] \\
\Diamond_i\text{-p} & \mathbb{W}; \mathbf{w} : [k \succ \mathtt{here}\, M] & \mapsto \mathbb{W}; \mathbf{w} : [k \lhd \mathtt{here} \circ \succ M] \\
\Diamond_i\text{-r} & \{\mathbf{w} : \langle \chi, b \rangle, \cdots\}; \mathbf{w} : [k \lhd \mathtt{here} \circ \prec v] & \mapsto \\
 & \{\mathbf{w} : \langle \chi, (b, \ell = v) \rangle, \cdots\}; \mathbf{w} : [k \prec \mathbf{w}.\ell] & (\ell \text{ fresh}) \\
\ell\text{-r} & \{\mathbf{w} : \langle \chi, b \rangle, \cdots\}; \mathbf{w} : [k \succ \ell] & \mapsto \\
 & \{\mathbf{w} : \langle \chi, b \rangle, \cdots\}; \mathbf{w} : [k \prec v] & (b(\ell) = v) \\
\Diamond_e\text{-p} & \mathbb{W}; \mathbf{w} : [k \succ \mathtt{letd}\, \omega.x = M\, \mathtt{in}\, N] & \mapsto \mathbb{W}; \mathbf{w} : [k \lhd \mathtt{letd}\, \omega.x = \circ\, \mathtt{in}\, N \succ M] \\
\Diamond_e\text{-r} & \mathbb{W}; \mathbf{w} : [k \lhd \mathtt{letd}\, \omega.x = \circ\, \mathtt{in}\, N \prec \mathbf{w}'.\ell] & \mapsto \mathbb{W}; \mathbf{w} : [k \succ [\ell/x][\mathbf{w}'/\omega]N] \\
\Box_e\text{-p} & \mathbb{W}; \mathbf{w} : [k \succ \mathtt{unbox}\, M] & \mapsto \mathbb{W}; \mathbf{w} : [k \lhd \mathtt{unbox} \circ \succ M] \\
\Box_e\text{-r} & \mathbb{W}; \mathbf{w} : [k \lhd \mathtt{unbox} \circ \prec \mathtt{box}\, \omega.M] & \mapsto \mathbb{W}; \mathbf{w} : [k \succ [\mathbf{w}/\omega]M] \\
\text{letcc} & \{\mathbf{w} : \langle \chi, b \rangle, \cdots\}; \mathbf{w} : [k \succ \mathtt{letcc}\, u\, \mathtt{in}\, M] & \mapsto \\
 & \{\mathbf{w} : \langle (\chi, \mathbf{k} = k), b \rangle, \cdots\}; \mathbf{w} : [k \succ [\mathbf{w}.\mathbf{k}/u]M] & (\mathbf{k} \text{ fresh}) \\
\text{throw} & \{\mathbf{w}' : \langle \chi, b \rangle, \cdots\}; \mathbf{w} : [k \succ \mathtt{throw}\, M\, \mathtt{to}\, \mathbf{w}'.\mathbf{k}] & \mapsto \\
 & \{\mathbf{w}' : \langle \chi, b \rangle, \cdots\}; \mathbf{w}' : [k' \succ M] & (\chi(\mathbf{k}) = k') \\
\text{rpc} & \mathbb{W}; \mathbf{w} : [k \succ \mathtt{go}[\mathbf{w}']M] & \mapsto \\
 & \mathbb{W}; \mathbf{w}' : [\mathtt{abort} \succ M] & (\mathbf{w} \in \mathrm{dom}(\mathbb{W})) \\
\Box_m & \{\mathbf{w} : \langle \chi, b \rangle, \cdots\}; \mathbf{w} : [k \succ \mathtt{get}_\Diamond[\mathbf{w}']M] & \mapsto \\
 & \{\mathbf{w} : \langle (\chi, \mathbf{k} = k), b \rangle, \cdots\}; \mathbf{w}' : [\mathtt{return}\, \mathbf{w}.\mathbf{k} \succ M] & (\mathbf{k} \text{ fresh}) \\
\Diamond_m & \{\mathbf{w} : \langle \chi, b \rangle, \cdots\}; \mathbf{w} : [k \succ \mathtt{get}_\Box[\mathbf{w}']M] & \mapsto \\
 & \{\mathbf{w} : \langle (\chi, \mathbf{k} = k), b \rangle, \cdots\}; \mathbf{w}' : [\mathtt{return}\, \mathbf{w}.\mathbf{k} \succ M] & (\mathbf{k} \text{ fresh}) \\
\text{ret} & \{\mathbf{w} : \langle \chi, b \rangle, \cdots\}; \mathbf{w}' : [\mathtt{return}\, \mathbf{w}.\mathbf{k} \prec v] & \mapsto \\
 & \{\mathbf{w} : \langle \chi, b \rangle, \cdots\}; \mathbf{w} : [k \prec v] & (\chi(\mathbf{k}) = k)
\end{array}
$$

Fig. 5. Selected rules from the operational semantics

the semantics can be made concurrent, and give some thoughts on applications of distributed continuations.

4.1 Operational Semantics

The operational semantics of our language is given in Fig. 5, as a binary relation $\mapsto$ between network states. The semantics evaluates programs sequentially, though we give a concurrent semantics in Section 5.

Not surprisingly, the semantics is continuation-based. At any step, the cursor is selecting a world and continuation, with a value to return to it or an expression to evaluate. The rules generally fall into a few categories, as exemplified by the (standard) rules for $\supset$: There are (p)ush rules, in which we begin evaluating a subexpression of some M, pushing the context into the continuation, (s)wap rules, where we have finished evaluating one sub-expression and move onto the next, and (r)eduction rules, where we finally have a value and eliminate it. Every well-typed machine state will be closed with respect to truth, falsehood, and world hypotheses, so we don't have rules for variables.

The first interesting rule is $\Diamond_i$-r. It publishes the value v by generating a new label ℓ, mapping that label to v within its value table, and returning the pair $\mathbf{w}.\ell$, where $\mathbf{w}$ is the current world. Whenever we try to evaluate a label (rule ℓ-r), we look it up in the current world's value table in order to find the value. A key consequence of type safety (Theorems 1, 2) is that labels are only evaluated in the correct world. To eliminate an address (rule $\Diamond_e$-r) we substitute the constituent world and label through the body of the `letd`. Note that this step is slightly non-standard, because we substitute the *expression* ℓ for a variable rather than some value. But because the variable is in general at a different world, we are not in a position to get its value yet. We instead wait until the expression ℓ is sent to its home world (perhaps as part of some larger expression) to be looked up. The rules for $\Box$ are much simpler: `box` $\omega.M$ is already a value (rule $\Box_i$-v), and to `unbox` we simply substitute the current world for the hypothetical one (rule $\Box_e$-r).

When encountering a `letcc`, we grab the current continuation k. Because the continuation may be referred to from elsewhere in the network, we publish it in a table and form a global address for it (of the form $\mathbf{w}.\mathbf{k}$), just as we did for $\Diamond$ addresses. This value is substituted for the falsehood variable u.

Throwing to a continuation (rule *throw*) is handled straightforwardly. The continuation expression will be closed, and therefore of the form $\mathbf{w}'.\mathbf{k}$. We look up the label $\mathbf{k}$ in $\mathbf{w}'$—or rather, *cause* $\mathbf{w}'$ to look it up—and pass the expression M to it. Note that we do not evaluate the argument before throwing it to the remote continuation. In general we *can not* evaluate it, because it is only well-typed at the remote world, which may be different from the world we're in.

Finally, we have the rules that move between worlds. The rule for `go` is easiest; since the target world expression must be closed it will be a world constant in the domain of $\mathbb{W}$. We simply move the cursor to that world (destroying the current continuation, which can never be reached), and begin evaluating the expression M under the unreachable continuation `abort`. The rules for $\texttt{get}_\Diamond$ and $\texttt{get}_\Box$ work similarly, but they need to save the current continuation since they will be returned to! These steps push a `return` frame, which reduces like `throw`. In contrast, however, the argument (of type $\Box A$ or $\Diamond A$) will be eagerly evaluated, because such values are portable. (After all, the whole point is to create the box at one world and then move it to another.)

In order for our language to make sense it must be type safe; any well-typed program must have a well-defined meaning as a sequence of steps in the abstract network. Type safety is stated as usual in terms of progress and preservation:

Theorem 1 (Progress)
If $\Sigma \vdash \mathbb{N}$ *then either* $\mathbb{N}$ *is terminal or* $\exists \mathbb{N}'.\mathbb{N} \mapsto \mathbb{N}'$.

Theorem 2 (Preservation)
If $\Sigma \vdash \mathbb{N}$ *and* $\mathbb{N} \mapsto \mathbb{N}'$ *then* $\exists \Sigma'.\ \Sigma' \supseteq \Sigma$ *and* $\Sigma' \vdash \mathbb{N}'$.

Progress says that any well-formed network state can take another step, or is done. (Recall a *terminal* network is one where the cursor is returning a value

to a `finish` continuation.) Preservation says that any well-typed network state that takes a step results in another well-typed state (perhaps in an extended[5] configuration typing Σ'). By iterating alternate applications of these theorems we see that any well-typed program is able to step repeatedly and remain well-formed, or else eventually comes to rest in a terminal state.

5 Concurrency and Communication

Many distributed computing problems benefit from concurrency, with one or more processes running on each node in the network. This section gives some brief thoughts on concurrency in our classical calculus.

First-class continuations are often used in the implementation of coroutines. With primitives for recursion and state we could also implement coroutines in C5, however, such an implementation is silly because it would require the implementation of a global scheduler, and would anyway defeat the purpose of concurrency on multiple nodes—only one coroutine would be running at any given time!

Fortunately, our operational semantics admits ad hoc concurrency easily. If we simply replace the cursor R in our network state "$\mathbb{W}; R$" with a multiset of cursors $\Re$, then we can permit a step on any one of these cursors essentially according to the old rules:

$$\frac{\mathbb{W}; R \;\mapsto\; \mathbb{W}'; R'}{\mathbb{W}; \{R\} \uplus \Re \;\mapsto^c\; \mathbb{W}'; \{R'\} \uplus \Re}$$

We can then add primitives as desired to spawn new cursors. A very simple one evaluates M and N in parallel and returns each one to the same continuation.

$$\frac{\Gamma; \Delta \vdash M : A @ \mathrm{w} \quad \Gamma; \Delta \vdash N : A @ \mathrm{w}}{\Gamma; \Delta \vdash M|N : A @ \mathrm{w}}\ \text{par}$$

$$\mathbb{W}; \Re \uplus \{\mathbf{w}{:}[k \succ M|N]\} \mapsto^c \mathbb{W}; \Re \uplus \{\mathbf{w}{:}[k \succ M]\} \uplus \{\mathbf{w}{:}[k \succ N]\}$$

A suitable extension of type safety holds for $\mapsto^c$.

With concurrency in place we can implement asynchronous CML-style channels [12] with the help of continuations (and a few other features for developing mutable recursive structures). The type of a channel that allows sending and receiving of values of type A could be

$$A \ \texttt{chan} \;\doteq\; \Diamond(A \ \texttt{queue} \wedge (\neg A) \ \texttt{queue})$$

Here a channel is represented as the address of a pair of queues. In order to send to this channel, the sender must be able to bring a value of type A to the world where the channel lives. Therefore it must be a box or diamond type itself

[5] $\Sigma' \supseteq \Sigma$ iff Σ' and Σ each describe the same set of worlds, and for each world, if $X(\mathbf{k}) = A$ then $X'(\mathbf{k}) = A$, and likewise for β and β'.

(although the class of types that are mobile in this way can be easily extended; see the technical report for details [7]). The first queue holds the values that have been sent on the channel and not yet received; the second holds the continuations of outstanding `recieve`s. To implement `recieve` (assuming no values are waiting in the first queue), we grab the current continuation, enqueue it, and abort.

This is a standard technique; our point is to emphasize the utility of continuations as primitives for implementing useful distributed computing features.

6 Disjunction

To add disjunction to C5, we need to use the following elimination form in order to preserve the correspondence with classical S5:

$$\frac{\Gamma;\Delta \vdash M : A \vee B @ \omega' \quad \begin{array}{c}\Gamma, x{:}A @ \omega';\Delta \vdash N_1 : C @ \omega \\ \Gamma, x{:}B @ \omega';\Delta \vdash N_2 : C @ \omega\end{array}}{\Gamma;\Delta \vdash \mathtt{case}\, M\, \mathtt{of}\, \mathtt{inl}\, x \Rightarrow N_1 \mid \mathtt{inr}\, x \Rightarrow N_2 : C @ \omega}\ \vee E$$

This rule is completely unsurprising except that the case object M is at a *different world*, ω'. In our logic we've tried hard to avoid this sort of *action-at-a-distance*, instead preferring to have our introduction and elimination rules compute locally. However, a motion rule for disjunction is out of the question, because it is unsound: it is not the case that if $\Gamma;\Delta \vdash A \vee B @ \omega$ then necessarily $\Gamma;\Delta \vdash A \vee B @ \omega'$. In our previous paper we speculated that the remote case analysis could be implemented nonetheless by sending back merely a *bit* telling the case-analyzing world which branch it should enter, but this requires some suspicious operational machinery. The same is true in the classical case, which is why we have avoided treating disjunction so far.

As it turns out, support for disjunction and remote disjunction elimination is already present in C5, via one of de Morgan's laws. We define $A \vee B$ as $\neg(\neg A \wedge \neg B)$, and $A \vee B$ thus becomes a continuation that takes *two* continuations: one if the disjunct is A, and one if the disjunct is B. This technique is well-known for CPS conversion, and first-class continuations let us employ it without having to CPS-convert the entire program. Encoding the injections is easy:

$$\mathtt{inl}\, M \doteq \lambda x.(\pi_1 x)M \qquad\qquad \mathtt{inr}\, M \doteq \lambda x.(\pi_2 x)M$$

By grabbing the continuation at the point of case analysis, we can allow ourselves to move to a remote world (via `go`) to do the case analysis and rely on `throw` to get us back:

$$\begin{array}{l}\mathtt{case}\, M\, \mathtt{of}\, \mathtt{inl}\, x \Rightarrow N_1 \\ \quad \mid \mathtt{inr}\, x \Rightarrow N_2\end{array} \doteq \begin{array}{l}\mathtt{letcc}\, u\, \mathtt{in}\, \mathtt{go}[\omega']M\langle \lambda x.\, \mathtt{throw}\, N_1\, \mathtt{to}\, u, \\ \qquad\qquad \lambda x.\, \mathtt{throw}\, N_2\, \mathtt{to}\, u\rangle\end{array}$$

This has exactly the same typing conditions as the remote rule above; x is bound to the remote type $A @ \omega'$, even though the expression N_1 is evaluated at ω.

Classical logic is ripe with possibilities for definition. It is interesting to consider their implications. Recall that in Section 3 we proved $\Diamond A$ equivalent to

$\neg\Box\neg A$. This means that, like classical logicians, we could then just consider $\Diamond A$ a derived form. This would amount to a roundabout way of using the continuation table to publish values rather than the value table. Clearly, we could also take the even stranger route of defining $\Box A$ in terms of $\Diamond$, which gives us a mobile code "server" that sends code to our continuation whenever we like.

7 Conclusions

Related Work. Parigot's $\lambda\mu$-calculus has inspired many computational proof systems for classical logic, including Wadler's dual calculus [15]. The calculus is sequent-oriented and contains *cut* as a computational primitive, emphasizing the duality of computing with values and covalues (continuations). For programming in C5, we choose a natural deduction system which is deliberately *non*-dual. We bias the logic towards truth, which corresponds to computing mainly with values (as is typical) rather than covalues. Nevertheless, we expect that a dual version of classical S5 could be easily made to work, perhaps starting from the sequent calculus presented in the Appendix.

Because our calculus extends Lambda 5 [8], it is also related to the same mobile calculi, for example Moody's distributed S4 calculus [6], and Jia and Walker's S5-like hybrid logic [5]. Both calculi employ the $\Box$ and $\Diamond$ connectives with similar interpretations, though aspects of the underlying logics differ. Both give operational interpretations via concurrent process calculi with passive synchronization, and both systems use non-local introduction and elimination forms. In contrast, we achieve explicit active synchronization (in the form of $\texttt{get}_{\Diamond}$, etc.) along with what we feel are more primitive operations for constructing and deconstructing objects of the modal types. With regard to the classical extensions, we know of no prior modal system that features distributed continuations.

Future Work. Our language now has a full arsenal of connectives and control operators, each connected to logic. Much work remains before C5 can be a practical programming language rather than exploratory calculus. Some are routine—adding extra-logical primitives like recursion and references—and some are difficult—compilation of mobile code fragments, distributed garbage collection, failure recovery, and certification.

Although we believe that C5 accommodates concurrency easily, it would be nice to have a logically-inspired account of it. Some other directions remain open to try. Proof search in linear logic sequent calculus [3] is known to admit an interpretation as concurrent computation [2]. Perhaps linear S5 in sequent style would be able to elegantly express both spatial properties and concurrency in logic?

We have presented a proof theory and corresponding programming language, C5, based on the classical modal logic S5. By exploiting the modalities we are able to give a logical account of mobility and locality, and thus an expressive programming language for distributed computing. From the logic's classical nature we derive the mechanism of distributed continuations, which creates a new

connection between the $\Box$ and $\Diamond$ connectives, and forms a basis for the implementation of distributed computing primitives.

References

1. Andreas Abel. A third-order representation of the $\lambda\mu$-Calculus. In S.J. Ambler, R.L. Crole, and A. Momigliano, editors, *Electronic Notes in Theoretical Computer Science*, volume 58. Elsevier, 2001.
2. Jean-Yves Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92:69–108.
3. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, January 1987.
4. Timothy G. Griffin. The formulae-as-types notion of control. In *Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL'90, San Francisco, CA, USA, 17–19 Jan 1990*, pages 47–57. ACM Press, New York, 1990.
5. Limin Jia and David Walker. Modal proofs as distributed programs. *13th European Symposium on Programming*, pages 219–223, March 2004.
6. Jonathan Moody. Modal logic as a basis for distributed computation. Technical Report CMU-CS-03-194, Carnegie Mellon University, Oct 2003.
7. Tom Murphy, VII, Karl Crary, and Robert Harper. Distribed control flow with classical modal logic (technical report). Technical Report CMU-CS-04-177, Carnegie Mellon University, Dec 2004.
8. Tom Murphy, VII, Karl Crary, Robert Harper, and Frank Pfenning. A symmetric modal lambda calculus for distributed computing. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS 2004)*. IEEE Press, July 2004.
9. Chetan Murthy. Classical proofs as programs: How, what and why. Technical Report TR91-1215, Cornell University, 1991.
10. Michel Parigot. $\lambda\mu$-Calculus: An algorithmic interpretation of classical natural deduction. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning, International Conference LPAR'92, St. Petersburg, Russia, July 15-20, 1992, Proceedings*, volume 624 of *Lecture Notes in Computer Science*. Springer, 1992.
11. Frank Pfenning and Carsten Schürmann. System description: Twelf – a meta-logical framework for deductive systems. In Harald Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag. LNAI 1632.
12. John H. Reppy. *Concurrent Programming in ML*. Cambridge University Press, Cambridge, England, 1999.
13. Carsten Schürmann and Frank Pfenning. A coverage checking algorithm for LF. In D. Basin and B. Wolff, editors, *Proceedings of the 16th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2003)*, pages 120–135, Rome, Italy, September 2003. Springer-Verlag LNCS 2758.
14. Alex Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic*. PhD thesis, University of Edinburgh, 1994.
15. Philip Wadler. Call-by-value is dual to call-by-name. In *Proceedings of the 8th International Conference on Functional Programming (ICFP)*. ACM Press, August 2003.

8 Appendix

This appendix contains sketches of the proofs relating C5 to a classical S5 sequent calculus. This serves two purposes. First, because the sequent calculus is purely logical and does not feature our decomposition of the $\Box$ and $\Diamond$ rules, it is more obviously S5. Second, because the sequent calculus has the subformula property and admits cut, we get some standard results for our proof theory, such as the existence of normal forms and soundness. To begin, we need a few substitution theorems for our natural deduction system, one of which is interesting.

Falsehood Substitution. For each sort of hypothesis we have a substitution theorem. Worlds can be substituted for hypothetical worlds, and substitution $[M/x]N$ for truth hypotheses is defined in the standard way. Substitution for falsehood hypotheses warrants special attention, however:

Theorem 3 (Falsehood Substitution)
If $\quad \forall C, \omega''.\ \Gamma, x{:}A \mathbin{@} \omega; \Delta \vdash M : C \mathbin{@} \omega''$
and $\Gamma; \Delta, u{:}A \star \omega \vdash N : B \mathbin{@} \omega'$
then $\Gamma; \Delta \vdash [\![x.M/u]\!]N : B \mathbin{@} \omega'$.

This principle is dual to the # rule just as truth substitution is dual to the *hyp* rule. The # rule contradicts an $A \star \omega$ with an $A \mathbin{@} \omega$, so when substituting for a falsehood assumption, we are able to assume $A \mathbin{@} \omega$ and must produce another contradiction.We write falsehood substitution as $[\![x.M/u]\!]N$ where x is a binder (with scope through M) that stands for the value thrown to u. Just like truth substitution, it is defined pointwise on N except for the appropriate variable case (rule #):

$$[\![x.M/u]\!]\, \mathtt{throw}\, N' \,\mathtt{to}\, u \doteq [N'/x]M$$

Operationally, we see this as replacing the throw with some other handler for A. Since the new handler must have parametric type, typically it is a `throw` to some other continuation, perhaps after performing some computation on the proof of A.

Sequent Calculus. Our sequent calculus is motivated by simplicity and duality alone, because we will not give it a computational interpretation. One traditional way of doing classical theorem proving is to negate the target formula and prove a contradiction from it. Our sequent calculus (Fig. 6) is based on this view: the sequent $\Gamma \ \# \ \Delta$ means that the truth assumptions in Γ and the falsehood assumptions in Δ are mutually contradictory.[6] We treat contexts as unordered multisets, so the *action* can occur anywhere in either context. World hypotheses are placed in Γ, although to get a notationally dual system, we would place them in a third context "in the middle" of the sequent.

[6] Our rules are also consistent with the more traditional multiple-conclusion reading, "if all of Γ are true, then one of Δ is true."

$$\frac{}{\Gamma, A@\omega \ \# \ A\star\omega, \Delta}\ \text{contra} \qquad \frac{}{\Gamma, \bot@\omega \ \# \ \Delta}\ \bot T$$

$$\frac{\Gamma, A \supset B@\omega, B@\omega \ \# \ \Delta}{\Gamma, A \supset B@\omega \ \# \ \Delta}\ \supset T \qquad \frac{\Gamma, A@\omega \ \# \ B\star\omega, A \supset B\star\omega, D}{\Gamma \ \# \ A \supset B\star\omega, D}\ \supset F$$

$$\frac{\Gamma, \Box A@\omega, A@\omega' \ \# \ \Delta}{\Gamma, \Box A@\omega \ \# \ \Delta}\ \Box T \qquad \frac{\Gamma, \omega' \ \# \ A\star\omega', \Box A\star\omega, \Delta}{\Gamma \ \# \ \Box A\star\omega, \Delta}\ \Box F$$

$$\frac{\Gamma, \omega', \Diamond A@\omega, A@\omega' \ \# \ \Delta}{\Gamma, \Diamond A@\omega \ \# \ \Delta}\ \Diamond T \qquad \frac{\Gamma \ \# \ A\star\omega', \Diamond A\star\omega, \Delta}{\Gamma \ \# \ \Diamond A\star\omega, \Delta}\ \Diamond F$$

$$\frac{\Gamma, A \wedge B@\omega, A@\omega, B@\omega \ \# \ \Delta}{\Gamma, A \wedge B@\omega \ \# \ \Delta}\ \wedge T \qquad \frac{\Gamma \ \# \ A\star\omega, A \wedge B\star\omega, \Delta \quad \Gamma \ \# \ B\star\omega, A \wedge B\star\omega, \Delta}{\Gamma \ \# \ A \wedge B\star\omega, \Delta}\ \wedge F$$

Fig. 6. Classical S5 sequent calculus

These rules should be read bottom-up, as if during proof search. The *contra* rule allows us to form a contradiction whenever a proposition is both true and false at the same world. The $\Box T$ rule says that if we know $\Box A@\omega$, then we know $A@\omega'$ for any ω' that exists. On the other hand, if we know that $\Box A$ is false, then we know A is false at some world ω'. However, we must treat this world as hypothetical and fresh since we don't know which one it is. The rules for $\Diamond$ are perfect mirror images of the rules for $\Box$. The treatment of implication is standard, and follows from the classical truth tables.

We then wish to prove that the natural deduction and sequent calculus are equivalent (Theorem 5). The translation from natural deduction to the sequent calculus requires a lemma. In an intuitionistic calculus this would be *cut*; for the symmetric classical calculus it turns out to be the familiar classical notion of *excluded middle*.

Theorem 4 (Excl. Middle) *If* $\Gamma, A@\omega \ \# \ \Delta$ *and* $\Gamma \ \# \ A\star\omega, \Delta$ *then* $\Gamma \ \# \ \Delta$.

Proof of Theorem 4 is by lexicographic induction on the proposition A and then simultaneously on the two derivations. □

Theorem 5 (Equivalence)

(a) *If* $\Gamma; \Delta \vdash M : A@\omega$ *then* $\Gamma \ \# \ A\star\omega, \Delta$.

(b) *If* $\Gamma \ \# \ \Delta$ *then* $\exists M.\ \forall C, \omega.\ \Gamma; \Delta \vdash M : C@\omega$.

It is easy to see why 5(b) is the right statement. Since we think of $\Gamma \ \# \ \Delta$ as a proof of contradiction, this corresponds to a natural deduction derivation that proves any proposition at any world. Theorem 5(a) is more subtle. We show that if A is true under assumptions Γ and Δ, then A being false at the same world is contradictory with those assumptions. Computationally, we can think of this as the "final continuation" to which the result computed in natural deduction is passed. Putting these two theorems together, we have that $\Gamma; \Delta \vdash M : A@\omega$ gives $\Gamma \ \# \ A\star\omega, \Delta$, which then gives $\forall C, \omega'.\ \Gamma; \Delta, u{:}A\star\omega \vdash M' : C@\omega'$. In

particular, we choose $C = A$ and $\omega' = \omega$, and then by application of bc we have the original judgment (with a normalized proof term $\texttt{letcc}\, u\, \texttt{in}\, M'$). Thus $\vdash$ and # are really equivalent.

The proof of Theorem 5(a) is by straightforward induction on the derivation, using Theorem 4 where necessary. (The structural rules bc and # just become uses of contraction and weakening in the sequent calculus.) □

Proof of 5(b) is interesting because of its manipulation of continuations through the use of falsehood substitution (Theorem 3). Uses of T rules are easy; they correspond directly to the elimination rules[7] in natural deduction. But since our natural deduction is biased towards manipulating truth rather than falsehood, the F rules are more difficult and make nontrivial use of the falsehood substitution theorem. For instance, in the $\wedge F$ case we have by induction:

$$\Gamma; \Delta, u_p{:}A \wedge B \star \omega, u_a{:}A \star \omega \vdash N_1 : C \mathbin{@} \omega' \ (\forall C, \omega')$$
$$\Gamma; \Delta, u_p{:}A \wedge B \star \omega, u_b{:}B \star \omega \vdash N_2 : C \mathbin{@} \omega' \ (\forall C, \omega')$$

By two applications of Theorem 3, we get that the following proof term has any type at any world:

$$\Big[\!\Big[x.[\![y.\, \texttt{throw}\, \langle x, y\rangle\, \texttt{to}\, u_p / u_b]\!] N_2 \,/ u_a \Big]\!\Big] N_1$$

We form an innermost `throw` of the pair $\langle x, y\rangle$ to our pair continuation u_p. This has free truth hypotheses $x : A$ and $y : B$. Therefore, we can use it to substitute away the u_b continuation in N_2 (any throw of M to u_b becomes a throw of $\langle x, M\rangle$ to u_p). Finally, we can use this new term to substitute away u_a in N_1, giving us a term that depends only on the pair continuation u_p. This pattern of *prepending* work onto continuations through substitution is characteristic of this proof.

The case for $\Box F$ is interesting because it uses `letcc`.[8] By induction we have:

$$\forall \omega'.\ \ \Gamma; \Delta, u{:}A \star \omega', u_b{:}\Box A \star \omega \vdash N : C \mathbin{@} \omega'' \quad (\forall C, \omega'')$$

Then the proof term witnessing the theorem here is:

$$\texttt{throw}(\texttt{box}\ \omega'.\, \texttt{letcc}\, u\, \texttt{in}\, N)\, \texttt{to}\, u_b$$

It is not possible to use falsehood substitution on u in this case. To do so we would need to turn a term of type $A \mathbin{@} \omega'$ into a $\Box A \mathbin{@} \omega$ to throw to u_b. Although at a meta-level we know that we can choose any ω', it won't be possible to internalize this in order to create a $\Box A$. Instead we must introduce a new box, and choose ω' to be the new hypothetical world that the $\Box I$ rule introduces. At that point we use `letcc` to create a real $A \star \omega'$ assumption to discharge u. The remaining cases are similar or straightforward, and can be found in full detail in the Twelf code.[9] □

[7] Except for implication, which is phrased differently in the sequent calculus.

[8] In fact, this is the only place in the proof where a `letcc` is necessary. This gives a normal form for natural deduction terms where `letcc` appears only once at the outermost scope and immediately inside each `box`.

[9] The most natural LF encoding of falsehood is 3^{rd}-order [1]; we use a 2^{nd}-order encoding in our proofs (proving the falsehood substitution theorem by hand) because third-order metatheorem checking is not yet available in the distribution.