A judgemental modal type theory for data accessibility

Giuseppe Primiero

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

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

AILA Meeting
3rd, February, 2011 - Bologna – Italy

Outline

1. Conceptual Background
2. A modal contextual type theory with judgemental modalities
3. The Operational Semantics
4. Conclusions

From Constructive Modalities to Modal Type Theories

- Semantic modeling:
  - [Simpson(1994)];
  - [Bierman and de Paiva(2000)],[Alechina, Mendler, de Paiva(2001)];
  - *Lambda*5 and *Ml*5 implemented in [Murphy(2008)] and [Murphy, Crary, and Harper(2008)] for Grid Computing;

From Constructive Modalities to Modal Type Theories

- Semantic modeling:
  - [Simpson(1994)];
  - [Bierman and de Paiva(2000)],[Alechina, Mendler, de Paiva(2001)];
  - *Lambda*5 and *Ml*5 implemented in [Murphy(2008)] and [Murphy, Crary, and Harper(2008)] for Grid Computing;
- Constructive modalities and modal type theories:
  - [Pfenning and Davies(2001)];
  - [Nanevski, Pfenning, and Pientka(2008)];
  - Reason about distributed and staged computation: [Moody(2003)], [Davies and Pfenning(2001)], [Jia and Walker(2004)].

Resources and Locations via Types

- Type Theories for safe distributed computing
  - [Davies and Pfenning(2001)], [Jia and Walker(2004)]
  - Ability to represent heterogeneity w.r.t. properties, resources, devices, software, services;
  - Locally sound and complete modalities;
  - Type theoretical formulations / Natural Deduction / Sequent Calculi.

Resources and Locations via Types

- Type Theories for safe distributed computing
  - [Davies and Pfenning(2001)], [Jia and Walker(2004)]
  - Ability to represent heterogeneity w.r.t. properties, resources, devices, software, services;
  - Locally sound and complete modalities;
  - Type theoretical formulations / Natural Deduction / Sequent Calculi.
- A typed λ-calculus with stationary situations and flowing informations
  - [Borghuis, Feijs(2000)]
  - Interesting take on the representation of the order of commands;
  - Inspiring for the focus on recover of data from locations.

Resources and Locations via Types

- Type Theories for safe distributed computing
  - [Davies and Pfenning(2001)], [Jia and Walker(2004)]
  - Ability to represent heterogeneity w.r.t. properties, resources, devices, software, services;
  - Locally sound and complete modalities;
  - Type theoretical formulations / Natural Deduction / Sequent Calculi.
- A typed λ-calculus with stationary situations and flowing informations
  - [Borghuis, Feijs(2000)]
  - Interesting take on the representation of the order of commands;
  - Inspiring for the focus on recover of data from locations.
- A modal logic for local resources
  - [Park(2006)]
  - How to distinguish between transmission of safe values and safe code.

This contribution: Type Theory with judgmental modalities

- A type theory including modal operators with judgmental scope;
  - $\Box(A\ \mathit{true})$: "command execution for A is valid at every address";
  - $\Diamond(A\ \mathit{true})$: "command execution for A is valid at a given address".

This contribution: Type Theory with judgmental modalities

- A type theory including modal operators with judgmental scope;
  - $\Box(A\ true)$: "command execution for A is valid at every address";
  - $\Diamond(A\ true)$: "command execution for A is valid at a given address".

- Indexed multimodalities to localize data and express interaction of commands;
  - *contexts* describe networks in which code is executed;
  - *order* of assumptions is used to mimick the composition of commands.

Structure of the language

- polymorphic language: $\mathcal{K} : \{type, type_{inf}\}$
  - *type*: computations with complete instructional informations;
  - *type$_{inf}$*: admissible computational instructions (functional to execute a command).

Structure of the language

- polymorphic language: $\mathcal{K} : \{type, type_{inf}\}$
  - *type*: computations with complete instructional informations;
  - $type_{inf}$: admissible computational instructions (functional to execute a command).

- *Type*
  - *type*-constructors composed by listing, application, abstraction and pairing for $\wedge, \vee, \rightarrow, \forall, \exists$;
  - $\rightarrow$ as a λ-term presented *together with* one of its α-terms;
  - $a_i : A$ induces $\Box_i(A\ true)$:
  - computations that can be safely run everywhere;
  - $\Box_i(A\ true)$ induces $\Box_j(A\ true)$ for all i, j;

Structure of the language

- polymorphic language: $\mathcal{K} : \{type, type_{inf}\}$
  - *type*: computations with complete instructional informations;
  - $type_{inf}$: admissible computational instructions (functional to execute a command).

- *Type*
  - *type*-constructors composed by listing, application, abstraction and pairing for $\wedge, \vee, \rightarrow, \forall, \exists$;
  - $\rightarrow$ as a λ-term presented *together with* one of its α-terms;
  - $a_i : A$ induces $\Box_i(A\ true)$:
  - computations that can be safely run everywhere;
  - $\Box_i(A\ true)$ induces $\Box_j(A\ true)$ for all i, j;

- $Type_{inf}$
  - Admissibility defined from $\neg(A \rightarrow \bot)$ to $x : A$;
  - $\supset$ is composition by abstraction (admissible command at address);
  - $x_i : A$ induces $\Diamond_i(A\ true)$:
  - address-bounded computations;
  - the given location needs to be called upon to produce safely a value;

Language (1)

Definition (Terms)

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

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

Interpreting complete code (1)

Definition (Rules for *type*)

The rules for signed expressions in the kind *type* are:

$$\frac{a_i : A}{A\ type}\ \text{Type Formation} \qquad \frac{a_i : A \quad b_j : B}{(a_i, b_j) : A \wedge B}\ I\wedge$$

$$\frac{a_i : A}{l(a_i) : A \vee B\ true}\ Left I \qquad \frac{a_i : A \quad A\ true \vdash b_j : B}{a_i(b_j) : A \to B}\ \ I \to$$

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

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

$$\frac{a_i : A}{\neg A \to \bot}\ \ I\bot$$

Interpreting complete code (2)

Definition (Structural Rules)

$$\frac{}{\Gamma, a_i : A, \Delta \vdash A\ true.}\ \text{Premise Rule}$$

$$\frac{\Gamma \vdash B\ type \quad \Gamma \vdash A\ type}{\Gamma, a_i : A \vdash B\ type.}\ \text{Weakening}$$

$$\frac{\Gamma, |\ a_i : A, b_j : B \vdash C\ type \quad \Gamma \vdash b_j : B}{\Gamma, a_i : A \vdash C\ type.}\ \text{Contraction}$$

$$\frac{\Gamma, a_i : A, b_j : B \vdash C\ type}{\Gamma, b_j : B, a_i : A, \vdash C\ type}\ \text{Exchange}$$

Interpreting executable code (1)

Definition (Rules for $type_{inf}$)

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

$$\frac{\neg(A \rightarrow \bot)\ type \quad x_i : A}{A\ type_{inf}}\ Type_{inf}\ \text{Formation}$$

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

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

$$\frac{\lambda((a_{1-i}(b_j))A, B) \quad (b_j)[a_i := a]}{(a_i(b_j)) : A \rightarrow B}\ \alpha - conversion$$

Interpreting executable code (2)

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

$$\frac{}{\Gamma, x_i : A, \Delta \vdash A \ true^*} \text{ Hypothesis Rule}$$

$$\frac{\Gamma \vdash B \ type_{inf} \quad x_i : A \vdash A \ type_{inf}}{\Gamma \mid x_i : A \vdash B \ type_{inf}.} \text{ Weakening}$$

$$\frac{\Gamma \mid x_i : A, y_j : B \vdash C \ type_{inf} \quad \Gamma \vdash y_j : B}{\Gamma \mid x_i : A \vdash C \ type_{inf}} \text{ Contraction}$$

$$\frac{\Gamma \mid x_i : A \mid y_j : B \vdash C \ type_{inf}}{\Gamma \mid y_j : B \mid x_i : A, \vdash C \ type_{inf}} \text{ Exchange}$$

Introduction and Elimination for $\Box$

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

$$\frac{\Gamma_i \mid x_j : A \vdash A\ true^{*} \qquad \Box_i\Gamma, [x_j/a_j] : A \vdash A\ true}{\Box_{\mathcal{G}}\Sigma \vdash \Box_{\mathcal{G}}(A\ true)}\ I\Box$$

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

- I-$\Box$: if program for A requires broadcastble code executed at j, then A can be executed everywhere in network Σ;
- E-$\Box$: sends ($A\ true$) from i, j to $\mathcal{G}$ where it can be used to evaluate B.

Introduction and Elimination for $\Diamond$

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

$$\frac{\Gamma_i \mid x_j : A \vdash B\ true^*}{\Diamond_{\mathcal{G}}\Sigma \vdash \Diamond_{i,j}(B\ true)}\ I\Diamond$$

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

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

Code Mobility Rules

Definition (Broadcast)

Broadcacst is used to send to a specific address an exec command that can be executed everywhere in the network Σ.

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

Code Mobility Rules

Definition (Broadcast)

Broadcacst is used to send to a specific address an exec command that can be executed everywhere in the network Σ.

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

Definition (Global Access)

Global Access is the reverse function that calls from a specific address within network Σ a command that becomes executable at any address.

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

Properties

Definition (Admissible Rules)

$$\frac{x_i : A \vdash A\ true^*}{\Gamma, x_i : A, \Delta \vdash \Diamond_i(A\ true)}\ \text{Reflexivity}$$

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

Properties

Definition (Admissible Rules)

$$\frac{\Box_i\Gamma, a_j : A \vdash \Box_k(B\ true) \quad x_j : A \vdash A\ true^*}{\Box_i\Gamma, \Diamond_j(A\ true) \vdash \Diamond_k(B\ true)}\ \text{Common Seriality}$$

$$\frac{\Box_i\Gamma \vdash A\ true \quad \Diamond_j(A\ true)[x_i : A]}{\Box_i\Gamma, x_i : A \vdash \Diamond_j(A\ true)}\ \text{Convergence}$$

Properties (cnt'd)

Definition (Admissible Rules)

$$\frac{\Box_{\mathcal{G}}\Sigma \vdash \Box_k(A\ true) \quad \Box_{i,j}\Sigma \mid a_k : A \vdash \Box_{\mathcal{G}}(A\ true)}{\Box_{\mathcal{G}}\Sigma \vdash \Box_{i,j}(A\ true)}\ \textit{Upper Inclusion}$$

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

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

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

Syntax

Definition (Syntax of the Programming Language)

The syntax is defined by the following alphabet:

$$\textit{Types} := \alpha \mid \tau_1 \wedge \tau_2 \mid \tau_1 \vee \tau_2 \mid \tau_1 \to \tau_2 \mid \tau_1 \supset \tau_2.$$

$$\textit{Terms} := x_i \mid a_i \quad \textit{constants}, \textit{variables}.$$

$$\textit{Pairs} := (a_i, b_j).$$

$$\textit{Functions} := \textit{exec}(\alpha) \mid \textit{run}_i(\alpha) \mid \textit{synchro}_j(\textit{run}_i(\alpha)).$$

$$\textit{Remote Operations} := \textit{GLOB}(\Box\Gamma, \alpha_{i\cup j}) \mid \textit{BROAD}(\Diamond\Gamma, \alpha_{i\cap j}).$$

$$\textit{Portable Code} := \textit{RET}(\Gamma, \alpha_{i\cup j}) \mid \textit{SEN}(\Gamma, \alpha_{i\cap j}).$$

$$\textit{Contexts} := \Gamma_i, \Box_i\Gamma; \Diamond_i\Gamma.$$

Syntax (2)

Definition (Typing Rules)

$$\frac{}{\Gamma, a_i : \alpha, \Delta \vdash exec(\alpha).}\ \text{global} \qquad \frac{}{\Gamma, x_i : \alpha, \Delta \vdash run_i(\alpha).}\ \text{local}$$

$$\frac{a_i : \alpha \quad b_j : \beta}{run_{i \cup j}(\alpha \times \beta)}\ I\wedge \qquad \frac{(a_i, b_j) : \alpha \times \beta}{exec(\alpha)}\ E \wedge (1)$$

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

$$\frac{x_i : \alpha \vdash run_i(\alpha) \quad x_i(b_j) : \alpha \supset \beta}{synchro(b_j(run_i(\alpha)))}\ synchro$$

Syntax (3)

Definition

$$\frac{\Gamma_i, x_j : \alpha \vdash run_i(\alpha) \quad \Box\Gamma, x_j(a_j) : \alpha \vdash exec(\alpha)}{GLOB(\Box\Gamma, \alpha_{i\cup j})}\ RPC1$$

$$\frac{\Gamma_i, x_j : \alpha \vdash run_i(\alpha) \quad \Diamond_{i,j}\Sigma \vdash run_{i,j}(\alpha)}{BROAD(\Diamond\Sigma, \alpha_{i\cap j})}\ RPC2$$

$$\frac{\Box_i\Gamma, a_j : \alpha \vdash run_{i\cup j}(\alpha) \quad GLOB(\Box\Gamma, \alpha_{i\cup j})}{RET(\Gamma, \alpha_{i\cup j})}\ PORT1$$

$$\frac{\Box_i\Gamma, \Diamond_j\Delta \vdash run_{i\cap j}(\alpha) \quad BROAD(\Diamond\Sigma, \alpha_{i\cap j})}{SEND(\Gamma, \alpha_{i\cap j})}\ PORT2$$

Operational Semantics

Definition (Operational Model)

Networks $\mathcal{N} := (\mathcal{I}, \mathcal{L})$.

Process Environments $\mathcal{L} := (l.i \mapsto e) \mid i \in \mathcal{I}; l, e \in \mathcal{T}$.

Terms $\mathcal{T} :=\mid synchro(.(.)) \mid run_i(\alpha) \mid exec(\alpha) \mid GLOB(.) \mid BROAD(.) \mid .$

Contexts $\mathcal{C} := \Gamma \mid \circ\Gamma \mid (\Gamma, e) \mid .$

Operational Semantics (II)

Rewriting rules for states transition:

	$\mathcal{L} \mapsto \mathcal{L}'$
run	$[\Gamma]x_i \mapsto [\Diamond\Gamma]run_i(\alpha)$
exec	$[\Gamma]a_i \mapsto [\Box\Gamma]exec(\alpha)$
$\rightarrow$	$[\Gamma]exec(\alpha) \vdash b_j \mapsto [\Box\Gamma]synchro(b_j(exec(\alpha))$
$\supset$	$[\Gamma]run_i(\alpha) \vdash b_j \mapsto [\Diamond\Gamma]synchro(b_j(run_i(\alpha))$
$\wedge$	$[\Gamma]run_i(\alpha), run_j(\beta) \mapsto [\Box\Gamma]exec(\alpha, \beta)$
$\Box 1$	$[\Gamma] \vdash run_i(\alpha) \mapsto GLOB[\Box\Gamma, \alpha]$
$\Box 2$	$[\Box_i\Gamma]exec(\alpha) \mapsto RET[\Gamma, \alpha_{i\cup j}]$
$\Diamond 1$	$[\Gamma] \vdash run_i(\alpha) \mapsto BROAD[\Diamond\Gamma, \alpha_{i\cap j}]$
$\Diamond 2$	$[\Diamond\Gamma] \vdash run_{i,j}(\alpha) \mapsto SEND[\Gamma, \alpha_{i\cap j}]$

Evaluation

- Termination requires all closed expressions (*exec*(α) and $[\Box\Gamma]\alpha$);
- Occurrences of run_i and $\Diamond_i$ require preservation of indices;
- Evaluation on contexts proceeds on the ordering induced by $i < j$;
- A transition $\mathcal{L} \mapsto \mathcal{L}'$ consists of
  - decomposing $\mathcal{L}$ into an evaluation context (if present) and an instruction;
  - evaluation of the context and execution of the instruction;
  - replacement of intruction execution in one of the rules to obtain $\mathcal{L}'$.

Safety

Theorem (Type Safety)

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

Theorem (Preservation)

If $[\Gamma]e : \alpha$ *for* $\mathcal{L} := (l.i \mapsto e)$, *and* $\mathcal{L} \mapsto \mathcal{L}'$, *then there is* $[\Box\Gamma]e' : \alpha$ *for* $\mathcal{L}' := (l.i \mapsto e')$

Theorem (Progress)

If $[\Box\Gamma]e : \alpha$ *for* $\mathcal{L} := (l.i \mapsto e)$, *then either* $\mathcal{L} \mapsto \mathcal{L}'$ *or* $exec(\alpha)$ *is the output value.*

Conclusions

- A Computational Interpretation for a Multimodal Type-Theory with indexed and ordered Contexts;
- Corresponding Epistemic Interpretation for Trusted Communications;
- Models:
  - Weakening of the poset $\{1, 0\}$ that satisfies inhabitness and intensional identity;
  - $type_{inf}$ admits undefinability, preserves only symmetricity; inhabitness is not guaranteed ('super-modest types');
  - Semantics of $cKT_{\Box,\Diamond}$ obtained by a composed set of (non-standard) Kripke models $\mathcal{M}^{(\mathcal{L}^{ver} \cup \mathcal{L}^{inf})}$.

References

N. Alechina, M. Mendler, V. de Paiva, and E. Ritter.
Categorical and Kripke Semantics for Constructive S4 Modal Logic.
In *Proceedings of the 15th International Workshop on Computer Science Logic*, volume 2142 of *Lecture Notes In Computer Science*, pages 292 – 307, 2001.

G.M. Bierman and V. de Paiva.
On an intuitionistic modal logic.
Studia Logica, (65):383–416, 2000.

T. Borghuis and L.M.G. Feijs.
A constructive logic for services and information flow in computer networks.
The Computer Journal, pp.274–289, vol.43, n.4, 2000.

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

References

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

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

T. Murphy.
Modal Types for Mobile Code.
PhD thesis, School of Computer Science, Carnegie Mellon University, 2008.
CMU-CS-08-126.

T. Murphy, K. Crary, and R. Harper.
Type-Safe Distributed Programming with ML5, volume 4912 of *Lectures Notes in Computer Science*, pages 108–123.
Springer Verlag, 2008.

References

A. Nanevski, F. Pfenning, and B. Pientka.
Contextual modal type theory.
ACM Transactions on Computational Logic, 9(3):1–48, 2008.

F. Pfenning and R. Davies.
A judgemental reconstruction of modal logic.
Mathematical Structures in Computer Science, 11:511–540, 2001.

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

G. Primiero.
Constructive contextual modal judgments for reasoning from open assumptions.
In *Proceedings of the Computability in Europe Conference*, 2010.

A.K. Simpson.
The Proof Theory and Semantics of Intuitionistic Modal Logic.
PhD thesis, University of Edinburgh, College of Science and