Logical Mobility and Locality Types

Jonathan Moody*

jwmoody@cs.cmu.edu
Carnegie Mellon University

Abstract. We present a type theory characterizing the mobility and locality of program terms in a distributed computation. The type theory of our calculus is derived from logical notions of necessity ($\Box A$) and possibility ($\Diamond A$) of the modal logic $S4$ via a Curry-Howard style isomorphism. Logical worlds are interpreted as sites for computation, accessibility corresponds to dependency between processes at those sites. Necessity ($\Box A$) describes terms of type A which have a structural kind of mobility or location-independence. Possibility ($\Diamond A$) describes terms of type A located somewhere, perhaps at a remote site. The modalities $\Box$ and $\Diamond$ are defined in a clean, orthogonal manner, leading to a simple account of mobility and higher-order functions. For illustration, we assume an execution environment with each location distinguished by a mutable store. Here modal types ensure that store addresses never escape from the location where they are defined, eliminating a source of runtime errors. We speculate as to other advantages or trade-offs of this disciplined style of distributed programming.

1 Introduction and Motivation

We claim that modal logic with necessity ($\Box A$) and possibility ($\Diamond A$) can serve as the basis of a *location-aware* type theory for distributed computation. We present a statically typed calculus derived from a natural deduction formulation of $S4$ modal logic — derived, in the sense that programs correspond to proof terms, and types to propositions. The modal propositions $\Box A$ ("mobile A") and $\Diamond A$ ("remote A") capture spatial properties of terms relevant to distributed computation. Mobility and locality are explicitly recognized, but the particular locations involved remain abstract.

We then give an operational interpretation compatible with the intuition that types $\Box A$ and $\Diamond A$ denote mobility and locality, respectively. Our operational model assumes a number of definite locations (computation sites) distinguished by fixed resources, as well as some indefinite or interchangeable locations. We consider mutable stores as an example of a fixed resource. To illustrate this application, we consider an extension of the pure $S4$ theory with mutable references and a simple form of effect typing.

* Acknowledgements: Frank Pfenning for feedback on a draft of this paper. This material is based upon work supported under a National Science Foundation Graduate Research Fellowship.

The logical reading of the typing rules leads naturally to the following operational interpretation: $\Box$ elimination spawns a freely mobile term of type $\Box A$ for evaluation at an arbitrary, indefinite location, and $\Diamond$ elimination sends a mobile fragment of code to a definite location where a remote resource of type $\Diamond A$ resides. Our intuitions are validated when we show type safety for the semantics. Only terms known to be portable (free of store dependencies) are moved between locations.

While it is desirable to insulate the programmer from the details of scheduling and communication in a distributed program, spatial considerations are entwined with certain language constructs at the level of implementation. A memory address or file handle, for example, is meaningful only at a particular machine. Implementation techniques such as remote references (proxy objects) and a clever marshalling scheme allow a language designer to conceal these exceptional cases. But this has costs in terms of complexity and performance. Rather, we choose to consider programs based on a simpler kind of mobility. The type theory of $S4$ is a static characterization of this class of programs — those that respect the locality of fixed resources without recourse to runtime mechanisms for creating and managing remote references.

In this work, we focus on the logical origins of mobility and locality, presenting the type theory, and exploring its application to distributed programming. But it is important to note that, from an *operational* perspective, the behaviors we describe are not novel. Our process spawning model is similar to futures of Multilisp [13], and jumping (or something similar) is a feature of many mobile process calculi such as Mobile Ambients [6], DPI [9], and others [18, 16]. Jagannathan [10] calls it a "communication-passing" style. In section 6, we discuss other distributed calculi, their approach to modeling locations, and their static type theories (if any). Most closely related are calculi due to Jia and Walker [12, 11], and Murphy *et. al* [20, 21]. These authors give a distributed interpretation to the modalities $\Box A$ and $\Diamond A$ — one based on $S5$ semantics (as opposed to $S4$). Though the methodologies are similar, the calculus derived from $S4$ is qualitatively different, both in terms of execution model and the programming discipline it imposes.

In section 2 we present an overview of the logical foundations of the type theory, referring to prior work [17]. We then define the calculus and its type system in section 3, justifying the typing rules by reference to logical meanings of $\Box$ and $\Diamond$. In section 4, we give an operational semantics and prove its type safety. In sections 5 and 6 we reflect on the programming discipline of mobility and locality types and how this solution is related to others.

2 Logical Preliminaries

Modal logic is built on a foundational assumption that truth is "localized". The Kripke semantics for a classical modal logic ascribes to each world w a local valuation function $\mathrm{V}_w(A_0)$ for the atomic propositions. Thus proposition A_0 may be true at world w but false at w'. This localized conception of truth is what gives modal logic the capacity to describe distributed computation. In a constructive formulation, we no longer have localized truth valuations, but proofs of a certain form may be portable, establishing the truth of A in the context of assumptions true at any accessible world. Others proofs may be tied to a particular world. When removed from that local context, they no longer establish truth of A.

The types, syntax, and static semantics of our calculus are derived from a constructive formalization of modal logic developed by Pfenning and Davies [17]. This was chosen over other intuitionistic formalisms, such as Simpson's [19], since proof reduction and substitution have simple explanations, and the logic does not rely on explicit reasoning about worlds and accessibility. The Pfenning/Davies formalism is based on three primitive judgments on A, a proposition: A `true`, meaning that A is locally true "here"; A `valid`, meaning that A `true` holds in *every* accessible world; and A `poss`, meaning that A `true` holds in *some* accessible world. Validity (A `valid`) is also commonly referred to as necessary truth. These judgments and the propositions $A \to B$ (implication), $\Box A$ (necessity), and $\Diamond A$ (possibility) are defined in relationship to one another, culminating in a natural deduction system for a modal logic supporting axioms characteristic of constructive S4. The primary judgments are $\Delta; \Gamma \vdash A\ \mathtt{true}$ and $\Delta; \Gamma \vdash A\ \mathtt{poss}$, where Δ are assumptions of the form $A_i\ \mathtt{valid}$, and Γ are assumptions $A_i\ \mathtt{true}$.

The intuition behind our application of modal logic to distributed programming is the following: If, following the Curry-Howard approach, we interpret propositions as types and proofs as programs, it is also quite natural to interpret the logical worlds as sites for computation. Proofs of validity (A `valid`) correspond to mobile, portable terms. And proofs of possibility (A `poss`) correspond to computations that produce a term in some definite, perhaps remote, location. An extended discussion of the background and logical motivation of the calculus can be found in the technical report [15].

3 Modal Type Theory and Calculus

The syntax of terms M and types A is given in figure 1. We call the fragment with types $A \to B$, $\Box A$ and $\Diamond A$ the core calculus. To support interesting examples, we include several extensions: generic effectful computation characterized by the monadic type $\bigcirc A$, and mutable references $\mathtt{ref}\ A$ as store-effects. We also assume a set of base types `nat`, `bool`, `1` (unit type), *etc.*

Corresponding to the judgments A `true` and A `poss` of the modal deduction system, we have typing judgments $M : A$ and $M \div A$, respectively. To characterize local, effectful computations we add a third form of judgment $M \,\tilde{\rightarrow}\, A$. We summarize the typing judgments and their informal readings:

Judgment	Significance
$\Delta; \Gamma \vdash M : A$	M has type A here (local pure term)
$\Delta; \Gamma \vdash M \,\tilde{\rightarrow}\, A$	M produces result of type A here (local effects)
$\Delta; \Gamma \vdash M \div A$	M produces result somewhere (non-localized comp.)

	Types (A, B)	Syntactic Forms (M, N)
Local	$A \to B$	$\mathtt{x} \mid \lambda\mathtt{x}{:}A.\,M \mid M\ N$
	$\mathtt{1}, \mathtt{bool}, \mathtt{nat}$	$\mathtt{()} \mid \mathtt{true} \mid \mathtt{false} \mid$ (defn. elided ...)
Spatial	$\Box A$	$\mathtt{u} \mid \mathtt{box}\,M \mid \mathtt{let\,box\,u} = M\,\mathtt{in}\,N$
	$\Diamond A$	$\mathtt{dia}\,M \mid \mathtt{let\,dia\,x} = M\,\mathtt{in}\,N$
Effects	$\bigcirc A$	$\mathtt{comp}\,M \mid \mathtt{let\,comp\,x} = M\,\mathtt{in}\,N$
	$\mathtt{ref}\,A$	$\mathtt{ref}\,M \mid\ !M \mid M := N$

Fig. 1. Types and terms of the modal calculus.

The hypothetical form(s) $\Delta; \Gamma \vdash M : A$, *etc.* are made in the context of some mobile variables $\mathtt{u} :: A \in \Delta$, corresponding to assumptions A `valid`, and local variables $\mathtt{x} : A \in \Gamma$ corresponding to assumptions A `true`. The typing rules are presented in figure 2.

Locations are implicit in the local context Γ, in the following sense: $\Delta; \Gamma \vdash M : A$ means M has type A *in a location where* local bindings for Γ are available. Thus certain forms of judgment have a special significance. $\Delta; \cdot \vdash M : A$, means that $M : A$ *any* place via weakening on Γ. This form of typing derivation corresponds to a proof of A `valid` in the deduction system. By analogy, $\Delta; \mathtt{x} : B \vdash M \div A$, means that $M \div A$ holds *in a location where* $\mathtt{x} : B$.

3.1 Spatial Content of Typing

Neither the deductive formalism [17] nor the typing rules involve locations explicitly, but a few of the typing rules nonetheless have spatial content, by virtue of interaction between the local context Γ, and judgments $M : A$, $M \div A$, *etc.* For example, rule $\Box I$ states that mobile terms are those which depend only on other mobile terms in Δ. Effects are prohibited under $\mathtt{box}\,M$ by requiring $M : A$.[1] The elimination form allows us to bind such a term to $\mathtt{u} :: A$ in Δ. Such variables have a scope extending beyond the confines of a single location, so it is essential that only mobile values be bound to $\mathtt{u} :: A$.

$$\frac{\Delta; \cdot \vdash M : A}{\Delta; \Gamma \vdash \mathtt{box}\,M : \Box A}\ \Box I \qquad \frac{\Delta; \Gamma \vdash M : \Box A \quad \Delta, \mathtt{u} :: A; \Gamma \vdash N : B}{\Delta; \Gamma \vdash \mathtt{let\,box\,u} = M\,\mathtt{in}\,N : B}\ \Box E$$

The rule $\bigcirc E$ describes binding the result of a local computation $M : \bigcirc A$ to a local variable $\mathtt{x} : A$. Of course, some variables in Γ could be bound to values dependent on prior effects. Under the assumption that effects are not destructive, it is sound to retain the local context Γ in typing $\Delta; \Gamma, \mathtt{x} : A \vdash N \sim B$.

$$\frac{\Delta; \Gamma \vdash M : \bigcirc A \quad \Delta; \Gamma, \mathtt{x} : A \vdash N \sim B}{\Delta; \Gamma \vdash \mathtt{let\,comp\,x} = M\,\mathtt{in}\,N \sim B}\ \bigcirc E$$

[1] This might be weakened to allow some benign, non-observable classes of effects, but executing I/O effects at an arbitrary location leads to unpredictable behavior.

$$\begin{array}{rl}\text{Mobile Context} & \Delta ::= \cdot \;\mid\; \Delta, \mathtt{u}{::}A \\ \text{Local Context} & \Gamma ::= \cdot \;\mid\; \Gamma, \mathtt{x}{:}A\end{array}$$

$\boxed{\Delta; \Gamma \vdash M : A}$

$$\frac{}{\Delta; \Gamma, \mathtt{x} : A, \Gamma' \vdash \mathtt{x} : A}\ hyp \qquad \frac{}{\Delta, \mathtt{u} :: A, \Delta'; \Gamma \vdash \mathtt{u} : A}\ hyp^{*}$$

$$\frac{\Delta; \Gamma, \mathtt{x} : A \vdash M : B}{\Delta; \Gamma \vdash \lambda \mathtt{x}{:}A \,.\, M : A \to B} \to I \qquad \frac{\Delta; \Gamma \vdash M : A \to B \quad \Delta; \Gamma \vdash N : A}{\Delta; \Gamma \vdash M\ N : B} \to E$$

$$\frac{\Delta; \cdot \vdash M : A}{\Delta; \Gamma \vdash \mathtt{box}\, M : \Box A}\ \Box I \qquad \frac{\Delta; \Gamma \vdash M : \Box A \quad \Delta, \mathtt{u} :: A; \Gamma \vdash N : B}{\Delta; \Gamma \vdash \mathtt{let\,box\,u} = M\, \mathtt{in}\, N : B}\ \Box E$$

$$\frac{\Delta; \Gamma \vdash M \leadsto A}{\Delta; \Gamma \vdash \mathtt{comp}\, M : \bigcirc A}\ \bigcirc I \qquad \frac{\Delta; \Gamma \vdash M \div A}{\Delta; \Gamma \vdash \mathtt{dia}\, M : \Diamond A}\ \Diamond I$$

$\boxed{\Delta; \Gamma \vdash M \leadsto A}$

$$\frac{\Delta; \Gamma \vdash M : A}{\Delta; \Gamma \vdash M \leadsto A}\ comp \qquad \frac{\Delta; \Gamma \vdash M : \bigcirc A \quad \Delta; \Gamma, \mathtt{x} : A \vdash N \leadsto B}{\Delta; \Gamma \vdash \mathtt{let\,comp\,x} = M\, \mathtt{in}\, N \leadsto B}\ \bigcirc E$$

$$\frac{\Delta; \Gamma \vdash M : A}{\Delta; \Gamma \vdash \mathtt{ref}\, M \leadsto \mathtt{ref}\ A}\ talloc \qquad \frac{\Delta; \Gamma \vdash M : \Box A \quad \Delta, \mathtt{u} :: A; \Gamma \vdash N \leadsto B}{\Delta; \Gamma \vdash \mathtt{let\,box\,u} = M\, \mathtt{in}\, N \leadsto B}\ \Box E_c$$

$$\frac{\Delta; \Gamma \vdash M : \mathtt{ref}\ A}{\Delta; \Gamma \vdash\ !M \leadsto A}\ tget \qquad \frac{\Delta; \Gamma \vdash M : \mathtt{ref}\ A \quad \Delta; \Gamma \vdash N : A}{\Delta; \Gamma \vdash M := N \leadsto \mathtt{1}}\ tset$$

$\boxed{\Delta; \Gamma \vdash M \div A}$

$$\frac{\Delta; \Gamma \vdash M : A}{\Delta; \Gamma \vdash M \div A}\ poss \qquad \frac{\Delta; \Gamma \vdash M : \Diamond A \quad \Delta; \mathtt{x} : A \vdash N \div B}{\Delta; \Gamma \vdash \mathtt{let\,dia\,x} = M\, \mathtt{in}\, N \div B}\ \Diamond E$$

$$\frac{\Delta; \Gamma \vdash Q \leadsto A}{\Delta; \Gamma \vdash Q \div A}\ poss' \qquad \frac{\Delta; \Gamma \vdash M : \Box A \quad \Delta, \mathtt{u} :: A; \Gamma \vdash N \div B}{\Delta; \Gamma \vdash \mathtt{let\,box\,u} = M\, \mathtt{in}\, N \div B}\ \Box E_p$$

$$\frac{\Delta; \Gamma \vdash M : \bigcirc A \quad \Delta; \Gamma, \mathtt{x} : A \vdash N \div B}{\Delta; \Gamma \vdash \mathtt{let\,comp\,x} = M\, \mathtt{in}\, N \div B}\ \bigcirc E_p$$

$$\begin{tikzcd}
\Delta; \Gamma \vdash M : A \arrow[dd, dashed, "\text{poss}"'] \arrow[dr, dashed, "\text{comp}"] & \\
& \Delta; \Gamma \vdash M \leadsto A \arrow[dl, dashed, "\text{poss'}"] \\
\Delta; \Gamma \vdash M \div A &
\end{tikzcd}$$

Fig. 2. Typing rules for the core modal calculus extended with effects. Allocation, dereference, update of reference cells are considered effectful computations. Rules for base types, other type constructors, *etc.* are omitted since they are orthogonal and defined in the usual ways under local term typing ($\Delta; \Gamma \vdash M : A$). The diagram illustrates subsumptions between the judgments (*poss*, *poss'* and *comp*).

Finally, $\Diamond E$ describes the binding of a remote term value to a local variable $\mathtt{x} : A$. For this to be sound, it must be the case that the continuation $\Delta; \mathtt{x} : A \vdash N \div B$ be well-formed at the remote location. This is ensured by restricting the local context to a single binding $\mathtt{x} : A$ available at some remote location. The $\Diamond I$ rule allows us to encapsulate a term $M \div A$ as a remote term $\Diamond A$.

$$\frac{\Delta; \Gamma \vdash M \div A}{\Delta; \Gamma \vdash \mathtt{dia}\, M : \Diamond A}\ \Diamond I \qquad \frac{\Delta; \Gamma \vdash M : \Diamond A \quad \Delta; \mathtt{x} : A \vdash N \div B}{\Delta; \Gamma \vdash \mathtt{let\,dia\,x} = M\ \mathtt{in}\, N \div B}\ \Diamond E$$

3.2 Examples

The definition of $\Box A$ (mobility) is orthogonal to the rest of the type constructors; even lexically scoped closures of type $\mathtt{nat} \to \mathtt{nat}$ can be mobile $\Box(\mathtt{nat} \to \mathtt{nat})$. This is more powerful than ad-hoc restrictions on mobility based on the form of types.

```
let plusk : □nat -> □(nat -> nat) =
  (λ x : □ nat .
    let box k = x in
      box (λ y : nat . y + k))
(* incr a mobile function *)
let box incr :: nat -> nat = plusk (box 1)
```

Moving the closure representation of `incr` is sound since we know the free variable $\mathtt{k} :: \mathtt{nat}$ in $\mathtt{box}\,(\lambda \mathtt{y{:}nat}\,.\,\mathtt{y} + \mathtt{k})$ is bound to a mobile value.

Mobile terms ($\mathtt{box}\, M : \Box A$) are Γ-closed and free of local dependencies. They can be evaluated at *any* location without regard to local resources. The elimination form $\mathtt{let\,box\,u} = \mathtt{box}\, M\ \mathtt{in}\, N$ spawns M for evaluation in an independent process (at an arbitrary location). It is straightforward to introduce parallelism with `box` and `let box`. Consider the Fibonacci function, implemented in a recursive fashion:

```
let fib : □nat -> nat =
    mfix f . λ bn .
        let box n = bn in
        if n < 2 then n
        else let box f1 = box f (box (n-1)) in
             let box f2 = box f (box (n-2)) in
                 f1 + f2
in fib (box 5)
```

Here the definition of `f` must itself be mobile, since `f` occurs under box. The typing rule for mobile fixpoint `mfix` requires that the body be Γ-closed analogous to the rule $\Box I$. Fixpoint operators are defined in appendix section 8.3.

Values of type $\Diamond A$ can be used to model locations with special roles during computation. Type A describes the resource/interface provided by that site. Non-trivial values of this type must be obtained through a primitive binding mechanism. We can form a term $\mathtt{dia}\, M : \Diamond A$ in the source language, but M is only "remote" in a trivial sense.

```
(* rqueue : ◇({insert:nat->◯unit, ...}) *)
let rqueue = bind_queue ... in

(* insert (x : □nat) into rqueue *)
let box v = x in
let dia q = rqueue in (* jump to queue location *)
  let comp () = q.insert v in
  ...
```

The actual location of `rqueue` is hidden by the type constructor $\Diamond$, so the binding mechanism is free to choose which location will provide the service. Our type discipline requires that the code sent to the remote location be free of any dependence on local bindings in Γ with the exception of `q` itself, which can be bound upon arrival. Primitive remote resources and their properties are discussed in the appendix, section 8.4.

4 Operational Interpretation

4.1 Model of Locations

We now formalize the operational semantics in a way that is consistent with the logical readings of $\Box A$ and $\Diamond A$ described above. The semantics should reflect clearly the spatial distribution of program fragments, so that communication (movement) of terms is evident. To this end we introduce processes $\langle l : M \rangle$ consisting of a term M labeled uniquely by l. The semantics should also represent concretely the distinguishing features of each location. In this instance, locations are distinguished by a store H mapping addresses a to values. Finally, processes are placed in structured configurations C that reflect the relationships between processes, stores, and other processes. See figure 3.

$$
\begin{array}{rl}
\text{Store} & H ::= \cdot \;\mid\; H[a \mapsto \overline{V}] \\[1ex]
\text{Co-located Processes} & P ::= \cdot \;\mid\; P, \langle l : M \rangle \\
\text{Configuration} & C ::= \cdot \;\mid\; \langle l : M \rangle \triangleleft C \;\mid\; [H \vDash P] \triangleleft C
\end{array}
$$

Fig. 3. Runtime Structures: processes, stores, and configurations

The notation $[H \vDash P]$ represents a collection of processes P executing inside a definite location under the store H. Some processes have no definite location, only a placement $\langle l : M \rangle \triangleleft C$ relative to other processes C. If one thinks of processes as worlds of a Kripke model, the connective $\triangleleft$ can be viewed as an assertion of accessibility. For example, $[H \vDash P] \triangleleft C$ means that processes C are accessible from P.

We permit process labels and store addresses in terms at runtime. Process labels occur in two forms: l and $@l$. Both refer to a process $\langle l : M\rangle$ but l denotes the result value of a mobile process and $@l$ denotes the value "at" process l (which may not be mobile). Typing contexts Δ and Γ are generalized to account for labels and addresses. We provide typing rules for these new syntactic forms in figure 4.

$$\text{Runtime Term} \quad M, N ::= \ldots \;\mid\; l \;\mid\; @l \;\mid\; a$$

$$\begin{array}{rl} \text{Mobile Context} & \Delta ::= \cdot \;\mid\; \Delta, \mathtt{u} :: A \;\mid\; \Delta, l :: A \\ \text{Local Context} & \Gamma ::= \cdot \;\mid\; \Gamma, \mathtt{x} : A \;\mid\; \Gamma, a : A \;\mid\; \Gamma, @l : A \end{array}$$

$$\frac{\Delta = \Delta_1, l :: A, \Delta_2}{\Delta; \Gamma \vdash l : A}\,res \qquad \frac{\Gamma = \Gamma_1, @l : A, \Gamma_2}{\Delta; \Gamma \vdash @l \div A}\,loc \qquad \frac{\Gamma = \Gamma_1, a : A, \Gamma_2}{\Delta; \Gamma \vdash a : \mathtt{ref}\ A}\,addr$$

$\boxed{\Delta \vdash^c C : \Gamma}$

$$\frac{\Delta; \cdot \vdash M : A \quad \Delta, l :: A \vdash^c C : \Gamma}{\Delta \vdash^c \langle l : M\rangle \triangleleft C : \Gamma}\,indef \qquad \frac{}{\Delta \vdash^c \cdot : \cdot}\,none$$

$$\frac{\Delta \vdash^c C : \Gamma \quad \Delta; \Gamma \vdash^s H : \Gamma^H \quad \Delta; \Gamma, \Gamma^H \vdash^c P : \Gamma'}{\Delta \vdash^c [H \vDash P] \triangleleft C : \Gamma', \Gamma}\,location$$

$\boxed{\Delta; \Gamma \vdash^c P : \Gamma'}$

$$\frac{\Delta; \Gamma \vdash^c P : \Gamma' \quad \Delta; \Gamma \vdash M \div A}{\Delta; \Gamma \vdash^c P, \langle l : M\rangle : \Gamma', @l : A}\,proc \qquad \frac{}{\Delta; \Gamma \vdash^c \cdot : \cdot}\,empty$$

$\boxed{\Delta; \Gamma \vdash^s H : \Gamma'}$

$$\Delta; \Gamma \vdash^s H : \Gamma' \iff \forall a \in \mathrm{Dom}(H \cup \Gamma') \,.\, \Delta; \Gamma, \Gamma' \vdash H(a) : \Gamma'(a)$$

Fig. 4. Runtime terms, generalized typing contexts, and configuration typing.

A configuration C is well-formed iff $\cdot \vdash^c C : \Gamma$. That is, all processes in C are well-formed, and the processes P at definite locations $[H \vDash P]$ have types given by Γ. See figure 4 for the definition. There are subsidiary judgments for typing stores $\Delta; \Gamma \vdash^s H : \Gamma'$, and co-located processes $\Delta; \Gamma \vdash^c P : \Gamma'$. The label-binding structure of a configuration is determined by accessibility ($\triangleleft$). The form $\langle l : M\rangle \triangleleft C$ binds $l :: A$ in the subsequent portion C. The form $[H \vDash P] \triangleleft C$ binds the labels $l_i : A_i$ due to C in the processes P. Local store addresses a defined by H are also bound in P.

4.2 Substitution and Values

We adopt the definitions of substitution from Pfenning and Davies [17] with trivial extensions to account for labels and store addresses. There are multiple forms of substitution, two of which are relevant here. $[\![M/\mathtt{u}]\!]$ is the substitution of a mobile term for $\mathtt{u}$ and $[M/\mathtt{x}]$ is substitution of a local term for $\mathtt{x}$. They are defined in the usual compositional way, avoiding variable capture. Labels and store addresses denote syntactically closed terms, so $[\![M/\mathtt{u}]\!]@l = @l$, for example. Because of this, we can say that substitution acts *locally* (within a single process). Variables of the two sorts have different typing properties so the relevant substitution properties are subtly different.

Lemma 1 (Substitution Properties)

$$
\begin{array}{lcl}
\Delta; \cdot \vdash M : A \;\; \wedge \; \Delta, \mathtt{u} :: A, \Delta'; \Gamma \vdash N : B & \Longrightarrow & \Delta, \Delta'; \Gamma \vdash [\![M/\mathtt{u}]\!]N : B \\
\Delta; \cdot \vdash M : A \;\; \wedge \; \Delta, \mathtt{u} :: A, \Delta'; \Gamma \vdash N \mathrel{\tilde{\rightarrow}} B & \Longrightarrow & \Delta, \Delta'; \Gamma \vdash [\![M/\mathtt{u}]\!]N \mathrel{\tilde{\rightarrow}} B \\
\Delta; \cdot \vdash M : A \;\; \wedge \; \Delta, \mathtt{u} :: A, \Delta'; \Gamma \vdash N \div B & \Longrightarrow & \Delta, \Delta'; \Gamma \vdash [\![M/\mathtt{u}]\!]N \div B \\
\Delta; \Gamma \vdash M : A \wedge \Delta; \Gamma, \mathtt{x} : A, \Gamma' \vdash N : B & \Longrightarrow & \Delta; \Gamma, \Gamma' \vdash [M/\mathtt{x}]N : B \\
\Delta; \Gamma \vdash M : A \wedge \Delta; \Gamma, \mathtt{x} : A, \Gamma' \vdash N \mathrel{\tilde{\rightarrow}} B & \Longrightarrow & \Delta; \Gamma, \Gamma' \vdash [M/\mathtt{x}]N \mathrel{\tilde{\rightarrow}} B \\
\Delta; \Gamma \vdash M : A \wedge \Delta; \Gamma, \mathtt{x} : A, \Gamma' \vdash N \div B & \Longrightarrow & \Delta; \Gamma, \Gamma' \vdash [M/\mathtt{x}]N \div B
\end{array}
$$

Proof: straightforward, by induction on the typing derivation for N. The property is established for derivations of $N : B$ first, then $N \mathrel{\tilde{\rightarrow}} B$ assuming the former case, then $N \div B$ assuming substitution properties hold in both the former cases. □

Substitutions $[\![M/\mathtt{u}]\!]N$ and $[M/\mathtt{x}]N$ are only properly defined for terms M satisfying $M : A$. For example, $[\![\mathtt{let\,dia\,x} = M \;\mathtt{in}\, N/\mathtt{u}]\!]$ is undefined. Terms $M \mathrel{\tilde{\rightarrow}} A$ and $M \div A$ have different logical properties which are not respected by ordinary substitution. Special forms of substitution can be defined for these cases (see [17]), but our operational semantics is designed so that only $[\![M/\mathtt{u}]\!]N$ and $[M/\mathtt{x}]N$ are required.

The values of the calculus are as follows, eliding values of base type which are standard. Local values correspond to the typing judgment $\Delta; \Gamma \vdash V : A$ (or $V \mathrel{\tilde{\rightarrow}} A$). General values correspond to $\Delta; \Gamma \vdash V^* \div A$.

$$
\begin{array}{rrl}
\text{Local Value} & V ::= & \mathtt{x} \mid \mathtt{u} \mid \lambda\mathtt{x}{:}A \,.\, M \mid \mathtt{box}\, M \mid \mathtt{dia}\, M \mid \mathtt{comp}\, M \mid a \mid \ldots \\
\text{Value} & V^* ::= & @l \mid V
\end{array}
$$

To achieve more concurrency, we treat labels l as pseudo-values, though they can be reduced further by synchronizing on the result of process l. We use the notation $\overline{V}$ to denote local value *or* label l.

4.3 Reduction Rules

We use evaluation context notation to specify where reduction steps may occur inside of terms; $\mathcal{R}[\,N\,]$ denotes a term decomposed into context $\mathcal{R}[\;]$ and subterm N. Any well-formed term has one or more decompositions. Non-uniqueness arises from the treatment of labels r. Pseudo-values $\overline{V}$ occur throughout the definition of contexts and redices, so $M = l\ N$ is decomposed as either $\mathcal{R} = [l]\ N$ (function position) or $\mathcal{R} = l\ \mathcal{R}'$ (argument position).

We present the reduction rules $C \Longrightarrow C'$ in a way that elides unchanged, irrelevant parts of the configuration. For example, the rule $\langle l : M \rangle \Longrightarrow \langle l : M' \rangle$ applies to any single process, in a definite location $[H \vDash \ldots]$ or not. See figure 5.

The most revealing reduction rules are those governing $\Box$ and $\Diamond$ introduction and elimination. This is where the *spatial* content of the calculus is found, and our semantics permits creation and interaction between processes in these rules. To reduce ($\texttt{let box u = box}\ M\ \texttt{in}\ N$), we spawn an independent process l_2 to carry out the evaluation of subterm M. The spawned process l_2 is placed outside of a definite location, reflecting the fact that M may be evaluated *anywhere*. The rule *syncr* allows retrieving the mobile result value of such a process. Reducing ($\mathcal{R}[\ \texttt{let dia x = dia}\ @l_2\ \texttt{in}\ N\ ]$) involves sending $\mathcal{R}[\ ]$ and N to the specific location where $\langle l_2 : \overline{V} \rangle$ resides. Notice that the value $\overline{V}$ is never moved outside $[H \vDash \ldots]$, though it is duplicated in the fresh process l_3. Rule *resolve* allows traversing chains of indirection to locate a remote value, and *letdia* covers the trivial case of a local term.

There are also pure and effectful local reduction steps. The local reductions involve only one process, and, in the latter case, the local store H. Reductions associated with additional base types, products $(A \times B)$, sums $(A + B)$, *etc.* would be in this family.

4.4 Properties

Type preservation and progress theorems hold for our semantics; mobility ($\Box A$) and locality ($\Diamond A$) types ensure that our distributed programs are safe. This would be unremarkable but for the presence of certain localized terms in our semantics — the store addresses a. The criterion for well-formed configurations specifies that addresses bound by store H only occur in processes P inside the definite location $[H \vDash P]$. The modal type discipline ensures that programs respect the locality of store addresses.

For well-formed decompositions $\mathcal{R}[\ M\ ]$, there is an inversion lemma which allows us to conclude M is also well-formed. This is a standard property for evaluation contexts, but in our case M might satisfy more than one of the three typing judgments. Some decompositions are ruled out by typing, $\mathcal{R}[\ M\ ] : B$ and $M \rightsquigarrow A$, for example, are incompatible.

Lemma 2 (Context Typing Inversion)

$$\begin{array}{l}
\Delta; \Gamma \vdash \mathcal{R}[\ M\ ] : B \implies \exists A\ .\ \Delta; \Gamma \vdash M : A \\
\Delta; \Gamma \vdash \mathcal{R}[\ M\ ] \rightsquigarrow B \implies \exists A\ .\ \Delta; \Gamma \vdash M : A\ \vee\ \Delta; \Gamma \vdash M \rightsquigarrow A \\
\Delta; \Gamma \vdash \mathcal{R}[\ M\ ] \div B \implies \exists A\ .\ \Delta; \Gamma \vdash M : A\ \vee\ \Delta; \Gamma \vdash M \rightsquigarrow A\ \vee\ \Delta; \Gamma \vdash M \div A
\end{array}$$

Proof: Each is proved in order, assuming the prior one(s) hold. Individually, we proceed by induction on evaluation contexts. For each form of context, we can

$$\begin{array}{lrl} \text{Eval. Ctxt.} & \mathcal{R} ::= & [\,] \mid \mathcal{R}\; N \mid \overline{V}\; \mathcal{R} \mid \texttt{let box u} = \mathcal{R}\; \texttt{in}\; N \\ & & \mid \texttt{let dia x} = \mathcal{R}\; \texttt{in}\; N \mid \texttt{let dia x} = \texttt{dia}\; \mathcal{R}\; \texttt{in}\; N \\ & & \mid \texttt{let comp x} = \mathcal{R}\; \texttt{in}\; N \mid \texttt{let comp x} = \texttt{comp}\; \mathcal{R}\; \texttt{in}\; N \\ & & \mid \texttt{ref}\; \mathcal{R} \mid \texttt{!}\mathcal{R} \mid \mathcal{R} \texttt{ := } N \mid a \texttt{ := } \mathcal{R} \end{array}$$

$$\begin{array}{ll} \mathit{letbox} & \langle l_1 : \mathcal{R}[\,\texttt{let box u} = \texttt{box}\; M\; \texttt{in}\; N\,]\rangle \Longrightarrow \langle l_2 : M\rangle \triangleleft \langle l_1 : \mathcal{R}[\,[\![l_2/\texttt{u}]\!]N\,]\rangle \\ & \qquad\qquad\qquad\qquad\qquad\qquad (\text{where } l_2\; \texttt{fresh}) \\ \mathit{syncr} & \langle l_2 : \overline{V}\rangle \triangleleft \ldots \langle l_1 : \mathcal{R}[\,l_2\,]\rangle \Longrightarrow \langle l_2 : \overline{V}\rangle \triangleleft \ldots \langle l_1 : \mathcal{R}[\,\overline{V}\,]\rangle \end{array}$$

$$\begin{array}{ll} \mathit{syncl} & \langle l_1 : \mathcal{R}[\,\texttt{let dia x} = \texttt{dia}\,(@l_2)\; \texttt{in}\; N\,]\rangle \triangleleft \ldots [H \vDash \langle l_2 : \overline{V}\rangle] \\ & \Longrightarrow \langle l_1 : @l_3\rangle \triangleleft \ldots [H \vDash \langle l_2 : \overline{V}\rangle, \langle l_3 : \mathcal{R}[\,[\overline{V}/\texttt{x}]N\,]\rangle] \\ & (\text{where } l_3\; \texttt{fresh}) \end{array}$$

$$\begin{array}{ll} \mathit{resolve} & \langle l_1 : \mathcal{R}[\,\texttt{let dia x} = \texttt{dia}\,(@l_2)\; \texttt{in}\; N\,]\rangle \triangleleft \ldots \langle l_2 : @l_3\rangle \\ & \Longrightarrow \langle l_1 : \mathcal{R}[\,\texttt{let dia x} = \texttt{dia}\,(@l_3)\; \texttt{in}\; N\,]\rangle \triangleleft \ldots \langle l_2 : @l_3\rangle \end{array}$$

$$\begin{array}{ll} \mathit{letdia} & \langle l_1 : \mathcal{R}[\,\texttt{let dia x} = \texttt{dia}\,\overline{V}\; \texttt{in}\; N\,]\rangle \\ & \Longrightarrow \langle l_1 : \mathcal{R}[\,[\overline{V}/\texttt{x}]N\,]\rangle \end{array}$$

$$\begin{array}{llll} \mathit{letcomp} & H \vDash \langle l : \mathcal{R}[\,M\,]\rangle & \Longrightarrow H & \vDash \langle l : \mathcal{R}[\,[\overline{V}/\texttt{x}]N\,]\rangle \\ & (\text{where } M = \texttt{let comp x} = \texttt{comp}\,\overline{V}\; \texttt{in}\; N) & & \\ \mathit{alloc} & H \vDash \langle l : \mathcal{R}[\,\texttt{ref}\,\overline{V}\,]\rangle & \Longrightarrow H[a \mapsto \overline{V}] & \vDash \langle l : \mathcal{R}[\,a\,]\rangle \\ & & & (\text{where } a\; \texttt{fresh}) \\ \mathit{get} & H \vDash \langle l : \mathcal{R}[\,\texttt{!}a\,]\rangle & \Longrightarrow H & \vDash \langle l : \mathcal{R}[\,H(a)\,]\rangle \\ \mathit{set} & H \vDash \langle l : \mathcal{R}[\,a \texttt{ := } \overline{V}\,]\rangle & \Longrightarrow H[a \mapsto \overline{V}] & \vDash \langle l : \mathcal{R}[\,\texttt{()}\,]\rangle \end{array}$$

$$\begin{array}{c} \mathit{app}\; \langle l : \mathcal{R}[\,(\lambda \texttt{x}{:}A\,.\,M)\; \overline{V}\,]\rangle \Longrightarrow \langle l : \mathcal{R}[\,[\overline{V}/\texttt{x}]M\,]\rangle \\ \ldots \Longrightarrow \ldots \end{array}$$

Fig. 5. Reduction rules, organized by category. There are spatial reductions associated with the modalities which allow interaction between processes, as well as pure and effectful local reductions.

invert to the relevant typing rule or a subsumption rule (*comp*, *poss*, or *poss'*) applies. □

Under certain conditions, a context $\mathcal{R}[\;]$ can be moved from one environment to another because its constituent subterms are Γ-closed. That is, $\mathcal{R}[\;]$ may be independent of local bindings and mobile, just as a term M encapsulated as `box` M is mobile.

Lemma 3 (Mobile Continuations) *Assume a context $\mathcal{R}[\,]$ such that $\Delta; \Gamma \vdash M \div A \implies \Delta; \Gamma \vdash \mathcal{R}[M] \div B$ (for any M). Then $\Delta, \Delta'; \Gamma' \vdash N \div A \implies \Delta, \Delta'; \Gamma' \vdash \mathcal{R}[N] \div B$ (for any Δ', Γ', and N).*

Proof: by induction on the structure of $\mathcal{R}$. Due to typing, the only possibility is $\mathcal{R} = \texttt{let dia}\,\texttt{x}_1 = \texttt{dia}\,\mathcal{R}'\,\texttt{in}\,N_1$. By inversion on typing we can apply the IH to $\mathcal{R}'$. Also by inversion, $\Delta; \texttt{x}_1 : A_1 \vdash N_1 \div B$. The conclusion follows by Δ-weakening and the $\Diamond E$ typing rule. □

Theorem 1 (Type Preservation) *If $\vdash^c C : \Gamma$ and $C \Longrightarrow C'$ then $\vdash^c C' : \Gamma'$ for some Γ' which extends Γ.*

Proof: by cases on derivation of $C \Longrightarrow C'$, using the definition of $\vdash^c C : \Gamma$, inversion on typing derivations, and substitution properties. In the critical cases where fragments of the program move from one process to another, these mobile terms, values, or contexts remain well-formed via weakening (of Δ and/or Γ). See appendix section 8.1 for selected cases. □

To establish progress, we first enumerate the redices of the semantics and give a decomposition lemma. The category $\langle localredex \rangle$ corresponds to local reduction rules. But note that reducing l or $\texttt{let dia x = dia}\,@l\,\texttt{in}\,N$ requires interaction with other processes.

Definition 1 (Redex and Local Redex)

$$
\begin{array}{rcl}
\langle redex \rangle & ::= & l \ \mid\ \texttt{let dia x = dia}\,@l\,\texttt{in}\,N \ \mid\ \langle localredex \rangle \\
\langle localredex \rangle & ::= & (\lambda \texttt{x}{:}A\,.\,M)\ \overline{V} \ \mid\ \texttt{let box u = box}\,M\,\texttt{in}\,N \\
& \mid & \texttt{let dia x = dia}\,\overline{V}\,\texttt{in}\,N \ \mid\ \texttt{let comp x = comp}\,\overline{V}\,\texttt{in}\,N \\
& \mid & \texttt{ref}\,\overline{V} \ \mid\ !a \ \mid\ a := \overline{V}
\end{array}
$$

Lemma 4 (Decomposition) *Well-formed terms M are either values, or can be decomposed as $\mathcal{R}[N]$ where N is of the form $\langle redex \rangle$.*

$$
\begin{array}{rcl}
\Delta; \Gamma \vdash M : B & \implies & M = V \quad \vee \quad \exists \mathcal{R}\,.\,M = \mathcal{R}[\langle redex \rangle] \\
\Delta; \Gamma \vdash M \sim B & \implies & M = V \quad \vee \quad \exists \mathcal{R}\,.\,M = \mathcal{R}[\langle redex \rangle] \\
\Delta; \Gamma \vdash M \div B & \implies & M = V^* \quad \vee \quad \exists \mathcal{R}\,.\,M = \mathcal{R}[\langle redex \rangle]
\end{array}
$$

Proof: Each is proved in order, assuming the prior one(s) hold. The proof is by induction on typing derivations. If M is a value V (or $M = V^*$), we are done. Otherwise, M is a redex in the empty context (trivial decomposition), or we invert the relevant typing rule and proceed by induction on a typing subderivation to show that M has a non-trivial decomposition. □

Theorem 2 (Progress) *If $\vdash^c C : \Gamma$ then either (1) there exists C' such that $C \Longrightarrow C'$ or (2) C is terminal (all processes contain V^*).*

Proof: Generalize the statement as follows: If $\Delta \vdash^c C : \Gamma$ then (1) $C \Longrightarrow C'$ or (2) C is terminal or (3) C has a process $\langle l : \mathcal{R}[l'] \rangle$ blocked on label $l' :: A \in \Delta$. The proof is by induction on derivations and relies on case analysis and the decomposition lemma. The main progress theorem is an instance where C is closed ($\Delta = \cdot$), so case (3) is vacuous. See appendix section 8.2. □

5 Consequences of Modal Types

Adherence to a theoretical definition of mobility and locality imposes a certain programming discipline. It is difficult to justify our choice of $S4$ as *the singular* theory of mobility and locality for distributed programming. However, we can point out the consequences of adopting $S4$ and identify situations in which such a discipline is advantageous.

In the first place, $S4$ gives us a clean, *static* definition of mobility independent of the dynamic one. It is embodied in the judgment $\Delta; \cdot \vdash M : A$ (and the variation $\Delta; \mathtt{x}{:}A \vdash M \div B$) which requires a mobile term be Γ-closed (or nearly so). Without this condition, the simple copying interpretation of mobility (rules *letbox*, *syncr*, *syncl*) is wrong! To repair this, we could handle store addresses as a special case. But runtime mobility would then *mean* different things for different types, indicating that something is perhaps amiss — that there is a distinction between terms we failed to recognize. The type discipline of $S4$ allows us to recognize this early, during typechecking. For example:

```
(* Incorrect *)                 (* Correct (explicit duplication) *)
let comp xr = comp ref 0 in     let comp xr = comp ref 0 in
                                let comp n = comp !xr in
                                let box u = marsh n in
let dia _ = someplace in        let dia _ = someplace in
                                  let comp xr' = comp ref u in
  let comp y = comp !xr in        let comp y = comp !xr' in
  ...                             ...
```

The incorrect code refers to a local value `xr` inside mobile code, while the correct version explicitly duplicates the reference cell as `xr'` in the new location. The function $\mathtt{marsh} : \mathtt{nat} \to \Box\mathtt{nat}$ is assumed to convert a natural number to its mobile representation. While the example above is trivial, the type theory also accounts for structured data and higher-order functions.

Secondly, there are consequences linked to the particular choice of $S4$ modal logic. The character of $S4$ comes from an assumption that accessibility between worlds is reflexive and transitive. In practice, this means that distance (number of hops) is irrelevant. This is reflected in both the type theory and the semantics of our calculus. For example: $\langle l_1 : V\rangle \triangleleft \langle l_2 : l_1\rangle \triangleleft \langle l_3 : l_2\rangle \Longrightarrow \langle l_1 : V\rangle \triangleleft \langle l_3 : l_1\rangle$ via rule *syncr*. However, there is no requirement that accessibility be symmetric. But why should *direction* matter? Essentially this means that programs execute without introducing 2-way interdependence between processes/locations. This is advantageous when one considers garbage collection of processes or store locations. With cycles among locations allowed, distributed garbage collection becomes quite difficult to implement efficiently.

If, on the other hand, symmetry is explicitly recognized (as it is in the logic $S5$), certain new programming idioms become admissible. [20, 12] Some of these involve a mobility by-remote-reference interpretation.[2] This second kind of mobility introduces interdependencies among locations. So it seems that the type discipline of $S4$, though not the only way to define mobility and locality, describes a simpler kind of distributed programming which is likely to be efficiently implementable.

6 Related Work

There are many prior foundational calculi which have a distributed operational interpretation. Most notably the Pi-calculus [14] and offspring. Pi-calculus processes interact by communicating names over named channels. Locations are thought of as implicit in the connectivity of processes. And names, the only form of resource, do not have a definite location or fixed scope (due to scope-extrusion). Thus locations in the Pi-calculus have no fixed properties or identity. Our approach differs in that we focus on describing locations and their distinguishing properties.

Various proposed calculi have added explicit locations to the Pi-calculus. Examples are the DPI calculus of Hennessy *et. al* [9] and $lsd\pi$ by Ravara, Matos, *et. al* [18]. These calculi allow some channel names to be declared fixed to a location, while others follow the laws of scope extrusion. DPI has a type system that tracks the locality of channel names, and associates each location with a set of resources (names) bound in that location. The Klaim calculus is also based on localized resources (multiple tuple spaces). De Nicola, *et. al* give a type system for Klaim [16] that checks process behaviors against administratively granted capabilities.

The ambient calculus [6] proposed by Cardelli and Gordon is a more radical departure, replacing channels with ambients $n[\ ]$. Ambients are places which may contain other ambients and running processes. They also serve as locations in which fragments of the program exchange messages. Cardelli and Gordon [8, 7] and Caires and Cardelli [2, 3] have developed an ambient logic with modal operators to characterize the location structure and behavior of ambient calculus programs. In their work, accessibility is interpreted as containment of ambients, $\Box\Psi$ requires all sub-locations satisfy Ψ, and $\Diamond\Psi$ requires that some sub-location satisfy Ψ. As with names in the Pi-calculus, untyped ambients have no fixed locality or scope; in the absence of a specification, nested ambients may move freely in and out of other ambients in response to actions of the running program. Cardelli, Ghelli, and Gordon also developed a static type system for ambients [4, 5] which restricts ambient mobility. But their notion of mobility is quite different from the one we derived from logical necessity. An ambient characterized as immobile may be nested inside a mobile one, for example.

Modal logics should be referred to in the plural, because there are several different ways to define the meaning of $\Box A$ and $\Diamond A$. Following a similar intuition,

[2] The detailed reasons behind this claim are beyond the scope of this paper, but the interested reader should compare our interpretation of values of type $\Diamond A$ to those in an $S5$-based calculus such as [20].

others have derived distributed calculi from $S5$ or $S5$-like hybrid logics. $S5$ is distinguished from $S4$ by the assumption that accessibility between worlds is symmetric, in addition to reflexive and transitive. The Lambda 5 calculus of Murphy, Crary, Harper, and Pfenning [20, 21] is derived from pure $S5$. And Jia and Walker's λ_{rpc} language [12, 11] is based on a $S5$-like hybrid logic with spatial types $A@w$ and $n[A]$ (absolute and relative locations) in addition to the pure modalities $\Box A$ and $\Diamond A$. Both type theories are based in a formalism with explicit worlds; the programmer specifies directly where all fragments of the program are evaluated. This is qualitatively different than our $S4$ calculus, in which boxed terms are evaluated at any indefinite location. In some ways, $S5$ allows a programmer to do more than $S4$. For the most part, this is explained by the axiom schemas (5) $\Diamond A \to \Box\Diamond A$ and (5') $\Diamond\Box A \to \Box A$. Axiom (5) represents the ability to make (references to) remote terms mobile, and (5') the ability to return a mobile term which happens to be remote. Both can be given a safe and sensible semantics, but there are costs to be weighed. For example, supporting (5) complicates storage management in a distributed implementation, since mobile remote references to arbitrary local values can be created. The $S4$ type theory defines mobility and locality in a minimal way without invoking this capability.

In [1] Borghuis and Feijs present a language based on a single $\Box^o$ modality. However, their operational interpretation of this modality is not based on the spatial interpretation of $\Box$ that we adopt. Rather $\Box^o(A \to B)$ represents location o's knowledge of how to transform a value of type A to one of type B. The calculus allows composing services and applying them to values, which are all assumed to be mobile.

7 Conclusions

While $S4$ does not lead to the most computationally powerful distributed language, it has a relatively simple programming model and type system. Furthermore, the discipline imposed by $S4$ could be advantageous because it simplifies the runtime support for marshalling and distributed garbage collection. It also encourages the programmer to work locally (as noted by Jagannathan [10])

In this instance, we treated stores H as the fundamental distinguishing property of locations, but the problem of local resources is more general. Store addresses serve as canonical example localized entities, but the type system is independent of this choice. Besides mutable references, file handles, or other pointers to OS data structures, it is not so clear what other classes of localized value there might be. Certainly mobility (by-copying) is not tenable in situations where identity of an object must be preserved. But one might choose to fix certain resources to locations for reasons of efficiency, privacy, or security.

In future work, we plan to continue this abstract investigation of distributed computation. Various second-order extensions of the type theory are under investigation. Hopefully this will shed new light on how abstract datatypes should behave in a distributed computation, and the meaning of abstraction-safety in that context. A prototype implementation is also planned, to determine where and to what degree the execution model of $S4$-based programs leads to savings in cost and complexity.

References

1. Tijn Borghuis and Loe Feijs. A constructive logic for services and information flow in computer networks. *The Computer Journal*, 43(4), 2000.
2. Luís Caires and Luca Cardelli. A spatial logic for concurrency (part I). In *Theoretical Aspects of Computer Software (TACS)*, volume 2215 of *LNCS*, pages 1–37. Springer, October 2001.
3. Luís Caires and Luca Cardelli. A spatial logic for concurrency (part II). In *CONCUR*, volume 2421 of *LNCS*, pages 209–225. Springer, August 2002.
4. Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Mobility types for mobile ambients. In Jiri Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languagese and Programming, 26th International Colloquium (ICALP)*, volume 1644 of *LNCS*, pages 230–239. Springer, 1999.
5. Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Mobility types for mobile ambients. Technical Report MSR-TR-99-32, Microsoft, June 1999.
6. Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures (FOSSACS)*, volume 1378 of *LNCS*, pages 140–155. Springer-Verlag, 1998.
7. Luca Cardelli and Andrew D. Gordon. Logical properties of name restriction. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications*, volume 46-60 of *LNCS*, pages 46–60. Springer, May 2001.
8. Luca Cardelli and Andrew D. Gordon. Ambient logic. Technical report, Microsoft, 2002.
9. Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.
10. Suresh Jagannathan. Continuation-based transformations for coordination languages. *Theoretical Computer Science*, 240(1):117–146, 2000.
11. Limin Jia and David Walker. Modal proofs as distributed programs. Technical Report TR-671-03, Princeton University, August 2003.
12. Limin Jia and David Walker. Modal proofs as distributed programs. In *European Symposium on Programming Languages*, April 2004.
13. D. A. Kranz, R. H. Halstead, Jr., and E. Mohr. Mul-T: a high-performance parallel lisp. In *Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation*, pages 81–90. ACM Press, 1989.
14. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes (I & II). *Information and Computation*, 100(1):1–40 & 41–77, 1992.
15. Jonathan Moody. Modal logic as a basis for distributed computation. Technical Report CMU-CS-03-194, Carnegie Mellon University, October 2003.
16. R. De Nicola, G. Ferrari, R. Pugliese, and B. Venneri. Types for access control. *Theoretical Computer Science*, 240(1):215–254, 2000. Klaim and tuple-spaces.
17. Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, August 2001.
18. António Ravara, Ana G. Matos, Vasco T. Vasconcelos, and Luís Lopes. Lexically scoped distribution: what you see is what you get. In *Foundations of Global Computing*. Elsevier, 2003.

19. Alex K. Simpson. *Proof Theory and Semantics of Intuitionistic Modal Logic*. PhD thesis, University of Edinburgh, 1994.
20. Tom Murphy VII, Karl Crary, Robert Harper, and Frank Pfenning. A symmetric modal lambda calculus for distributed computing. In *LICS (to appear)*, 2004.
21. Tom Murphy VII, Karl Crary, Robert Harper, and Frank Pfenning. A symmetric modal lambda calculus for distributed computing. Technical Report CMU-CS-04-105, Carnegie Mellon University, 2004.

8 Appendix

8.1 Type Preservation: selected cases

Case: *syncr*

$$\langle l_2 : \overline{V} \rangle \triangleleft \ldots \langle l_1 : \mathcal{R}[\, l_2 \,] \rangle \Longrightarrow \langle l_2 : \overline{V} \rangle \triangleleft \ldots \langle l_1 : \mathcal{R}[\, \overline{V} \,] \rangle$$

Let $\Delta = \Delta_2, l_2 :: A, \Delta_1$	
Process l_1: $\Delta; \Gamma \vdash \mathcal{R}[\, l_2 \,] \div B$	Assumption
$\Delta; \Gamma \vdash l_2 : A$	Context Inversion
Process l_2: $\Delta_2; \cdot \vdash \overline{V} : A$	Assumption
$\Delta; \Gamma \vdash \overline{V} : A$	Weakening $(\Delta_2; \cdot \subseteq \Delta; \Gamma)$
Process l_1: $\Delta; \Gamma \vdash \mathcal{R}[\, \overline{V} \,] \div B$	Typing

Case: *letbox*

$$\langle l_1 : \mathcal{R}[\, \mathtt{let\,box\,u = box}\, M \,\mathtt{in}\, N \,] \rangle \Longrightarrow \langle l_2 : M \rangle \triangleleft \langle l_1 : \mathcal{R}[\, [\![l_2/\mathtt{u}]\!] N \,] \rangle$$
$$\text{(where } l_2 \text{ } \mathtt{fresh}\text{)}$$

Process l_1: $\Delta; \Gamma \vdash \mathcal{R}[\, \mathtt{let\,box\,u = box}\, M \,\mathtt{in}\, N \,] \div C$	Assumption
$\Delta; \Gamma \vdash \mathtt{let\,box\,u = box}\, M \,\mathtt{in}\, N \div B$	Context Inversion
(1) $\Delta; \cdot \vdash M : A$	Inversion
$\Delta, \mathtt{u} :: A; \Gamma \vdash N \div B$	Inversion
Let $\Delta' = \Delta, l_2 :: A$	
Process l_2: $\Delta; \cdot \vdash M : A$	by (1)
$\Delta', \mathtt{u} :: A; \Gamma \vdash N \div B$	Weakening $(\Delta \subseteq \Delta')$
$\Delta'; \Gamma \vdash [\![l_2/\mathtt{u}]\!] N \div B$	Substitution
Process l_1: $\Delta'; \Gamma \vdash \mathcal{R}[\, [\![l_2/\mathtt{u}]\!] N \,] \div C$	Typing

Case: *syncl*

$$\begin{aligned} & \langle l_1 : \mathcal{R}[\, \mathtt{let\,dia\,x = dia}\, (@l_2) \,\mathtt{in}\, N \,] \rangle \triangleleft \ldots [H \vDash \langle l_2 : \overline{V} \rangle] \\ \Longrightarrow & \langle l_1 : @l_3 \rangle \triangleleft \ldots [H \vDash \langle l_2 : \overline{V} \rangle, \langle l_3 : \mathcal{R}[\, [\overline{V}/\mathtt{x}] N \,] \rangle] \\ & \text{(where } l_3 \text{ } \mathtt{fresh}\text{)} \end{aligned}$$

Let $\Gamma_1 = \Gamma, @l_2 : A, \Gamma_2$	
Let $\Delta_2 = \Delta_1, \Delta$	
Process l_1: $\Delta_1; \Gamma_1, \Gamma^{H_1} \vdash \mathcal{R}[\, \mathtt{let\,dia\,x = dia}\, (@l_2) \,\mathtt{in}\, N \,] \div C$	Assumption
Process l_2: $\Delta_2; \Gamma_2, \Gamma^{H_2} \vdash \overline{V} \div A$	Assumption
$\Delta_1; \Gamma_1, \Gamma^{H_1} \vdash \mathtt{let\,dia\,x = dia}\, (@l_2) \,\mathtt{in}\, N \div B$	Context Inversion
$\Delta_1; \Gamma_1, \Gamma^{H_1} \vdash @l_2 \div A$	Inversion
$\Delta_1; \mathtt{x} : A \vdash N \div B$	Inversion
Let $\Gamma_1' = \Gamma, @l_2 : A, @l_3 : C, \Gamma_2$	
Process l_1: $\Delta_1; \Gamma_1', \Gamma^{H_1} \vdash @l_3 \div C$	Rule *loc*
Process l_2: $\Delta_2; @l_3 : C, \Gamma_2, \Gamma^{H_2} \vdash \overline{V} \div A$	Weakening $(\Gamma_2, \Gamma^{H_2} \subseteq @l_3 : C, \Gamma_2, \Gamma^{H_2})$

$\Delta_2; \Gamma_2, \Gamma^{H_2}, \mathtt{x} : A \vdash N \div B$	Weakening ($\Delta_1 \subseteq \Delta_2$)
	($\mathtt{x} : A \subseteq \Gamma_2, \Gamma^{H_2}, \mathtt{x} : A$)
$\Delta_2; \Gamma_2, \Gamma^{H_2} \vdash [\overline{V}/\mathtt{x}]N \div B$	Substitution
Process l_3 : $\Delta_2; \Gamma_2, \Gamma^{H_2} \vdash \mathcal{R}[\,[\overline{V}/\mathtt{x}]N\,] \div C$	Mobile Continuation

Case: *resolve*

$$\begin{aligned} &\langle l_1 : \mathcal{R}[\, \mathtt{let\,dia\,x = dia}\,(@l_2)\,\mathtt{in}\,N\,]\rangle \triangleleft \ldots \langle l_2 : @l_3\rangle \\ \Longrightarrow &\langle l_1 : \mathcal{R}[\, \mathtt{let\,dia\,x = dia}\,(@l_3)\,\mathtt{in}\,N\,]\rangle \triangleleft \ldots \langle l_2 : @l_3\rangle \end{aligned}$$

Let $\Gamma_1 = \Gamma, @l_2 : A, \Gamma_2$	
Let $\Delta_2 = \Delta_1, \Delta$	
Process l_1: $\Delta_1; \Gamma_1, \Gamma^{H_1} \vdash \mathcal{R}[\, \mathtt{let\,dia\,x = dia}\,(@l_2)\,\mathtt{in}\,N\,] \div C$	Assumption
Process l_2 : $\Delta_2; \Gamma_2, \Gamma^{H_2} \vdash @l_3 \div A$	Assumption
$\Gamma_2 = \ldots, @l_3 : A, \ldots$	Inversion
$\Delta_1; \Gamma_1, \Gamma^{H_1} \vdash \mathtt{let\,dia\,x = dia}\,(@l_2)\,\mathtt{in}\,N \div B$	Context Inversion
$\Delta_1; \Gamma_1, \Gamma^{H_1} \vdash @l_3 \div A$	Rule *loc*
$\Delta_1; \Gamma, \Gamma^{H_1} \vdash \mathtt{let\,dia\,x = dia}\,(@l_3)\,\mathtt{in}\,N \div B$	Rules $\Diamond I$,$\Diamond E$
Process l_1: $\Delta_1; \Gamma, \Gamma^{H_1} \vdash \mathcal{R}[\, \mathtt{let\,dia\,x = dia}\,(@l_3)\,\mathtt{in}\,N\,] \div C$	Typing

8.2 Progress Theorem

Case: *none*

$$\frac{}{\Delta \vdash^c \ \cdot \ : \ \cdot}\ none$$

($C = \cdot$) is terminal	Definition

Case: *indef*

$$\frac{\Delta; \cdot \vdash M : A \quad \overset{\mathcal{D}}{\overline{\Delta, l :: A \vdash^c C \ : \ \Gamma}}}{\Delta \vdash^c \langle l : M\rangle \triangleleft C \ : \ \Gamma}\ indef$$

$\Delta; \cdot \vdash^c M : A$	Assumption
$\Delta, l :: A \vdash^c C : \Gamma$	Assumption
Decomposition of M:	Decomp. Lemma
(a) $M = V$, or	
(b) $M = \mathcal{R}[\,\langle redex\rangle\,]$	
Progress for $C : \Gamma$	IH($\mathcal{D}$)
(1) $C \Longrightarrow C'$, or	
(2) C is terminal, or	
(3) exists Δ-blocked $\langle l' : \mathcal{R}[\,l''\,]\rangle$ in C	
If (a) then:	
If (1) then: $\langle l : V\rangle \triangleleft C \Longrightarrow \langle l : V\rangle \triangleleft C'$	Immediate
If (2) then: $\langle l : V\rangle \triangleleft C$ is terminal	Definition

If (3) and $l = l''$ then: $\langle l : V \rangle \triangleleft C \Longrightarrow \langle l : V \rangle \triangleleft C'$ — Rule *syncr*
If (3) and $l \neq l''$ then: $l'' :: A \in \Delta$ (Δ-blocked process) — Typing Inv.

If (b) and $M = \mathcal{R}[\, l''' \,]$ then: $l''' :: A \in \Delta$ (Δ-blocked process) — Typing Inv.
If (b) and $M = \mathcal{R}[\, \mathtt{let\,dia\,x = dia}\,@l_4\,\mathtt{in}\,N \,]$ then: Contradiction — Typing
If (b) and $M = \mathcal{R}[\, \langle \mathit{localredex} \rangle \,]$ then:
$\langle l : M \rangle \triangleleft C \Longrightarrow \langle l : M' \rangle \triangleleft C$ — Local Redex

Case: *location*

$$\dfrac{\dfrac{\mathcal{D}}{\Delta \vdash^c C : \Gamma} \quad \Delta \vdash^s H : \Gamma^H \quad \Delta; \Gamma, \Gamma^H \vdash^c P : \Gamma'}{\Delta \vdash^c [H \vDash P] \triangleleft C \;:\; \Gamma', \Gamma} \; \mathit{location}$$

$\Delta \vdash^c C : \Gamma$ — Assumption
$\Delta; \Gamma, \Gamma^H \vdash^c P : \Gamma'$ — Assumption
Decomposition of $\langle l_i : M_i \rangle \in P$: — Decomp. Lemma
(a) all $M_i = V_i$, or
(b) exists $M_i = \mathcal{R}[\, \langle \mathit{redex} \rangle \,]$
Progress for $C : \Gamma$ — IH($\mathcal{D}$)
(1) $C \Longrightarrow C'$, or
(2) C is terminal, or
(3) exists Δ-blocked $\langle l' : \mathcal{R}[\, l'' \,] \rangle$ in C — Typing Inv.

If (1) then: $[H \vDash P] \triangleleft C \Longrightarrow [H \vDash P] \triangleleft C'$ — Immediate
If (2) and (a) then: $[H \vDash P] \triangleleft C$ is terminal — Definition
If (2) and (b) then:
If $M_i = \mathcal{R}[\, l''' \,]$ then: $l''' :: A \in \Delta$ (Δ-blocked process) — Typing Inv.
If $M_i = \mathcal{R}[\, \mathtt{let\,dia\,x = dia}\,@l_4\,\mathtt{in}\,N \,]$ then:
$[H \vDash \langle l_i : M_i \rangle] \triangleleft C \Longrightarrow [H \vDash \langle l_i : M_i' \rangle] \triangleleft C'$ — Rule *syncl* or *resolve*
Otherwise $M_i = \mathcal{R}[\, \langle \mathit{localredex} \rangle \,]$ then:
$[H \vDash \langle l_i : M_i \rangle] \triangleleft C \Longrightarrow [H' \vDash \langle l_i : M_i' \rangle] \triangleleft C$ — Local Redex
If (3) then: $l'' :: A \in \Delta$ (Δ-blocked process) — Typing Inv.

8.3 Fixpoints

There are two natural forms of fixpoint corresponding to the distinction between variables ($\mathtt{u} :: A$) and ($\mathtt{x} : A$). We refer to $\mathtt{mfix}\,(\mathtt{u}{::}A)\,.\,M$ as mobile fixpoint, and $\mathtt{fix}\,(\mathtt{x}{:}A)\,.\,M$ as local fixpoint. The operational semantics is given in the conventional way, with substitution used to perform unrolling.

$$\frac{\Delta, \mathtt{u} :: A; \cdot \vdash M : A}{\Delta; \Gamma \vdash \mathtt{mfix}\,(\mathtt{u}{::}A)\,.\,M : A}\ mfix \qquad \frac{\Delta; \Gamma, \mathtt{x} : A \vdash M : A}{\Delta; \Gamma \vdash \mathtt{fix}\,(\mathtt{x}{:}A)\,.\,M : A}\ fix$$

$$\begin{array}{ll} unroll_m & \langle r : \mathcal{R}[\,\mathtt{mfix}\,(\mathtt{u}{::}A)\,.\,M\,]\rangle \Longrightarrow \langle r : \mathcal{R}[\,[\![\mathtt{mfix}\,(\mathtt{u}{::}A)\,.\,M/\mathtt{u}]\!]M\,]\rangle \\ unroll & \langle r : \mathcal{R}[\,\mathtt{fix}\,(\mathtt{x}{:}A)\,.\,M\,]\rangle \Longrightarrow \langle r : \mathcal{R}[\,[\mathtt{fix}\,(\mathtt{x}{:}A)\,.\,M/\mathtt{x}]M\,]\rangle \end{array}$$

One can encode recursive computations involving effects or remote locations using $\mathtt{fix}\,(\mathtt{x}{:}\bigcirc A)\,.\,M$ or $\mathtt{mfix}\,(\mathtt{u}{::}\Diamond A)\,.\,M$. The same encoding strategy fails for mobile fixpoints encoded as $\mathtt{fix}\,(\mathtt{x}{:}\Box A)\,.\,M$ because the encoding does not have the right behavior under reduction.

8.4 Configurations and Globally Accessible Resources

Ordinarily, one does not think of resource handles or pointers as themselves being localized. After all, the label $@l$ is separated from the underlying term, and there is no obvious operational reason why it must behave differently than a label of the form l. There is, however, a subtle logical explanation. Assuming *only* reflexive and transitive accessibility ($S4$), a resource accessible at one world (location) may become inaccessible upon shifting perspective to a second world. Without stronger assumptions about the topology of worlds, treating labels $@l$ as freely mobile terms is unsound. This presents difficulties when we try to explain binding of programs to external resources of type $\Box\Diamond A$ that are both mobile and remote.

The difficulty is as follows: Our simple notion of a process configuration C cannot reflect all forms of dependency between processes executing in different locations. There is only enough expressivity to describe an antisymmetric accessibility relation (*i.e.* some ordering of locations). While source programs execute safely in this kind of environment, we cannot immediately support certain natural extensions, mobile handles to remote resources in particular. No value of the form $\mathtt{box}\,(\mathtt{dia}\,@l)$ can be well-typed. To rectify this, we assume certain processes denoting global resources are accessible from all others; their labels (written $\overline{@}l$) are then treated as part of the mobile context ($\overline{@}l :: A) \in \Delta$. Values $\mathtt{box}\,(\mathtt{dia}\,\overline{@}l) : \Box\Diamond A$ denoting a globally accessible resource are then properly accounted for. $S4$ does not *mandate* antisymmetry, and source programs don't conatin $\overline{@}l$, so the logical character of the calculus is not disturbed. We do not pursue this extension here because it complicates the structure of process configurations without shedding much light on new phenomena.