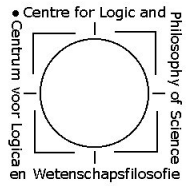
# A modal type system for safe distributed computing

Giuseppe Primiero

FWO - Flemish Research Foundation
Centre for Logic and Philosophy of Science, Ghent University





Giuseppe.Primiero@Ugent.be
http://www.philosophy.ugent.be/giuseppeprimiero/

PCC12, Copenhagen, 17th August 2012

# Outline

# Logical Approaches to Distributed Programming

- There is a great variety of systems that approach the issue of distributed programming in view of resource accessibility, safety, security, etc.

# Logical Approaches to Distributed Programming

- There is a great variety of systems that approach the issue of distributed programming in view of resource accessibility, safety, security, etc.
- The idea of using Curry-Howard based Semantics to this purpose is natural . . .

# Logical Approaches to Distributed Programming

- There is a great variety of systems that approach the issue of distributed programming in view of resource accessibility, safety, security, etc.
- The idea of using Curry-Howard based Semantics to this purpose is natural . . .
- . . . and it has been very recently combined with the extension of the modal apparatus to gain more control on the notion of resources, their location and accessibility.

# Some relevant works

- A typed $\lambda$-calculus with stationary situations and flowing informations:
  - [Borghuis and Feijs, 2000]: focus on the representation of the order of commands and recover of data from locations.

# Some relevant works

- A typed $\lambda$-calculus with stationary situations and flowing informations:
  - [Borghuis and Feijs, 2000]: focus on the representation of the order of commands and recover of data from locations.

- Type Theories for (safe) distributed and staged computing:
  - [Davies and Pfenning, 2001], [Jia and Walker, 2004], [Moody, 2003]: represent heterogeneity w.r.t. properties, resources, devices, software, services.

# Some relevant works

- A typed $\lambda$-calculus with stationary situations and flowing informations:
  - [Borghuis and Feijs, 2000]: focus on the representation of the order of commands and recover of data from locations.

- Type Theories for (safe) distributed and staged computing:
  - [Davies and Pfenning, 2001], [Jia and Walker, 2004], [Moody, 2003]: represent heterogeneity w.r.t. properties, resources, devices, software, services.

- Modal logics for local resources:
  - [Park, 2006]: distinction between transmission of safe values and safe code.

# Some relevant works

- A typed $\lambda$-calculus with stationary situations and flowing informations:
  - [Borghuis and Feijs, 2000]: focus on the representation of the order of commands and recover of data from locations.
- Type Theories for (safe) distributed and staged computing:
  - [Davies and Pfenning, 2001], [Jia and Walker, 2004], [Moody, 2003]: represent heterogeneity w.r.t. properties, resources, devices, software, services.
- Modal logics for local resources:
  - [Park, 2006]: distinction between transmission of safe values and safe code.
- **ILP** with $\Box$ for certified mobile computing:
  - [Bonelli and Feller, 2009]: code and certificate development; corresponds to a variant of the intensional $\lambda$-calculus introduced in [Artemov and Bonelli, 2007]: operational interpretation for remote calls.

# This contribution: meaning of expressions

We introduce a polymorphic typed system with multi-modal operators that distinguishes between safe values and safe code (extended from [Primiero, 2012]):

# This contribution: meaning of expressions

We introduce a polymorphic typed system with multi-modal operators that distinguishes between safe values and safe code (extended from [Primiero, 2012]):

- $a_i : A$ says that program $a$ is executed at address $i$ for specification $A$, producing a safe value;

# This contribution: meaning of expressions

We introduce a polymorphic typed system with multi-modal operators that distinguishes between safe values and safe code (extended from [Primiero, 2012]):

- $a_i : A$ says that program $a$ is executed at address $i$ for specification $A$, producing a safe value;
- $x_i : A$ says that code for specification $A$ is validly executable at (and bounded to) address $i$, producing mobile code;

# This contribution: meaning of expressions

We introduce a polymorphic typed system with multi-modal operators that distinguishes between safe values and safe code (extended from [Primiero, 2012]):

- $a_i : A$ says that program $a$ is executed at address $i$ for specification $A$, producing a safe value;
- $x_i : A$ says that code for specification $A$ is validly executable at (and bounded to) address $i$, producing mobile code;
- $\Box_i(A\ true)$: "$A$ is valid at every address accessible from $i$";

# This contribution: meaning of expressions

We introduce a polymorphic typed system with multi-modal operators that distinguishes between safe values and safe code (extended from [Primiero, 2012]):

- $a_i : A$ says that program $a$ is executed at address $i$ for specification $A$, producing a safe value;
- $x_i : A$ says that code for specification $A$ is validly executable at (and bounded to) address $i$, producing mobile code;
- $\Box_i(A\ true)$: "$A$ is valid at every address accessible from $i$";
- $\Diamond_i(A\ true)$: "$A$ is executable from address $i$";

# This contribution: meaning of expressions

We introduce a polymorphic typed system with multi-modal operators that distinguishes between safe values and safe code (extended from [Primiero, 2012]):

- $a_i : A$ says that program $a$ is executed at address $i$ for specification $A$, producing a safe value;
- $x_i : A$ says that code for specification $A$ is validly executable at (and bounded to) address $i$, producing mobile code;
- $\Box_i(A\ true)$: "$A$ is valid at every address accessible from $i$";
- $\Diamond_i(A\ true)$: "$A$ is executable from address $i$";
- $\circ\Gamma \vdash \circ(A\ true)$: *contexts* describe networks in which code is executed; their internal structure refers to ordered composition of commands.

# Some Features

- Language in an operational semantics with underlying Curry-Howard isomorphism:
  - Categorical fragment used to interpret operations with safe values
  - Functional fragment used to interpret distributed computations with safe code
- Modification of the standard interpretation of propositional modalities;
- Modalities are used to reason on the contexts/locations that are safe to the evaluation of the $\lambda$-terms.

# This Contribution: sum up

The polymorphism and the resulting modal type system are used therefore to explore reasoning about distributed computing, obtaining Code Mobility Rules from corresponding properties of the modal operators. Significant contributions of this work are:

1. the strong assumption on the polymorphic nature of codes and values, understood as locally and globally valid processes;
2. an alternative formulation and interpretation of (multi-)modalities for safe distributed computing;
3. the underlying operational semantics for the interpretation of distributed programs.

# Language

## Definition (Kinds)

The set $\mathcal{K} =: \{\mathtt{type}, \mathtt{type}_{\mathtt{inf}}\}$ contains

- the kind $\mathtt{type}$ of all specifications valid by everywhere executable programs, defined by term constructors $\mathcal{C}$;
- the kind $\mathtt{type}_{\mathtt{inf}}$ of specifications valid by locally executable codes, defined by variable constructors $\mathcal{V}$.

# Language

## Definition (Kinds)

The set $\mathcal{K} =: \{\texttt{type}, \texttt{type}_{\texttt{inf}}\}$ contains

- the kind $\texttt{type}$ of all specifications valid by everywhere executable programs, defined by term constructors $\mathcal{C}$;
- the kind $\texttt{type}_{\texttt{inf}}$ of specifications valid by locally executable codes, defined by variable constructors $\mathcal{V}$.

## Definition (Terms)

The set of terms $\mathcal{T} = \{\mathcal{C}, \mathcal{V}\}$ is given by:

- constructors $\mathcal{C} := \{a_i; (a_i, b_j); a_i(b_j); \lambda(a_i(b_j)); < a_i, b_j >\}$;
- variables $\mathcal{V} := \{x_i; (x_i(b_j)); (x_i(b_j))(a_i)\}$.

# Language

## Definition (Kinds)

The set $\mathcal{K} =: \{\mathtt{type}, \mathtt{type}_{\mathtt{inf}}\}$ contains

- the kind $\mathtt{type}$ of all specifications valid by everywhere executable programs, defined by term constructors $\mathcal{C}$;
- the kind $\mathtt{type}_{\mathtt{inf}}$ of specifications valid by locally executable codes, defined by variable constructors $\mathcal{V}$.

## Definition (Terms)

The set of terms $\mathcal{T} = \{\mathcal{C}, \mathcal{V}\}$ is given by:

- constructors $\mathcal{C} := \{a_i; (a_i, b_j); a_i(b_j); \lambda(a_i(b_j)); < a_i, b_j >\}$;
- variables $\mathcal{V} := \{x_i; (x_i(b_j)); (x_i(b_j))(a_i)\}$.

Judgements are generalised to their contextual form:

$$\Gamma_i := \cdot, x_i : A_i, \Delta_i$$
$$\Delta_i := \cdot, a_i : A_i$$

# Interpreting safe values

## Definition (Introduction Rules for *type*)

The rules for signed expressions in the kind *type* are (Eliminations and Equality are omitted):

$$\frac{a_i : A}{A\ type}\ \textit{Type Formation}$$

$$\frac{a_i : A \quad b_j : B}{(a_i, b_j) : A \wedge B}\ I\wedge \qquad \frac{a_i : A}{l(a_i) : A \vee B}\ \textit{Left}\, I \qquad \frac{b : B}{r(b) : A \vee B}\ \textit{Right}\, I$$

$$\frac{a_i : A \quad A\ type \vdash b_j : B}{a_i(b_j) : A \to B}\ I\to$$

$$\frac{a_1 : A, \ldots, a_n : A \quad a_i : A \vdash b_j : B \quad \lambda((a_i(b_j))A, B)}{(\forall a_i : A)B\ type}\ I\forall$$

$$\frac{a_1 : A, \ldots, a_n : A \quad a_i : A \vdash b_j : B \quad (< a_i, b_j >, A, B)}{(\exists a_i : A)B\ type}\ I\exists$$

# Interpreting safe code

## Definition (Rules for $type_{inf}$)

The rules for signed expressions in the kind $type_{inf}$ are:

$$\frac{}{\Gamma_i, x_j : A, \Delta_i \vdash A\ type_{inf}}\ \text{Local Validity Rule}$$

$$\frac{A\ type_{inf} \quad x_i : A \vdash b_j : B}{((x_i)b_j) : A \supset B}\ \text{Functional abstraction}$$

$$\frac{A\ type_{inf} \quad x_i : A \vdash b_j : B \quad a_i : A}{(x_i(b_j))(a_i) = b_j[a/x]_i : B\ type[a/x]_i}\ \beta - conversion$$

$$\frac{((x_{i-1}(b_j))A, B) \quad (b_j)[x/a_{i-1} := a_i]}{(a_i(b_j)) : A \to B}\ \alpha - conversion$$

# Structural Rules

## Lemma

*Structural Rules for*

- *Weakening,*
- *Contraction*
- *and Exchange*

*are admissible for type and (restrictedly for) $type_{inf}$ expressions.*

# Truth Predicates

## Definition (Semantic Judgements)

The sorting $\mathcal{K}$ induces truth definitions as follows:

$$\frac{\Delta_i; \cdot \vdash a_i : A}{A\ true}\ GlobalTruth$$

$$\frac{\Delta_i; \Gamma_j \vdash x_j : A}{\Delta_i; \Gamma_j \vdash A\ true^*}\ LocalTruth$$

where $\Delta$ contains only valid assumptions of the form $B\ type$ and $\Gamma$ contains at least one true assumption of the form $B\ type_{inf}$, appropriately addressed at $i$.

# The Modal Extension

We now induce modalities for expressions from the constructors for kinds, according to the following intuitive explanations:

# The Modal Extension

We now induce modalities for expressions from the constructors for kinds, according to the following intuitive explanations:

- $\Box(A\ true)$:
  - Program for $A$ is everywhere satisfied;
  - $A\ true$ holds under any extension of globally valid conditions $\emptyset, \Delta$.

# The Modal Extension

We now induce modalities for expressions from the constructors for kinds, according to the following intuitive explanations:

- $\Box(A\ true)$:
  - Program for $A$ is everywhere satisfied;
  - $A\ true$ holds under any extension of globally valid conditions $\emptyset, \Delta$.

- $\Diamond(A\ true)$:
  - Program for $A$ can be executed somewhere (where correctly accessed);
  - $A\ true$ holds under some context extension $\Delta, \Gamma$.

# Introduction and Elimination for $\Box$

## Definition (Rules for $\Box_{\mathcal{G}}\Sigma$)

$$\frac{\Gamma_i; x_j : A \vdash A\ true^* \quad \Box_i\Gamma, [x_j/a_j] : A \vdash A\ true}{\Box_{i\cup j}\Sigma \vdash \Box_{i\cup j}(A\ true)}\ I\Box$$

$$\frac{\Box_i\Gamma, a_j : A \vdash \Box_{i\cup j}(A\ true) \qquad \Box_{i\cup j}(A\ true), \Box_k\Delta \vdash \Box_{\cup(i,j,k)}(B\ true)}{\Gamma_i, a_j : A, \Delta_k \vdash B\ true}\ E\Box$$

# Introduction and Elimination for $\Box$

## Definition (Rules for $\Box_{\mathcal{G}}\Sigma$)

$$\frac{\Gamma_i; x_j : A \vdash A\ true^* \quad \Box_i\Gamma, [x_j/a_j] : A \vdash A\ true}{\Box_{i\cup j}\Sigma \vdash \Box_{i\cup j}(A\ true)}\ I\Box$$

$$\frac{\Box_i\Gamma, a_j : A \vdash \Box_{i\cup j}(A\ true) \qquad \Box_{i\cup j}(A\ true), \Box_k\Delta \vdash \Box_{\cup(i,j,k)}(B\ true)}{\Gamma_i, a_j : A, \Delta_k \vdash B\ true}\ E\Box$$

- I-$\Box$: if a program for $A$ uses only safe values originating at $i, j$, then it can be executed everywhere in network $\mathcal{G} = \{i, j\}$ (induces an operational interpretation as Remote Procedure Call);
- E-$\Box$: sends value $A$ from $i, j$ to $\mathcal{G}$, where it can be used to evaluate $B$ at any further accessible address $k$.

# Introduction and Elimination for $\Diamond$

## Definition (Rules for $\Diamond_{\mathcal{G}}\Sigma$)

$$\frac{\Gamma_i; x_j : A \vdash B\ true^*}{\Diamond_{i\cap j}\Sigma \vdash \Diamond_{i\cap j}(B\ true)}\ I\Diamond$$

$$\frac{\Box_i\Gamma; \Diamond_j\Delta \vdash \Diamond_{i\cap j}(A\ true) \qquad \Diamond_j\Delta; x_k : A \vdash \Diamond_{j\cap k}(B\ true)}{\Gamma_i; \Delta_j \vdash B\ true^*}\ E\Diamond$$

# Introduction and Elimination for $\Diamond$

## Definition (Rules for $\Diamond_{\mathcal{G}}\Sigma$)

$$\frac{\Gamma_i; x_j : A \vdash B\ true^*}{\Diamond_{i\cap j}\Sigma \vdash \Diamond_{i\cap j}(B\ true)}\ I\Diamond$$

$$\frac{\Box_i\Gamma; \Diamond_j\Delta \vdash \Diamond_{i\cap j}(A\ true) \qquad \Diamond_j\Delta; x_k : A \vdash \Diamond_{j\cap k}(B\ true)}{\Gamma_i; \Delta_j \vdash B\ true^*}\ E\Diamond$$

- I-$\Diamond$: if value $B$ requires safe code executable at $i$ and $j$, then resources at the intersection of $i, j$ are needed for any execution; it constructs a return value for a RPC;
- E-$\Diamond$: from $\Diamond_{i,j}(A\ true)$ infer its variable constructor, then deriving local validity of $B$ without the additional location of $A$.

# Interaction among Modalities: Code Mobility Rules

## Definition (Broadcast)

Broadcast is used to send to a specific additional address a safe value in the network $\mathcal{G} = \{i, j\}$.

$$\frac{\Box_i \Gamma, a_j : A \vdash \Box_{i \cup j}(B\ true) \qquad x_j : A \vdash A\ true^*}{\Box_i \Gamma, \Diamond_j (A\ true) \vdash \Diamond_{i \cap j}(B\ true)} \quad (Broadcast)$$

# Interaction among Modalities: Code Mobility Rules

## Definition (Broadcast)

Broadcast is used to send to a specific additional address a safe value in the network $\mathcal{G} = \{i, j\}$.

$$\frac{\Box_i \Gamma, a_j : A \vdash \Box_{i \cup j}(B\ true) \qquad x_j : A \vdash A\ true^*}{\Box_i \Gamma, \Diamond_j (A\ true) \vdash \Diamond_{i \cap j}(B\ true)} \quad (Broadcast)$$

## Definition (Rules)

Global Access is the reverse function that calls from a specific address within network $\mathcal{G}$ safe code that becomes executable at any address.

$$\frac{\Box_i \Gamma; x_j : A \vdash \Diamond_{i \cap j}(B\ true) \qquad a_j : A \vdash A\ true}{\Box_i \Gamma, a_j : A \vdash \Box_{i \cup j}(B\ true)} \quad (Global\ Access)$$

# Interaction among Modalities: Some Properties

## Definition (Admissible Rules)

Transmission (downward transitivity): if a process $B'$ at $k$ uses code $B$ at $j$, and $B$ uses code $A$ at $i$, then $B$ at $k$ uses $A$ at $i$ (for $i < j < k \in \mathcal{G}$)

$$\frac{x_i : A \vdash A\ true^* \quad \Diamond_j(B\ true)[\Diamond_i(A\ true)] \quad \Diamond_k(B'\ true)[\Diamond_j(B\ true)]}{\Diamond_i(A\ true) \vdash \Diamond_k(B'\ true)}\ \text{Transmission}$$

# Interaction among Modalities: Properties (cnt'd)

## Definition

Upper Inclusion: if value for $A$ is valid at $k$, then it can be accessed from any location $i, j < k$ within that network

$$\frac{\Box_{i\cup j}\Sigma \vdash \Box_k(A\ true) \quad \Box_{i\cup j}\Sigma; a_k : A \vdash \Box_{\cup(i,j,k)}(A\ true)}{\Box_{\cup(i,j,k)}\Sigma \vdash \Box_{i\cup j}(A\ true)}\ \textit{Upper Inclusion}$$

## Definition (Admissible Rules)

Lower Inclusion: if value for $A$ is valid at $i, j$, then it can be sent to any location $k > i, j$ within that network;

$$\frac{\Box_i\Gamma, \Box_j\Delta \vdash \Box_{i\cup j}(A\ true) \quad \Box_{i\cup j}\Sigma \vdash \Box_k(A\ true)}{\Box_{\cup(i,j,k)}\Sigma \vdash \Box_k(A\ true)}\ \textit{Lower Inclusion}$$

# Interaction among Modalities: Properties (cnt'd)

## Definition (Admissible Rules)

$$\frac{\Box_i\Gamma, \Box_j\Delta \vdash \Box_k(A\ true)}{\Box_{\cup(i,j,k)}\Sigma \vdash \Box_k(\Box_{i\cup j}(A\ true))}\ \textit{Ascending Iteration}$$

$$\frac{\Box_i\Gamma, \Box_j\Delta \vdash \Box_k(A\ true)}{\Box_{\cup(i,j,k)}\Sigma \vdash \Box_{i\cup j}(\Box_k(A\ true))}\ \textit{Descending Iteration}$$

By Ascending Iteration, one can access at $k$ a value for $A$ executed at $i, j$ whenever a program for $A$ can be executed at $k$ using values at $i, j$;

by Descending Iteration, one can access at $k$ a value for $A$ executed at $i, j$, whenever a program for $A$ is executable at $k$ with values at $i, j$

# Language

## Definition (Syntax)

The syntax is defined by the following alphabet:

$$\texttt{Types} := \{\alpha \mid \alpha \times \beta \mid \alpha + \beta \mid \alpha \to \beta \mid \alpha \supset \beta\}$$

$$\texttt{Terms} := \{x_i \mid a_i, \text{ for } i \in \texttt{Indices}\}$$

$$\texttt{Indices} := \{1, \ldots, n\}$$

$$\texttt{Functions} := \{\mathit{exec}(\alpha) \mid \mathit{run}_i(\alpha) \mid \mathit{run}_{i\cup j}(\alpha \cdot \beta) \mid \mathit{run}_{i\cap j}(\alpha \cdot \beta) \mid \mathit{synchro}_j(\beta(\mathit{exec}(\alpha)))\}, \text{ where } \cdot = \{+, \times\}$$

$$\texttt{Contexts} := \{\Gamma_i \mid \circ_i \Gamma\}, \text{ where } \circ = \{\Box, \Diamond\}$$

$$\texttt{Remote Operations} := \{\mathit{GLOB}(\Box_{i\cup j}\Gamma, \alpha) \mid \mathit{BROAD}(\Diamond_{i\cap j}\Gamma, \alpha)\}$$

$$\texttt{Portable Code} := \{\mathit{RET}(\Gamma_{i\cup j}, \alpha) \mid \mathit{SEND}(\Gamma_{i\cap j}, \alpha)\}$$

# Model

Syntactic expressions are then evaluated in a model defined by states of the machine.

## Definition (Operational Model)

The set $\texttt{States} := \{S, S', \dots\}$ contains states of the machine. A state

$$S := (\mathcal{C}, t.i:\alpha) \mid \mathcal{C} \in \texttt{Contexts}; t \in \texttt{Terms}; i \in \texttt{Indices}; \alpha \in \texttt{Types}$$

is an occurrence of an indexed typed term in context. An operational model of the procedural semantics for the machine is a model where each $S$ is evaluated by transition to some $S'$. An indexed transition system, called a $\texttt{Network}$

$$\texttt{Network} := (\mathcal{S}, \mapsto, \mathcal{I})$$

is a triple with $\mathcal{S} \subseteq \texttt{States}$, $\mathcal{I} \subseteq \texttt{Indices}$ and $\mapsto$ a ternary relation over indexed states $(\mathcal{S} \times \mathcal{I} \times \mathcal{S})$. If $S, S' \in \mathcal{S}$ and $i, j \in \mathcal{I}$, then $\mapsto (S, i, j, S')$ is written as $S_i \mapsto S'_j$. This means that there is a transition $\mapsto$ from state $S$ valid at index $i$ to state $S'$ valid at index $j$ defined according to the machine typing rules.

# Operational Semantics (II)

## Definition (Network State)

| | $S \mapsto S'$ |
|---|---|
| run | $(\Gamma_i, x_i : \alpha) \mapsto (\Diamond_i \Gamma, run_i(\alpha))$ |
| exec | $(\Gamma_i, a_i : \alpha) \mapsto (\Box_i \Gamma, exec(\alpha))$ |
| corun | $(\Gamma_i, run_i(\alpha) \vdash b_j : \beta) \mapsto (\Box_i \Gamma, run_{i \cap j}(\alpha(\beta))$ |
| coexec | $(\Gamma_i, exec(\alpha) \vdash b_j : \beta) \mapsto (\Box_i \Gamma, run_{i \cup j}(\alpha(\beta))$ |
| synchro | $(\Box_i \Gamma, run_{i \cup j}(\alpha(\beta)) \mapsto (\Box_i \Gamma, synchro_j(\beta(exec(\alpha))))$ |
| product | $(\Gamma_i, exec(\alpha), exec(\beta)) \mapsto (\Box_i \Gamma, run_{i \cap j}(\alpha \times \beta))$ |
| extraction1 | $(\Box_i \Gamma, run_{i \cap j}(\alpha \times \beta)) \mapsto (\Box_i \Gamma, exec(\alpha))$ |
| extraction2 | $(\Box_i \Gamma, run_{i \cap j}(\alpha \times \beta)) \mapsto (\Box_i \Gamma, exec(\beta))$ |
| tagunion | $(\Gamma_i, exec(\alpha)) \mapsto (\Box_i \Gamma, run_{i \cup j}(\alpha + \beta))$ |
| patternmatch1 | $(\Box_i \Gamma, run_{i \cup j}(\alpha + \beta) \vdash c_k : \gamma) \mapsto (\Box_i \Gamma, run_{i \cap k}(\alpha(\gamma)))$ |
| patternmatch2 | $(\Box_i \Gamma, run_{i \cup j}(\alpha + \beta) \vdash c_k : \gamma) \mapsto (\Box_i \Gamma, run_{j \cap k}(\beta(\gamma)))$ |
| $\Box 1$ | $(\Box_i \Gamma, exec(\alpha)) \mapsto (GLOB(\Box_{i \cup j} \Gamma, \alpha))$ |
| $\Box 2$ | $(\Box_{i \cup j} \Gamma, \alpha) \mapsto (RET(\Gamma_{i \cup j}, \alpha))$ |
| $\Diamond 1$ | $(\Diamond_i \Gamma, run_j(\alpha)) \mapsto (BROAD(\Diamond_{i \cap j} \Gamma, \alpha))$ |
| $\Diamond 2$ | $(\Diamond_{i \cap j} \Gamma, \alpha) \mapsto (SEND(\Gamma_{i \cap j}, \alpha))$ |

# Rewriting Rules I

$$\frac{}{\Delta_i, a_i : \alpha \vdash exec(\alpha)}\ Global$$

$$\frac{}{\Gamma_i, x_i : \alpha; \Delta_i \vdash run_i(\alpha)}\ Local$$

$$\frac{a_i : \alpha \quad b_j : \beta}{run_{i \cap j}(\alpha \times \beta)}\ I\times$$

$$\frac{run_{i \cap j}(\alpha \times \beta)}{exec(\alpha)}\ E \times (I)$$

$$\frac{a_i : \alpha}{run_i(\alpha + \beta)}\ I + (1)$$

$$\frac{b_j : \beta}{run_j(\alpha + \beta)}\ I + (2)$$

# Rewriting Rules II

$$\frac{run_{i\cup j}(\alpha+\beta) \quad run_i(\alpha)\vdash c_k:\gamma \quad run_j(\beta)\vdash c_k:\gamma}{run_{i\cap k;j\cap k}(\gamma)}\ E+$$

$$\frac{x_i:\alpha \quad run_i(\alpha)\vdash b_j:\beta}{run_{i\cap j}(\alpha\supset\beta)}\ I\supset$$

$$\frac{a_i:\alpha \quad exec(\alpha)\vdash b_j:\beta}{run_{i\cup j}(\alpha\rightarrow\beta)}\ I\rightarrow$$

$$\frac{run_{i\cap j}(\alpha\supset\beta) \quad a_i:\alpha}{synchro_j(\beta(exec(\alpha)))}\ Synchro$$

$$\frac{\Gamma_i, x_j:\alpha\vdash run_j(\alpha) \quad \Box_i\Gamma, x_j(a_j):\alpha\vdash exec(\alpha)}{GLOB(\Box_{i\cup j}\Gamma,\alpha)}\ RPC1$$

$$\frac{\Gamma_i, x_j:\alpha\vdash run_j(\alpha) \quad \Diamond_i\Gamma\vdash run_j(\alpha)}{BROAD(\Diamond_{i\cap j}\Gamma,\alpha)}\ RPC2$$

# Rewriting Rules III

$$\frac{\Box_i\Gamma, a_j : \alpha \vdash \mathit{exec}(\alpha) \quad \mathit{GLOB}(\Box_{i\cup j}\Gamma, \alpha)}{\mathit{RET}(\Gamma_{i\cup j}, \alpha)}\ \mathit{PORT1}$$

$$\frac{\Box_i\Gamma, x_j : \alpha \vdash \mathit{run}_{i\cap j}(\alpha) \quad \mathit{BROAD}(\Diamond_{i\cap j}\Gamma, \alpha)}{\mathit{SEND}(\Gamma_{i\cap j}, \alpha)}\ \mathit{PORT2}$$

# Safety

## Theorem (Progress)

*If* $S := (\Gamma, t.i : \alpha)$*, then either* $S \mapsto S'$ *or exec*$(\alpha)$ *is the output value.*

## Theorem (Preservation)

*If* $S := (\Gamma, t.i : \alpha)$ *and* $S \mapsto S'$*, then* $S' := (\Gamma, t' : \alpha)$*.*

## Theorem (Type Safety)

*Safety is satisfied by transformations (according to the table in Definition 17) or by terminating expression (exec*$(\alpha)$*):*

1. *If* $S := (t.i : \alpha)$*, and* $S \mapsto S'$*, then* $S' := (t.i : \alpha)$*;*
2. *If* $S := (t.i : \alpha)$*, then either exec*$(\alpha)$ *is the output value or there is* $\alpha'$ *for* $S' := (t.i : \alpha')$ *s.t.* $S \mapsto S'$*.*

1 Background

2 Contextual Types with Multi-Modalities

3 The Operational Semantics

4 Conclusions

# Results

- Local soundness and completeness generalized from the expansion/reduction in the mono-modal case from [Primiero, 2012];
- Substitution on terms and truth predicates;
- Theorems for Strong Normalization and Confluence via the Operational Semantics;
- Theorems for the equivalence of $\Box_{i \cup j}$ to a CK-operator and $\Diamond_{i \cap j}$ to a DK-operator.

# Conclusions

- We have introduced a Computational Interpretation for a Multimodal Type-Theory with indexed and ordered Contexts;
- It treats modalities in a structurally different way than other modal type theories;
- It has an operational interpretation via a Procedural Semantics;
- It can be interpreted for Trusted Communications ([Primiero and Taddeo, 2012]).
- Further Work: working paper on a notion of Dependent Evidence in **ILP** and extensions to Semantics for Error States in Distributed Setting.

# References I



Artemov, S. and Bonelli, E. (2007).
The intensional lambda calculus.
In *Proceedings of the international symposium on Logical Foundations of Computer Science*, LFCS '07, pages 12–25, Berlin, Heidelberg. Springer-Verlag.


Bonelli, E. and Feller, F. (2009).
The logic of proofs as a foundation for certifying mobile computation.
In Artëmov, S. N. and Nerode, A., editors, *LFCS*, volume 5407 of *Lecture Notes in Computer Science*, pages 76–91. Springer.


Borghuis, T. and Feijs, L. (2000).
A constructive logic for services and information flow in computer networks.
*The Computer Journal*, 43(4):274–289.

# References II



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


Jia, L. and Walker, D. (2004).
Modal Proofs as Distributed Programs.
In *Programming Languages and Systems, ESOP2004*, volume 2986 of *Lectures Notes in Computer Science*. Springer Verlag.


Moody, J. (2003).
Modal logic as a basis for distributed computation.
Technical Report CMU-CS-03-194, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, USA.


Park, S. (2006).
A modal language for the safety of mobile values.
In *In Fourth ASIAN Symposium on Programming Languages and Systems*, pages 217–233. Springer.

# References III



Primiero, G. (2012).
A contextual type theory with judgemental modalities for reasoning from open assumptions.
*Logique & Analyse*, 220.


Primiero, G. and Taddeo, M. (2012).
A modal type theory for formalizing trusted communications.
*Journal of Applied Logic*, 10:92–114.