Modal Logic: Implications for Design of a Language for Distributed Computation

Jonathan Moody
(with Frank Pfenning)

Department of Computer Science
Carnegie Mellon University
Talk Outline

- Concepts of modal logic
- Intuitionistic formalism
- Distributed programming
- Conclusions
Modal logic(s) distinguish **modes** of truth.

For the generalized modal logic (S4) these modes of truth are explained by referring to (abstract) “worlds”:

- Truth in all (accessible) worlds.
- Truth in **this** world.
- Truth in **some** (accessible) world.
Concepts: A Kripke model

Worlds of the Kripke structure

W1 W2

W3 W4
Concepts: A Kripke model

Accessibility between worlds
Concepts: A Kripke model

Primitive assumptions
From the perspective of world $W_1$ ...
A and C are true here ($W_1$)
By introducing new forms of proposition, we can make statements about other worlds.

\[ \Box A \] — \( A \) true in all accessible worlds.

\[ \Diamond A \] — \( A \) true in some accessible world.
Concepts: A Kripke model

\( \Box C \) true at \( W_1 \) because...

\( C \) true at \( W_1, W_2, W_3, W_4 \) (refl. & trans.)
Diamond $\diamondsuit A$ true at $W_1$ because...

Bullet $\bullet A$ true at $W_1$ (reflexivity)
Concepts: A Kripke model

\[ \Diamond B \text{ true at } W_1 \text{ because...} \]
\[ B \text{ true at } W_3 \]
Concepts: A Kripke model

Modal Logic: Implications for Design of a Language for Distributed Computation – p.6/53

\[ \Diamond D \text{ true at } W_1 \text{ because...} \]

\[ D \text{ true at } W_4 \text{ (transitivity)} \]
But what are the “worlds” we refer to?

It is quite possible to remain abstract, but for applications it helps to have a class of worlds in mind.

- **Temporal properties**
  (worlds are moments, ordering determines accessibility)

- **Stateful computation**
  (worlds are states, effects determine accessibility)
For **distributed computation**, we adopt a *spatial* interpretation of worlds.
For distributed computation, we adopt a spatial interpretation of worlds.

Worlds are:
- where program fragments reside,
- where these fragments are well-typed,
- and hence where evaluation may happen.
For **distributed computation**, we adopt a *spatial* interpretation of worlds.

Worlds are:
- where program fragments **reside**,  
- where these fragments are **well-typed**,  
- and hence where **evaluation** may happen.

Accessibility (hypothesis):
- the capability to **move** program fragments between worlds.
Talk Outline

- Intuitionistic formalism
  - Judgements and propositions
  - Language of proof terms
  - Operational reading
  - Properties
The meaning of propositions in the intuitionistic formulation is consistent with those of the classical formulation.

However, we are now in an intuitionistic setting...
- We focus on the form of proofs,
- not truth relative to a particular model.
Judgements and propositions

- Judgements formalize the **modes** of truth:
  - $A_{\text{valid}}$ (true everywhere)
  - $A_{\text{true}}$ (true here)
  - $A_{\text{poss}}$ (true somewhere)

- Propositions of the logic remain the same:
  - $\Box A$ (internalizes $A_{\text{valid}}$)
  - $\Diamond A$ (internalizes $A_{\text{poss}}$)
  - $A \rightarrow B$ (internalizes entailment)
Hypothetical judgements are represented as:

\[ \Delta; \Gamma \vdash A \text{true} \] (or \[ \Delta; \Gamma \vdash A \text{poss} \])

- \( \Delta \) holds “global” hypotheses (\( A \text{valid} \))
- \( \Gamma \) holds “local” hypotheses (\( A \text{true} \))
Language of proof terms

Via a Curry-Howard isomorphism we can:

\[ \Delta; \Gamma \vdash \text{true} \quad \text{to} \quad \Delta; \Gamma \vdash M : A \]

\[ \Delta; \Gamma \vdash \text{poss} \quad \text{to} \quad \Delta; \Gamma \vdash E \div A \]

Terms and expressions are simultaneously proof objects and programs.
Language of proof terms

Quick overview of syntax (more depth later):

Term \( M, N \) ::= \( x \) \mid u \mid \lambda x : A. M \mid M \ N \mid \text{box} \ M \mid \text{dia} \ E \mid \text{let box} \ u = M \ \text{in} \ N

Expr. \( E, F \) ::= \{ M \} \mid \text{let box} \ u = M \ \text{in} \ F \mid \text{let dia} \ x = M \ \text{in} \ F
Operational reading

- Principles of the operational semantics:
  - We may interpret terms/expressions only in a location where they “make sense”, that is, where they establish $A_{\text{true}}$
  - Evaluation at separate “worlds” proceeds concurrently.
Operational reading: Notation

- Process notation $\langle r : M \rangle$
  - A process labeled $r$ containing term $M$.
  - Each process represents a (possibly) distinct “world”.

- Transition relation $C \Rightarrow C'$
  - $C, C'$ are process configurations (collections of processes).
Evaluation context notation $\mathcal{R}[M]$

(Term) values of the language:

\[
\lambda x : A. M \text{ tvalue} \quad \Box M \text{ tvalue}
\]

\[
\text{dia} E \text{ tvalue} \quad r \text{ tvalue}
\]

Note that language of terms is extended:
- “Result” label $r$ is considered a term value.
- Allows processes to refer to one another.
Operational reading: \( A \rightarrow B \)

"Local" variables and \( \rightarrow \) intro/elim:

\[
\Delta; \Gamma, x : A, \Gamma' \vdash x : A \quad \text{hyp}
\]

\[
\frac{\Delta; \Gamma, x : A \vdash M : B}{\Delta; \Gamma \vdash \lambda x : A . M : A \rightarrow B} \quad \rightarrow I
\]

\[
\frac{\Delta; \Gamma \vdash M : A \rightarrow B \quad \Delta; \Gamma \vdash N : A}{\Delta; \Gamma \vdash MN : B} \quad \rightarrow E
\]
Operational reading: \( A \rightarrow B \)

Local reduction step:

\[
\frac{\Delta; \Gamma, x : A \vdash M' : B}{\Delta; \Gamma \vdash \lambda x : A . M' : A \rightarrow B \rightarrow I} \quad \frac{\Delta; \Gamma \vdash N : A}{\Delta; \Gamma \vdash (\lambda x : A . M') \ N : B} \quad \rightarrow E
\]

\[
V_1 = (\lambda x : A . M') \quad V_2 \text{ tvalue}
\]

\[
\langle r : R[V_1 \ V_2] \rangle \Rightarrow \langle r : R[[V_2/x]M'] \rangle
\]

app
Operational reading: □A

“Global” variables and □ intro/elim:

\[
\Delta, u :: A, \Delta'; \Gamma \vdash u : A \quad \text{hyp}^*
\]

\[
\Delta; \cdot \vdash M : A
\]

\[
\Delta; \Gamma \vdash \text{box} M : \Box A \quad \Box I
\]

\[
\Delta; \Gamma \vdash M : \Box A \quad \Delta, u :: A; \Gamma \vdash N : B
\]

\[
\Delta; \Gamma \vdash \text{let box} u = M \text{ in } N : B \quad \Box E
\]
Local reduction step:

\[ \Delta; \cdot \vdash M : A \quad \Delta; \Gamma \vdash \text{box } M : \Box A \quad \Box I \quad \Delta, u :: A; \Gamma \vdash N : B \quad \Box E \]

\[ \Delta; \Gamma \vdash \text{let } \text{box } u = \text{box } M \text{ in } N : B \]

\[ V = \text{box } M \quad r_2 \text{ fresh} \]

\[ \langle r_1 : \mathcal{R}[\text{let } \text{box } u = V \text{ in } N] \rangle \]

\[ \Rightarrow \langle r_2 : M \rangle; \langle r_1 : \mathcal{R}[[[r_2/u]]N] \rangle \]
Operational reading: \( \square A \)

- Synchronization on “result” labels \((r)\)

\[
\frac{V \text{ tvalue}}{\langle r_2 : V \rangle; \langle r_1 : R[r_2] \rangle \Rightarrow \langle r_2 : V \rangle; \langle r_1 : R[V] \rangle} \]

- Immediate synch. is not required \((r \text{ tvalue})\).
  - We have a choice between synchronization \((R[r])\) or the “usual” reduction step.
  - Concurrency is a **secondary effect** of the spatial interpretation, **not logically essential**.
Now considering the **expression fragment** of the language...

- Having $E \vdash A$ means that $E$ “makes sense” somewhere (but not necessarily “here”).
- We may not interpret expressions $E$ until they are placed in the proper context.
It is convenient to introduce expression variants of:

- Processes: $\langle l : E \rangle$ and $\langle l_1 : l_2 \rangle$
- Evaluation contexts: $S[M]$ and $S[E]$
- Expression values:

$$V \text{ tvalue}$$

$$\{V\} \text{ evalue}$$
Operational reading: $\diamond A$

- **Relationship between truth and possibility:**

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

- **“Global” variables bound in expressions:**

$$
\Delta; \Gamma \vdash M : \Box A \\
\Delta, u :: A; \Gamma \vdash F \div B
\implies
\Delta; \Gamma \vdash \text{let } \text{box } u = M \text{ in } F \div B
$$

\[\Box E_p\]
Operational reading: ◊A

◊ introduction and elimination:

\[ \frac{\Delta; \Gamma \vdash E \div A}{\Delta; \Gamma \vdash \text{dia } E : ◊A} \quad \diamond I \]

\[ \frac{\Delta; \Gamma \vdash M : ◊A \quad \Delta; x : A \vdash F \div B}{\Delta; \Gamma \vdash \text{let } \text{dia } x = M \text{ in } F \div B} \quad \diamond E \]
Local reduction step:

\[
\Delta; \Gamma \vdash E \div A \\
\Delta; \Gamma \vdash \text{dia } E : \diamond A \\
\Delta; \Gamma \vdash \text{let } \text{dia } x = \text{dia } E \text{ in } F \div B \\
\Delta; \Gamma \vdash F \div B
\]

\[
V = \text{dia } E \\
\text{l}_2 \text{ fresh} \\
\left\langle \text{l}_1 : \text{let } \text{dia } x = V \text{ in } F \right \rangle \\
\Rightarrow \left\langle \text{l}_2 : \left\langle \left\langle E/x \right\rangle F \right \rangle; \left\langle \text{l}_1 : \text{l}_2 \right \rangle \right
\]

Note: location \text{l}_2 is not arbitrary.
Language of proof terms

In summary:

Term $M, N ::= x \mid u$

$\mid \lambda x : A. M \mid M \ N$

$\mid \text{box } M \mid \text{dia } E$

$\mid \text{let } \text{box } u = M \text{ in } N$

Expr. $E, F ::= \{ M \} \mid \text{let } \text{box } u = M \text{ in } F$

$\mid \text{let } \text{dia } x = M \text{ in } F$
Properties

Typing for process configurations ($\vdash C \ C : \Lambda$)

Conf. Typing $\Lambda ::= \cdot \mid \Lambda, r :: A \mid \Lambda, l \vdash A$

- “Result” labels $r :: A$ (logical validity).
- “Location” labels $l \vdash A$ (logical possibility).
Properties

- **Type preservation** holds for \( C \Rightarrow C'' \):
  - If \( \vdash_C C : \Lambda \) and \( C \Rightarrow C'' \)
  - then \( \vdash_C C'' : \Lambda' \) (where \( \Lambda' \supseteq \Lambda \)).

Proof depends on:
- Various substitution properties (from previous work).
Terminal processes:

\[
\begin{align*}
V_{\text{tvalue}} & \quad \langle r:V \rangle_{\text{terminal}} \\
V_{\text{evalue}} & \quad \langle l:V \rangle_{\text{terminal}} \quad \langle l_1:l_2 \rangle_{\text{terminal}}
\end{align*}
\]
Progress holds for well-formed config. $C$:

- if $\vdash_C C : \Lambda$
- then $C \Rightarrow C''$ or $C$ terminal

Proof depends on:

- $\vdash_C C : \Lambda$ requires labels $r$ to be non-cyclic (similar to heap typing).
- Thus $\vdash_C C : \Lambda$ imposes an ordering on processes in $C$ which permits induction.
Confluence (plausible but not proved)

\( C \Rightarrow C' \) permits non-deterministic, interleaved evaluation, but the results are always the “same” (modulo synchronization).

Essentially there are only two forms of choice:
- Which process to focus on.
- Performing synchronization or the “usual” reduction step.
Talk Outline

- Distributed programming
  - Marshalling
  - The logical solution
  - Examples
Distributed Programming

From the perspective of ConCert, remote evaluation is the key.

To support remote evaluation, we need mechanisms for:

- Code distribution
- Parameter distribution
Code distribution:
- Pre-distribute code (RPC, Globus).
- Distribute at runtime (Concert).
- In either case, it is assumed that code is “global” (ignoring binary compatibility).

Parameter distribution:
- Marshalling some things is tricky.
- Hence implementors usually make a marshallable/non-marshallable distinction.
The marshallable/non-marshallable distinction is critical:
- Semantic anomalies if you get it wrong.
- Code mobility depends on parameter mobility.
The key is to recognize that some things are inherently localized.

Need to ask ourselves: Which objects can sensibly be transferred between locations?
The key is to recognize that some things are inherently localized.

Need to ask ourselves: Which objects can sensibly be transferred between locations?
- integers, strings, (etc.)?
The key is to recognize that some things are inherently localized.

Need to ask ourselves: Which objects can sensibly be transferred between locations?
- integers, strings, (etc.)? yes.
The key is to recognize that some things are inherently localized.

Need to ask ourselves: Which objects can sensibly be transferred between locations?
- integers, strings, (etc.)? yes.
- functions?
Marshalling

- The key is to recognize that some things are inherently localized.
- Need to ask ourselves: Which objects can sensibly be transferred between locations?
  - integers, strings, (etc.)? yes.
  - functions? depends on env. of closure.
The key is to recognize that some things are inherently localized.

Need to ask ourselves: Which objects can sensibly be transferred between locations?

- integers, strings, (etc.)? yes.
- functions? depends on env. of closure.
- heap addresses?
The key is to recognize that some things are inherently localized.

Need to ask ourselves: Which objects can *sensibly* be transferred between locations?

- integers, strings, (etc.)? *yes.*
- functions? *depends on env. of closure.*
- heap addresses? *no.*
The key is to recognize that some things are inherently localized.

Need to ask ourselves: Which objects can sensibly be transferred between locations?
- integers, strings, (etc.)? yes.
- functions? depends on env. of closure.
- heap addresses? no.
- file handles?
The key is to recognize that some things are inherently localized.

Need to ask ourselves: Which objects can sensibly be transferred between locations?
- integers, strings, (etc.)? yes.
- functions? depends on env. of closure.
- heap addresses? no.
- file handles? no.
The logical solution

The language of modal logic reflects (and resolves) these issues!
The logical solution

- The language of modal logic reflects (and resolves) these issues!
- The expressions $E \div A$ of our language have the desired properties!
The logical solution

- The language of modal logic reflects (and resolves) these issues!
- The expressions $E \div A$ of our language have the desired properties!
- Benefits of the logical approach:
  - We get a clean type-analysis framework automatically.
  - Suggests two forms of code mobility, one of which is not so obvious.
The logical solution

Typing judgement reflects marshallable/non-marshallable distinction:

\[ \Delta; \cdot \vdash M : A \quad \frac{\Delta; \Delta; \Gamma \vdash \Box M : \Box A}{\Box I} \]

\[ \Delta; \Gamma \vdash M : \Diamond A \quad \frac{\Delta; \Gamma \vdash x : A \vdash F \div B}{\Delta; \Gamma \vdash \text{let dia} \ x = M \ 	ext{in} \ F \div B \quad \Diamond E} \]

- \( \Box I \) permits only globally valid parameters.
- \( \Diamond E \) permits param. from a single location.
The logical solution

Moreover, we have two forms of remote evaluation:

- `let box u = box M in N`
  Ordinary “spawn anywhere” evaluation.

- `let dia x = dia E in F`
  Sending code to the place where local resources reside.
Examples: Recursive Fibonacci

\[
\text{fix}_v \text{fib} :: \Box \text{int} \to \text{int}.
\]
\[
\lambda n : \Box \text{int}.
\]
\[
\text{let box } u = n \text{ in }
\]
\[
\text{if } (u < 2) \text{ then } u
\]
\[
\text{else }
\]
\[
\text{let box } a =
\]
\[
\text{box(\text{fib } \text{box}(u-1))) \text{ in }
\]
\[
\text{let box } b =
\]
\[
\text{box(\text{fib } \text{box}(u-2))) \text{ in }
\]
\[
(a + b)
\]
Examples: I/O Operations

Assuming `console :: Diamond console` representing a localized file handle:

```lisp
let dia c = console in
  write c "Enter a number:");
  write c "answer = ");
  write c ((\ x : int . M) (read c))
```
Examples: Callbacks

Assuming \( \text{lift} :: \text{int} \rightarrow \boxint \text{int} \), a runtime boxing operation.

let box callback =
  box (dia \{ (\lambda x : \text{int} . M) \}) in
  (* jump to console location *)
let dia c = console
  write c "Enter a number:";
let box n = lift (read c) in
  (* jump back to callback loc *)
let dia cb = callback in
  \{cb n\}
Talk Outline

- Concepts of modal logic
- Intuitionistic formalism
- Distributed programming
- Conclusions
Modal logic shows how to safely program with a combination of mobile and immobile entities.

- Restrictions of modal logic are not mandatory if you deny the existence of localized entities.
- Other ad-hoc solutions to marshalling/safety are possible.
- Novelty is in the logical explanation of distributed computation.
Conclusions

The assumptions at the foundations of modal logic bore fruit:

- From $A_{\text{poss}}$,
  - we have expressions (things with localized meaning).

- From $A_{\text{valid}}$,
  - we have closed terms (which are fully mobile).
Conclusions: Future work

- More logically explicit (explicit worlds)
  - Should allow more precise treatment of dia/letdia.

- Lower-level operational semantics
  (environments, stacks)

- Separate concurrency from distribution.
  - Concurrency could be orthogonal to box/letbox.
Conclusions

Acknowledgements:
- Frank Pfenning and Rowan Davies:
  “A Judgemental Reconstruction of Modal Logic”

Further Reading:
- “Modal Logic as a Basis for Distributed Computation”
Expression Substitution

Expression substitution is defined:

\[
\langle\langle \{M\}/x\rangle\rangle F = [M/x]F
\]

\[
\langle\langle \text{let } \text{dia } y = M \text{ in } E/x\rangle\rangle F = \\
\quad \text{let } \text{dia } y = M \text{ in } \langle\langle E/x\rangle\rangle F
\]

\[
\langle\langle \text{let } \text{box } u = M \text{ in } E/x\rangle\rangle F = \\
\quad \text{let } \text{box } u = M \text{ in } \langle\langle E/x\rangle\rangle F
\]
none