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

Jonathan Moody

(with Frank Pfenning)

Department of Computer Science

Carnegie Mellon University

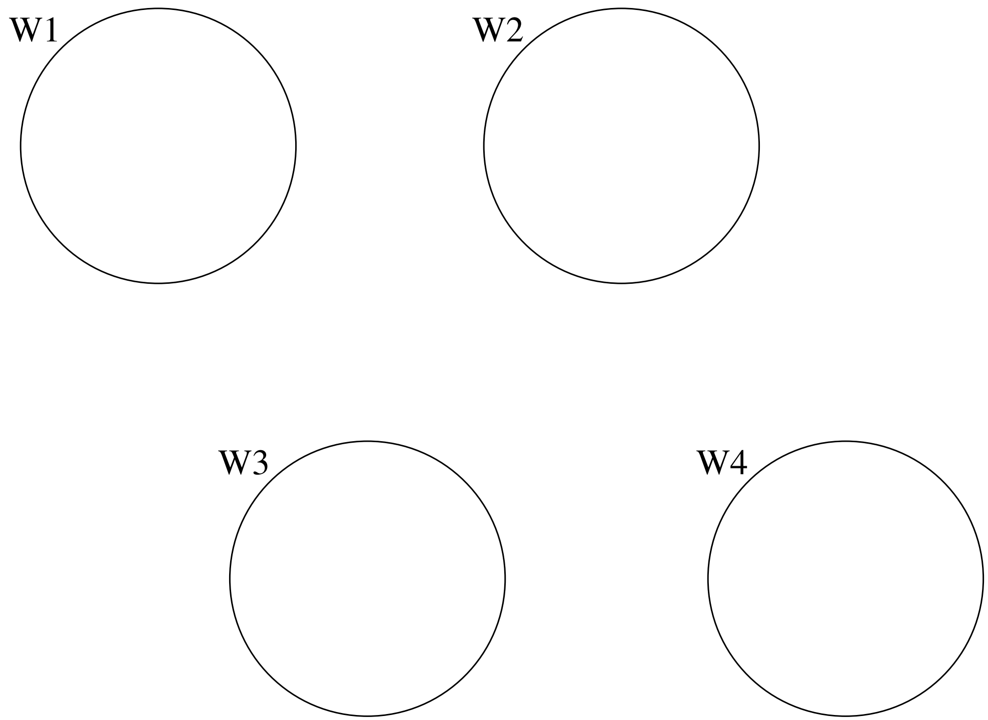
Talk Outline

- **Concepts of modal logic**
- Intuitionistic formalism
- Distributed programming
- Conclusions

Concepts of modal logic

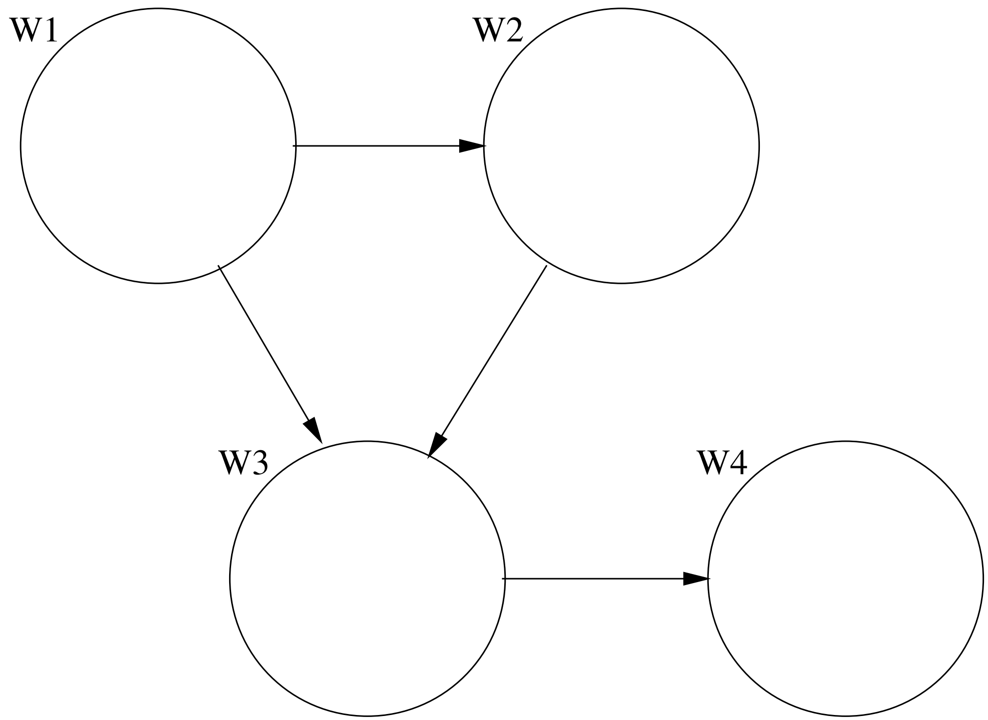
- Modal logic(s) distinguish **modes** of truth.
- For the generalized modal logic (S4) these modes of truth are explained by referring to (abstract) “worlds”:
  - Truth in **all** (accessible) worlds.
  - Truth in **this** world.
  - Truth in **some** (accessible) world.

Concepts: A Kripke model



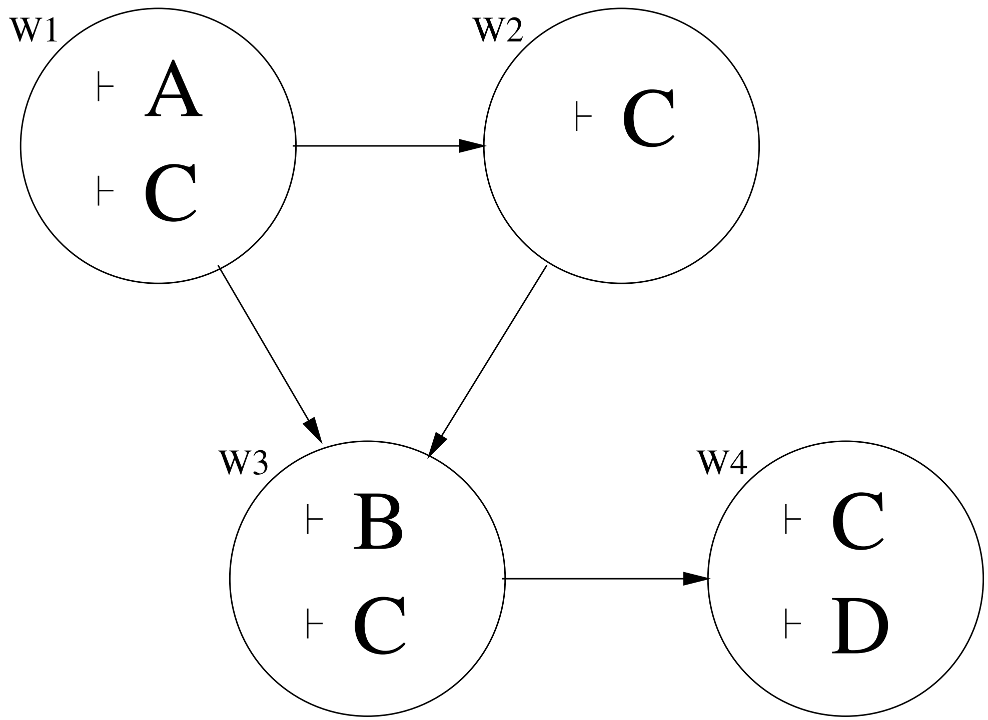
Worlds of the Kripke structure

Concepts: A Kripke model



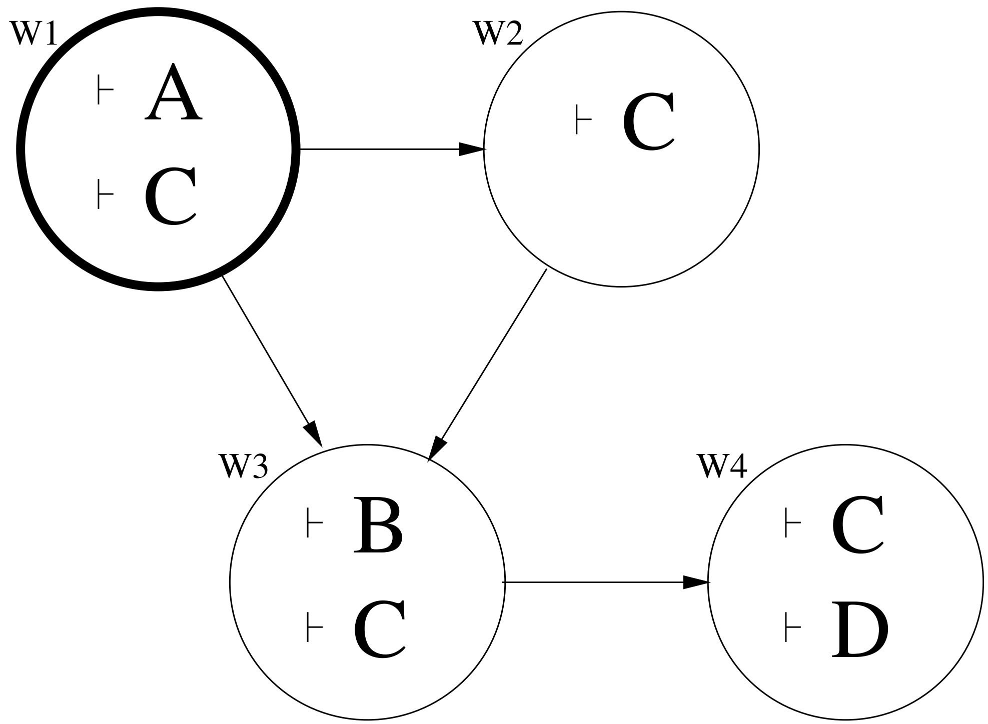
Accessibility between worlds

Concepts: A Kripke model



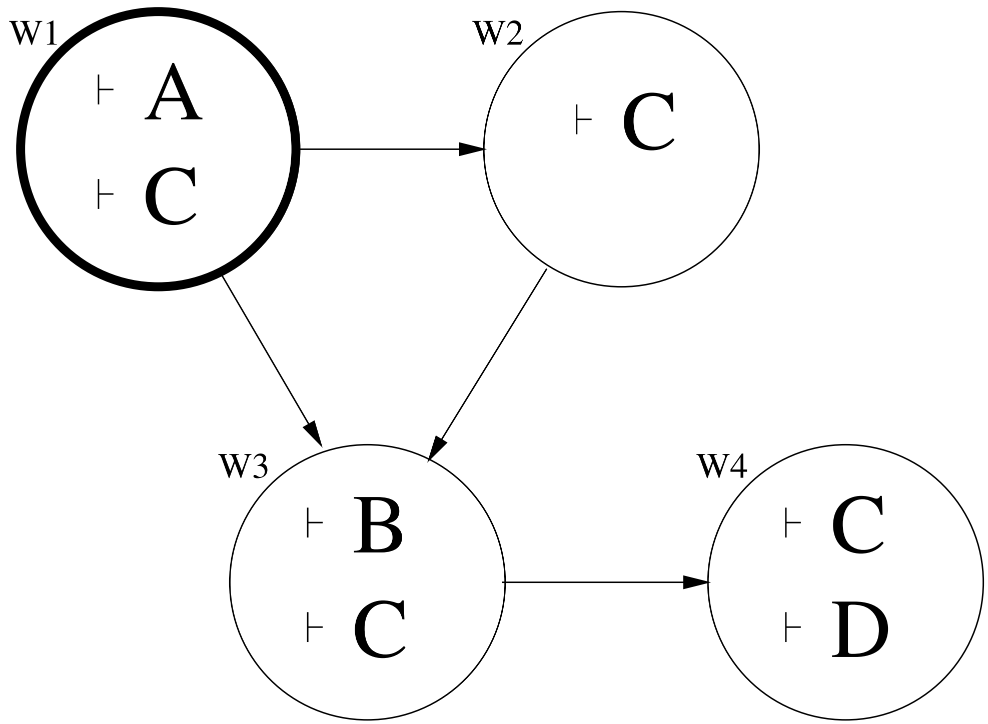
Primitive assumptions

Concepts: A Kripke model



From the perspective of world W_1 ...

Concepts: A Kripke model



- A and C are true here (W_1)

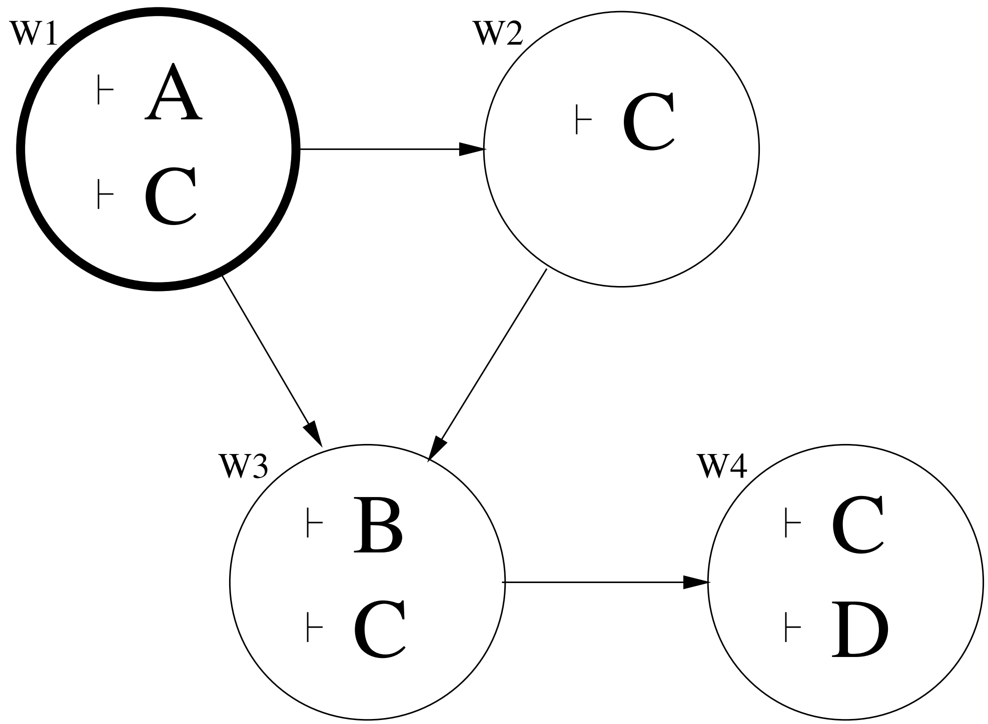
Concepts: Modal propositions

- By introducing new forms of proposition, we can make statements about other worlds.

$\Box A$ — A true in **all** accessible worlds.

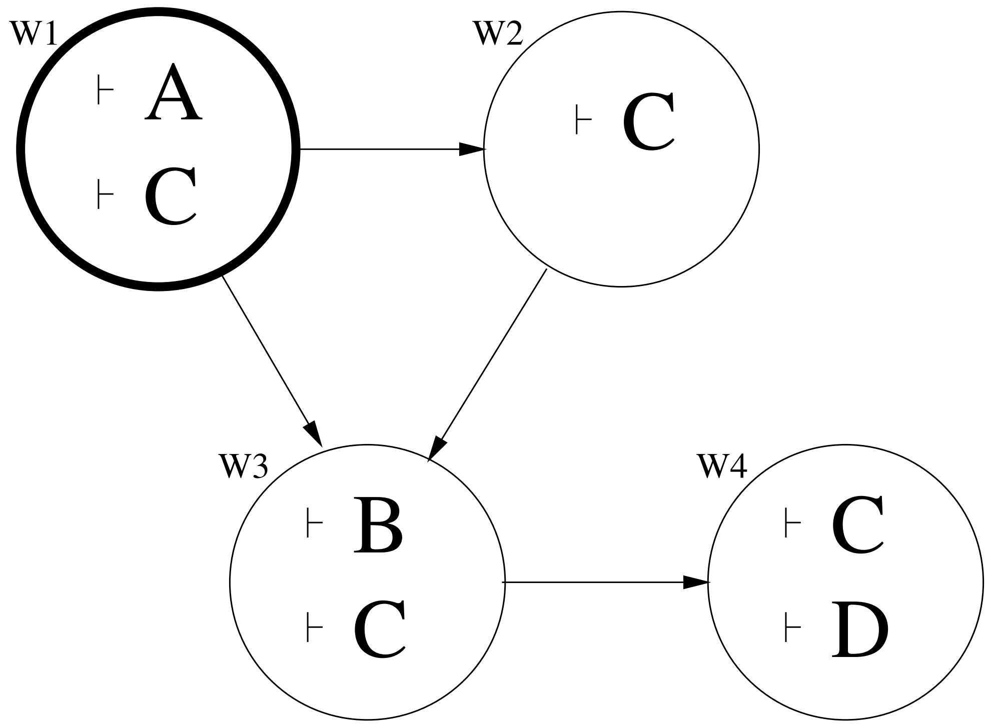
$\Diamond A$ — A true in **some** accessible world.

Concepts: A Kripke model



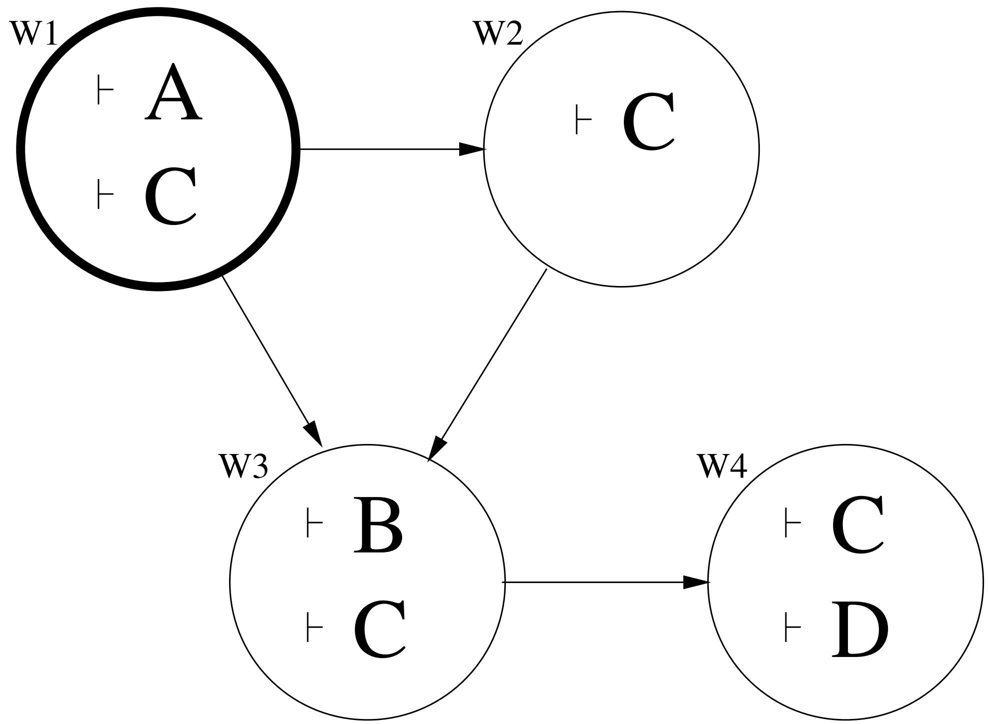
- $\Box C$ true at W_1 because...
  - C true at W_1, W_2, W_3, W_4 (refl. & trans.)

Concepts: A Kripke model



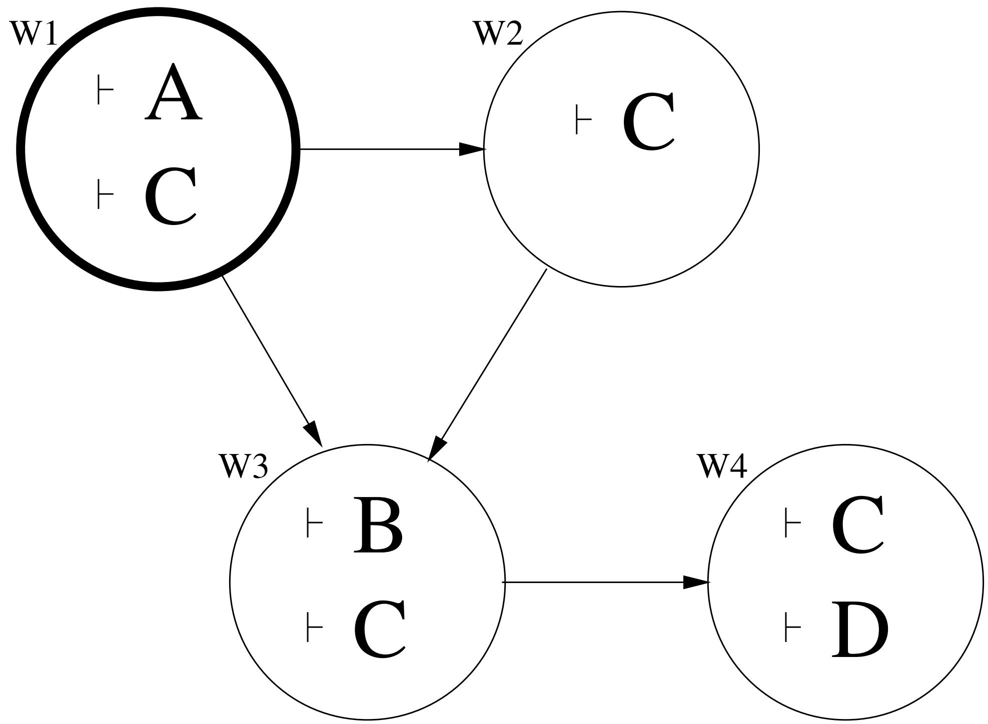
- $\Diamond A$ true at W_1 because...
  - A true at W_1 (reflexivity)

Concepts: A Kripke model



- $\Diamond B$ true at W_1 because...
  - B true at W_3

Concepts: A Kripke model



- $\Diamond D$ true at W_1 because...
  - D true at W_4 (transitivity)

Concepts: More concrete models

- 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)

Concepts: More concrete models

- For **distributed computation**, we adopt a ***spatial*** interpretation of worlds.

Concepts: More concrete models

- 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.

Concepts: More concrete models

- 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

Judgements and propositions

- 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:
  - $A\ \texttt{valid}$ (true everywhere)
  - $A\ \texttt{true}$ (true here)
  - $A\ \texttt{poss}$ (true somewhere)
- Propositions of the logic remain the same:
  - $\Box A$ (internalizes $A\ \texttt{valid}$)
  - $\Diamond A$ (internalizes $A\ \texttt{poss}$)
  - $A \rightarrow B$ (internalizes entailment)

Judgements and propositions

- Hypothetical judgements are represented as:
- $\Delta; \Gamma \vdash A\, \mathtt{true}$ (or $\Delta; \Gamma \vdash A\, \mathtt{poss}$)
  - Δ holds "global" hypotheses ($A\, \mathtt{valid}$)
  - Γ holds "local" hypotheses ($A\, \mathtt{true}$)

Language of proof terms

- Via a Curry-Howard isomorphism we can:

Pass from $\dfrac{\mathcal{P}}{\Delta;\Gamma \vdash A\,\texttt{true}}$ to $\Delta;\Gamma \vdash M : A$

And from $\dfrac{\mathcal{Q}}{\Delta;\Gamma \vdash A\,\texttt{poss}}$ to $\Delta;\Gamma \vdash E \div A$

- Terms and expressions are simultaneously proof objects **and** programs.

Language of proof terms

- Quick overview of syntax (more depth later):

$$
\begin{array}{lcl}
\text{Term } M, N & ::= & \mathbf{x} \mid \mathbf{u} \\
 & \mid & \lambda \mathbf{x} : A \,.\, M \mid M\ N \\
 & \mid & \texttt{box}\, M \mid \texttt{dia}\, E \\
 & \mid & \texttt{let box}\, \mathbf{u} = M \,\texttt{in}\, N \\
\text{Expr. } E, F & ::= & \{M\} \mid \texttt{let box}\, \mathbf{u} = M \,\texttt{in}\, F \\
 & \mid & \texttt{let dia}\, \mathbf{x} = M \,\texttt{in}\, F
\end{array}
$$

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 $A\,\texttt{true}$
  - 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).

Operational reading: Notation

- Evaluation context notation $\mathcal{R}[\,M\,]$
- (Term) values of the language:

$$\frac{}{\lambda \mathbf{x} : A \, . \, M \ \mathtt{tvalue}} \qquad \frac{}{\mathtt{box}\, M \ \mathtt{tvalue}}$$

$$\frac{}{\mathtt{dia}\, E \ \mathtt{tvalue}} \qquad \frac{}{r \ \mathtt{tvalue}}$$

- Note that language of terms is extended:
  - “Result” label r is considered a term value.
  - Allows processes to refer to one another.

Operational reading: $A \to B$

- "Local" variables and $\to$ intro/elim:

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

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

$$\frac{\Delta; \Gamma \vdash M : A \to B \quad \Delta; \Gamma \vdash N : A}{\Delta; \Gamma \vdash M\ N : B} \to E$$

Operational reading : $A \to B$

- Local reduction step:

$$\dfrac{\dfrac{\Delta;\Gamma,\mathbf{x}:A \vdash M' : B}{\Delta;\Gamma \vdash \lambda\mathbf{x}:A\,.\,M' : A \to B} \to I \qquad \Delta;\Gamma \vdash N : A}{\Delta;\Gamma \vdash (\lambda\mathbf{x}:A\,.\,M')\;N : B} \to E$$

$$\dfrac{V_1 = (\lambda\mathbf{x}:A\,.\,M') \qquad V_2\;\texttt{tvalue}}{\langle r : \mathcal{R}[\,V_1\;V_2\,]\rangle \Rightarrow \langle r : \mathcal{R}[\,[V_2/\mathbf{x}]M'\,]\rangle}\;app$$

Operational reading: $\Box A$

- “Global” variables and $\Box$ intro/elim:

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

$$\frac{\Delta; \cdot \vdash M : A}{\Delta; \Gamma \vdash \mathtt{box}\, M : \Box A}\ \Box I$$

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

Operational reading: $\Box A$

- Local reduction step:

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

$$\dfrac{V = \texttt{box}\, M \qquad r_2\ \texttt{fresh}}{\begin{array}{c}\langle r_1 : \mathcal{R}[\,\texttt{let box}\, \mathbf{u} = V\ \texttt{in}\, N\,]\rangle \\ \Rightarrow\ \ \langle r_2 : M\rangle; \langle r_1 : \mathcal{R}[\,[\![r_2/\mathbf{u}]\!]N\,]\rangle\end{array}}\ letbox$$

Operational reading: $\Box A$

- Synchronization on "result" labels (r)

$$\frac{V\ \texttt{tvalue}}{\langle r_2 : V \rangle; \langle r_1 : \mathcal{R}[\,r_2\,] \rangle \Rightarrow \langle r_2 : V \rangle; \langle r_1 : \mathcal{R}[\,V\,] \rangle}\ \mathit{syncr}$$

- Immediate synch. is not required ($r\ \texttt{tvalue}$).
  - We have a choice between synchronization ($\mathcal{R}[\,r\,]$) or the "usual" reduction step.
  - Concurrency is a **secondary effect** of the spatial interpretation, **not logically essential**.

Operational reading: Notation

- Now considering the **expression fragment** of the language...
  - Having $E \div A$ means that E “makes sense” somewhere (but not necessarily “here”).
  - We may not interpret expressions E until they are placed in the proper context.

Operational reading: Notation

- It is convenient to introduce expression variants of:
  - Processes: $\langle l : E \rangle$ and $\langle l_1 : l_2 \rangle$
  - Evaluation contexts: $\mathcal{S}[\, M \,]$ and $\mathcal{S}[\, E \,]$
  - Expression values:

$$\frac{V \;\texttt{tvalue}}{\{V\} \;\texttt{evalue}}$$

Operational reading: $\Diamond A$

- Relationship between truth and possibility:

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

- "Global" variables bound in expressions:

$$\frac{\Delta;\Gamma \vdash M : \Box A \quad \Delta, \mathbf{u} :: A;\Gamma \vdash F \div B}{\Delta;\Gamma \vdash \texttt{let box}\, \mathbf{u} = M\, \texttt{in}\, F \div B}\ \Box E_p$$

Operational reading: $\Diamond A$

- $\Diamond$ introduction and elimination:

$$\frac{\Delta;\Gamma \vdash E \div A}{\Delta;\Gamma \vdash \texttt{dia}\, E : \Diamond A}\ \Diamond I$$

$$\frac{\Delta;\Gamma \vdash M : \Diamond A \qquad \Delta;\mathbf{x} : A \vdash F \div B}{\Delta;\Gamma \vdash \texttt{let dia}\,\mathbf{x} = M\, \texttt{in}\, F \div B}\ \Diamond E$$

Operational reading: $\Diamond A$

- Local reduction step:

$$\dfrac{\dfrac{\Delta;\Gamma \vdash E \div A}{\Delta;\Gamma \vdash \mathtt{dia}\, E : \Diamond A}\,\Diamond I \qquad \Delta; \mathbf{x} : A \vdash F \div B}{\Delta;\Gamma \vdash \mathtt{let\ \ dia}\, \mathbf{x} = \mathtt{dia}\, E\, \mathtt{in}\, F \div B}\,\Diamond E$$

$$\dfrac{V = \mathtt{dia}\, E \quad l_2\ \mathtt{fresh}}{\begin{array}{ll} & \langle l_1 : \mathtt{let\ \ dia}\, \mathbf{x} = V\, \mathtt{in}\, F\rangle \\ \Rightarrow & \langle l_2 : \langle\langle E/\mathbf{x}\rangle\rangle F\rangle ; \langle l_1 : l_2\rangle \end{array}}\, letdia$$

- Note: location l_2 is **not** arbitrary.

Language of proof terms

- In summary:

$$
\begin{array}{llll}
\text{Term } M, N & ::= & \mathbf{x} \mid \mathbf{u} \\
& \mid & \lambda\mathbf{x} : A \,.\, M \mid M\ N \\
& \mid & \mathtt{box}\, M \mid \mathtt{dia}\, E \\
& \mid & \mathtt{let\ box}\, \mathbf{u} = M\ \mathtt{in}\ N \\
\text{Expr. } E, F & ::= & \{M\} \mid \mathtt{let\ box}\, \mathbf{u} = M\ \mathtt{in}\ F \\
& \mid & \mathtt{let\ dia}\, \mathbf{x} = M\ \mathtt{in}\ F
\end{array}
$$

Properties

- Typing for process configurations ($\vdash_C C : \Lambda$)

  Conf. Typing $\Lambda \quad ::= \quad \cdot \quad | \quad \Lambda, r :: A \quad | \quad \Lambda, l \div A$

  - "Result" labels $r :: A$ (logical validity).
  - "Location" labels $l \div A$ (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).

Properties

- Terminal processes:

$$\frac{V\ \mathtt{tvalue}}{\langle r : V \rangle\ \mathtt{terminal}}$$

$$\frac{V\ \mathtt{evalue}}{\langle l : V \rangle\ \mathtt{terminal}} \quad \frac{}{\langle l_1 : l_2 \rangle\ \mathtt{terminal}}$$

Properties

- **Progress** holds for well-formed config. C:
  - if $\vdash_C C : \Lambda$
  - then $C \Rightarrow C'$ or C $\texttt{terminal}$
- 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.

Properties

- 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

Marshalling

- 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.

Marshalling

- The marshallable/non-marshallable distinction is **critical**:
  - Semantic anomalies if you get it wrong.
  - Code mobility depends on parameter mobility.

Marshalling

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

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.)?

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.**

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?

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.**

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.**
  - heap addresses?

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.**
  - heap addresses? **no.**

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.**
  - heap addresses? **no.**
  - file handles?

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.**
  - 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:

$$\frac{\Delta; \cdot \vdash M : A}{\Delta; \Gamma \vdash \texttt{box}\, M : \Box A}\ \Box I$$

$$\frac{\Delta; \Gamma \vdash M : \Diamond A \qquad \Delta; \mathbf{x} : A \vdash F \div B}{\Delta; \Gamma \vdash \texttt{let dia}\, \mathbf{x} = M\, \texttt{in}\, F \div B}\ \Diamond E$$

- $\Box I$ permits only globally valid parameters.
- $\Diamond E$ permits param. from a **single** location.

The logical solution

- Moreover, we have two forms of remote evaluation:
  - $\texttt{let box}\,\mathbf{u} = \texttt{box}\,M\,\texttt{in}\,N$
    - Ordinary “spawn anywhere” evaluation.
  - $\texttt{let dia}\,\mathbf{x} = \texttt{dia}\,E\,\texttt{in}\,F$
    - Sending code to the place where local resources reside.

Examples : Recursive Fibonacci

```
fix_v fib :: □int → int .
λ n : □int .
  let box u = n in
    if (u < 2) then u
    else
      let box a =
        box(fib (box(u-1))) in
      let box b =
        box(fib (box(u-2))) in
     (a + b)
```

Examples: I/O Operations

- Assuming `console` $:: \Diamond$`con` representing a localized file handle:

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

Examples: Callbacks

- Assuming $\texttt{lift} :: \texttt{int} \rightarrow \Box\texttt{int}$, a **runtime** boxing operation.

```
let box callback =
  box (dia {(λ x : 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**

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 `poss`,
    - we have expressions (things with localized meaning).
  - From A `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"
    `http://www.cs.cmu.edu/~jwmoody/doc/np/modalbasis.pdf`

The End

Expression Substitution

- Expression substitution is defined:

$$\begin{aligned}
\langle\langle \{M\}/\mathbf{x}\rangle\rangle F &= [M/\mathbf{x}]F \\
\langle\langle \texttt{let dia}\,\mathbf{y} = M \,\texttt{in}\, E/\mathbf{x}\rangle\rangle F &= \\
&\texttt{let dia}\,\mathbf{y} = M \,\texttt{in}\, \langle\langle E/\mathbf{x}\rangle\rangle F \\
\langle\langle \texttt{let box}\,\mathbf{u} = M \,\texttt{in}\, E/\mathbf{x}\rangle\rangle F &= \\
&\texttt{let box}\,\mathbf{u} = M \,\texttt{in}\, \langle\langle E/\mathbf{x}\rangle\rangle F
\end{aligned}$$

- none

- none

- none