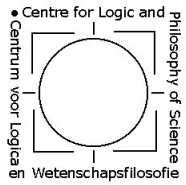
Procedural Semantics for a Modal Type System

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/

CLMPS, Nancy, France
July, 2011

Outline

Modalities for localized computations

- Procedural Semantics with Modalities for Contextual (localized) Computing;
- designed from a multi-modal type system with a BHK semantics Martin-Löf's style with Proofs-as-Programs (at IMLA11 on Saturday);
- localization of processes to represent distributed computing;
- rules for connectives intepret composition of processes;
- modal rules interpret interaction of code at locations (mobility).

Other Extended Semantics

- (Modal Types based) Dynamic Semantics in terms of a big-step evaluation relation in [Murphy, 2008];
- (Modal) Network Operational Semantics in [Jia and Walker, 2004] and [Park, 2006];
- (BHK-inspired) Operational Semantics of expressions encoding proofs in LP in terms of global computation in [Artemov and Bonelli, 2007];

1 Localized Curry-Howard Semantics

2 Operational Semantics for our Modal TT

3 Conclusions

Semantics with indexed modal types

- $a_i : \alpha$ expresses the existence of a program valid at location i of type α;
- $\Gamma_i \vdash \alpha$ is the sequence of computational steps valid at location i that validate a program of type α;
- the meaning of program α is given by explaining how steps in Γ_i are obtained and where they hold;
- Use modalities in $\circ_i \Gamma \vdash \alpha$ to express local/global validity of program/processes.

Translation to an Operational Semantics

- Provide a syntax-oriented inductively defined semantics reflecting the original BHK proof interpretation;
- Define the behavior of programs by transition relations among states of the corresponding (abstract) machine;
- Define the valid transitions as a set of inference rules to give a composite piece of syntax in terms of the transitions of its components;
- Enrich the language with locations and values/code mobility operations.

Language

Definition (Syntax of the Programming Language)

The syntax is defined by the following alphabet:

$$\textit{Types} := \alpha \mid \alpha \times \beta \mid \alpha \sqcup \beta \mid \alpha \to \beta \mid \alpha \supset \beta$$

$$\textit{Terms } \mathcal{T} := x_i \mid a_i$$

$$\textit{Functions} := \textit{exec}(\alpha) \mid \textit{run}_i(\alpha) \mid \textit{run}_{i\cup j}(\alpha \cdot \beta) \mid \textit{run}_{i\cap j}(\alpha \cdot \beta) \mid \textit{synchro}_j(\beta(\textit{exec}(\alpha)))$$

$$\textit{Contexts } \mathcal{C} := \Delta_i \mid \Gamma_i \mid \circ_{i,j}\Gamma$$

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

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

Conventions

- *exec* refers to the output of a running program; can take *any* index;
- *run* is the procedural representation of a function; occurs with a single index when referring to a single process;
- *run* takes compositions of indices when it composes processes: $\cup$ for executability at either location; $\cap$ for executability at ordered intersection;
- *synchro* computes a function using *exec* of some value it depends from (Call by Value): semantic equivalent for β-reduction or function application;
- Introduction Rules for Modalities correspond to Rules for Remote Operations; Eliminations Rules to Rules for Portable Code.

Operational Semantics

Definition (State Machine)

A state machine $S \in \mathcal{S}$

$$\mathcal{S} := (\mathcal{C}, t.i : \alpha) \mid \mathcal{C} \in \textit{Context};\ t \in \mathcal{T}; i \in \mathcal{I}; \alpha \in \textit{Types}$$

is an occurrence of an indexed typed term in context.

Computational Rules

Definition (Typing Rules)

$$\frac{}{\Delta_i, a_i : \alpha \vdash exec(\alpha)}\ \text{global} \qquad \frac{}{\Gamma_i, x_i : \alpha; \Delta_i \vdash run_i(\alpha)}\ \text{local}$$

$$\frac{a_i : \alpha \quad b_j : \beta}{run_{i \cup j}(\alpha \times \beta)}\ I\times \qquad \frac{a_i : \alpha}{run_i(\alpha \sqcup \beta)}\ I\sqcup$$

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

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

The Modal Extension

Definition (Modal Judgements)

The set of modal judgements $\mathcal{M}$ for any $i \in \mathcal{G}$ is defined by the following modal formation rules:

$$\frac{exec(\alpha)}{\Box_i \Gamma \vdash \alpha}\ \Box - \text{Formation} \qquad \frac{\Gamma_i \vdash run_i(\alpha)}{\Diamond_i \Gamma \vdash \alpha}\ \Diamond - \text{Formation}$$

Modal Rules

Definition

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

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

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

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

Operational Semantics

Definition (Operational Model)

An indexed transition system (also called Network)

$$\textit{Networks}\ \mathcal{N} := (\mathcal{S}, \mapsto, \mathcal{I})$$

is a triple where $\mathcal{S}$ is a set of states, $\mathcal{I}$ is a set of indices and $\mapsto (\mathcal{S} \times \mathcal{I} \times \mathcal{S})$ is a ternary relation of indexed transitions. 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 state typing rules.

Evaluation

Rewriting rules for states transition:

	$S \mapsto S'$
run	$(\Gamma_i, x_i : \alpha) \mapsto (\Diamond_i \Gamma, \mathit{run}_i(\alpha))$
exec	$(\Gamma_i, a_i : \alpha) \mapsto (\Box_i \Gamma, \mathit{exec}(\alpha))$
$\rightarrow$	$(\Gamma_i, \mathit{exec}(\alpha) \vdash b_j) \mapsto (\Box_i \Gamma, \mathit{run}_{i \cup j}(\alpha \rightarrow \beta))$
$\supset$	$(\Gamma_i, \mathit{run}_i(\alpha) \vdash b_j) \mapsto (\Box_i \Gamma, \mathit{synchro}(b_j(\mathit{exec}(\alpha))))$
$\times$	$(\Gamma_i, \mathit{exec}(\alpha), \mathit{exec}(\beta)) \mapsto (\Box_i \Gamma, \mathit{run}_{i \cup j}(\alpha \times \beta))$
$\sqcup$	$(\Gamma_i, \mathit{exec}(\alpha)) \mapsto (\Box_i \Gamma, \mathit{run}_i(\alpha \sqcup \beta))$
$\Box 1$	$(\Gamma_i, \mathit{exec}(\alpha)) \mapsto (\mathit{GLOB}(\Box_{i \cup j} \Gamma, \alpha))$
$\Box 2$	$(\Box_i \Gamma, \alpha_{i \cup j}) \mapsto (\mathit{RET}(\Gamma_{i \cup j}, \alpha))$
$\Diamond 1$	$(\Gamma_i, \mathit{run}_i(\alpha)) \mapsto (\mathit{BROAD}(\Diamond_{i \cap j} \Gamma, \alpha))$
$\Diamond 2$	$(\Diamond_i \Gamma, \alpha_{i \cap j}) \mapsto (\mathit{SEND}(\Gamma_{i \cap j}, \alpha))$

Semantic Validity

Definition (Semantic Expressions)

- Evaluation defines strong typing (normalisation) by reduction to expressions $(\Box_i\Gamma, \mathit{exec}(\alpha))$ and $\mathit{GLOB}(\Box_i\Gamma, \alpha)$.
- Expressions $(\Gamma_i, \mathit{run}_i(\alpha))$ and $\mathit{BROAD}(\Diamond_i\Gamma, \alpha)$ are admissible procedural steps but may fail to produce a safe value (when called upon at wrong addresses).
- This makes (only) the following expressions valid (safely evaluated):

$$\frac{}{a_i : \alpha \ \mathit{value}} \qquad \frac{}{\Box_i\Gamma, \alpha \ \mathit{value}}$$

Some Results

Theorem (Type Safety)

Safety is satisfied by transformations (according to the table of rewriting rules) or by terminating expression (exec(α))

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

Proof.

By (*i*) evaluation steps preserve typing. By (*ii*) closed expressions induce overall execution, hence are safe processes. □

Some Results

Theorem (Preservation)

If $S := (\Gamma_i, t.i : \alpha)$, *then* $S \mapsto S'$ *for some* $S' := (\Box\Gamma'_i, t'.i : \alpha')$.

Proof.

By induction on $\alpha, \alpha' \in$ *Types* and the structure of Γ_i and by the Safety Theorem for $S \mapsto S'$. □

Some results

Theorem (Progress)

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

Proof.

By induction on $\alpha \in$ *Types* using the properties induced by $\Box_i\Gamma$; by Safety Theorem for $S \mapsto S'$ and using the Preservation Theorem as last step. $\square$

1. Localized Curry-Howard Semantics

2. Operational Semantics for our Modal TT

3. Conclusions

Conclusions

- A Computational Interpretation for a Multimodal Type-Theory with indexed and ordered Contexts;
- Operational Interpretation by a Procedural Semantics for Mobile Code and Mobile Values;
- Corresponding Epistemic Interpretation for Trusted Communications with definitions of DK/CK;

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.


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.


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

References II


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.