Accessibility, Matching, Use: on limitations of computing in a distributed setting

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/

Colloquium Logicum, Paderborn, 14th September 2012

Logical Approaches to Distributed Computing

- Task: study of formal issues in (distributed) computing;

Logical Approaches to Distributed Computing

- Task: study of formal issues in (distributed) computing;
- Proofs-as-programs isomorphism: a proof of a proposition derivable in a context corresponds to a program for a specification executable in a network;

Logical Approaches to Distributed Computing

- Task: study of formal issues in (distributed) computing;
- Proofs-as-programs isomorphism: a proof of a proposition derivable in a context corresponds to a program for a specification executable in a network;
- Modalities: more control on resources, their location and accessibility.

Logical Approaches to Distributed Computing

- Task: study of formal issues in (distributed) computing;
- Proofs-as-programs isomorphism: a proof of a proposition derivable in a context corresponds to a program for a specification executable in a network;
- Modalities: more control on resources, their location and accessibility.
  - [Bierman and de Paiva, 2000],[Alechina et al., 2001]: *CS*4;
  - [Murphy, 2008] and [Murphy et al., 2008]: *Lambda*5 for Grid Computing;
  - [Borghuis, 1994], [Borghuis, 1998], [Pfenning and Wong, 1995]: modal λ-calculi;
  - [Park, 2006]: safe values vs. safe code.
  - [Blass and Gurevich, 2010]: Primal Infon Logic.
  - [Bonelli and Feller, 2009]: code and certificate development;
  - [Artemov and Bonelli, 2007]: operational interpretation for remote calls.

Some restrictions of applied computing

Some Interesting issues in safe distributed computing ([Primiero, 2011], [Primiero, forth]).

- Polymorphism
- Resources
- Global vs. Local Validity
- Mobility
- Error-states

Language

Definition (Syntax)

The syntax is defined by the following alphabet:

$$\sigma(\texttt{Specs}) := \{\alpha \mid \alpha \times \beta \mid \alpha + \beta \mid \alpha \to \beta \mid \alpha \supset \beta\}$$

$$\pi(\texttt{Programs}) := \{x_i \mid a_i, \text{ for } i \in \iota\}$$

$$\iota(\texttt{Locations}) := \{1, \ldots, n\}$$

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

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

Model

Syntactic expressions are evaluated in a model defined by states of an abstract machine.

Definition (Operational Model)

$$S := (\mathcal{C}, t.i : \alpha) \mid \mathcal{C} \in \texttt{Contexts}; t \in \texttt{Programs}; i \in \texttt{Locations}; \alpha \in \texttt{Specs}$$

is an occurrence of an indexed typed term in context evaluated by transition to some S' in a `Network`

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

Polymorphism

Not all formulas are of the same kind (i.e. not all programs are executed in the same way): *exec* function for terms of values; *run* function for terms of code.

Definition

	$S \mapsto S'$
`run`	$(\Gamma_i, x_i : \alpha) \mapsto (\Diamond_i \Gamma, \mathit{run}_i(\alpha))$
`exec`	$(\Gamma_i, a_i : \alpha) \mapsto (\Box_i \Gamma, \mathit{exec}(\alpha))$
`corun`	$(\Gamma_i, \mathit{run}_i(\alpha) \vdash b_j : \beta) \mapsto (\Box_i \Gamma, \mathit{run}_{i \cap j}(\alpha(\beta))$
`coexec`	$(\Gamma_i, \mathit{exec}(\alpha) \vdash b_j : \beta) \mapsto (\Box_i \Gamma, \mathit{run}_{i \cup j}(\alpha(\beta))$
`synchro`	$(\Box_i \Gamma, \mathit{run}_{i \cup j}(\alpha(\beta)) \mapsto (\Box_i \Gamma, \mathit{synchro}_j(\beta(\mathit{exec}(\alpha))))$

Resources

Formulas hold in contexts (i.e. programs are executed in networks):

Definition

The set of conditions for formulas are inductively defined by valid and true assumptions:

$$\texttt{Contexts}(\texttt{DataStacks}) := \{\Gamma_i \mid \circ_i \Gamma\}, \circ = \{\Box, \Diamond\}$$

Local vs Global

Not all formulas are valid in the same way (i.e. not all programs are executable under the same conditions):

- *GLOB*($\Box_{i\cup j}\Gamma, \alpha$): α is output everywhere satisfied in i, j;
- *BROAD*($\Diamond_{i\cap j}\Gamma, \alpha$): α is code valid by accessing $i \cap j$;

Mobility: Call-by-value

Rules for $\Box$ express mobility of globally valid values:

Definition

	$S \mapsto S'$
$\Box 1$	$(\Box_i \Gamma, \mathit{exec}(\alpha)) \mapsto (\mathit{GLOB}(\Box_{i \cup j} \Gamma, \alpha))$
$\Box 2$	$(\Box_{i \cup j} \Gamma, \alpha) \mapsto (\mathit{RET}(\Gamma_{i \cup j}, \alpha))$

- $\Box 1$: takes value α and make it available everywhere in network $\mathcal{G} = \{i, j\}$ (induces an operational interpretation as Remote Procedure Call);
- $\Box 2$: sends value α from i, j to $\mathcal{G}$.

Mobility: Call-by-name

Rules for $\Diamond$ express mobility of locally valid code:

Definition

	$S \mapsto S'$
$\Diamond 1$	$(\Diamond_i \Gamma, \mathit{run}_j(\alpha)) \mapsto (\mathit{BROAD}(\Diamond_{i \cap j} \Gamma, \alpha))$
$\Diamond 2$	$(\Diamond_{i \cap j} \Gamma, \alpha) \mapsto (\mathit{SEND}(\Gamma_{i \cap j}, \alpha))$

- $\Diamond 1$: take code for α at j and make it valid at $i \cap j$; it constructs a return value for a RPC;
- $\Diamond 2$: from local code for α, send it to $i \cap j$.

Safety

Theorem (Progress)

If $S := (\Gamma, t.i : \alpha)$*, then either* $S \mapsto S'$ *or exec*(α) *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 or by terminating expression (exec(α)*):*

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

Errors

With such a machinery it is possible to represent incorrect states, with application to analysis of security breaching and failing communications:

- Errors in Access
- Errors in Matching
- Errors in Use

Non-safe states: Error Access

An error state obtained by the access of wrong data/locations in the network.

Definition

	$S \mapsto S'$
`access`	$(\Gamma_i, access@_i(t)) \mapsto ((\Gamma_i, fail@_i(t))$
`fail`	$((\Gamma_i, fail@_i(t)) \mapsto (\Gamma_i, error(\tau))$

Non-safe states: Error Matching

An error state obtained by the execution of the wrong program (no successive state, no final state *exec*, no well-typed state):

Definition

	$S \mapsto S'$
$\texttt{access}_{\texttt{with}}$	$(\Gamma_i, access@_i(t')FROM(t)) \mapsto ((\Gamma_i, exec(\tau)WITH(t'))$
$\texttt{fail}_{\texttt{with}}$	$((\Gamma_i, exec(\tau)WITH(t')) \mapsto (\Gamma_i, error(\tau)WITH(t'))$

Non-safe states: Wrong Use

An error state obtained by the execution of the wrong rule:

Definition

	$S \mapsto S'$
$\texttt{access}_{\texttt{from}}$	$(RET(\Gamma_{i \cap j}, \tau) \mapsto \Gamma_i access@_j(t) FROM(\tau) \vdash fail@_j(\upsilon)$
$\texttt{error}_{\texttt{with}}$	$fail@_j(\upsilon) \mapsto (\Gamma_i, synchro_j(\tau(error(\upsilon) WITH(\tau)))$

Non-safe states: Wrong Use (II)

An error state obtained by the execution of a program for the wrong goal:

Definition

	$S \mapsto S'$
$\texttt{access}_{\texttt{from}}$	$(RET(\Gamma_{i \cap j}, \tau) \mapsto \Gamma_i access@_j(t) FROM(\tau) \vdash fail@_j(\upsilon)$
$\texttt{error}_{\texttt{with}}$	$fail@_j(\upsilon) \mapsto (\Gamma_i, synchro_j(\tau(error(\upsilon) WITH(\tau)))$

Open Questions and Further Research

- Extend the semantics with functions for determining error cases and recovering correct ones; does soundness still hold?
- Simulate implementations (e.g. deadlock cases, followed by resolving priority functions); it offers a formal and technical study of malfunctioning in computational artifacts;
- Use the local properties $\texttt{run}, \Diamond$ on processes to model unsafe and uncertified programs.

(Philosophical) Conclusions

- In applied computing a novel notion of correctness is at stake, based on practical limitations and constraints on computing;
- It offers new insights into logical validity: its model differs from both standard realistic truth-preserving consequence relation and anti-realistic knowledge-preserving validity relation;
- Formal conditions for failing (distributed) processes reveal a new perspective on the notion of logical system and its limits.

References I


Alechina, N., Mendler, M., de Paiva, V., and Ritter, E. (2001).
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.


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.


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

References II


Blass, A. and Gurevich, Y. (2010).
Hilbertian Deductive Systems, Infon Logic and Datalog.
Bulletin of the European Association for Theoretical Computer Science, 102:122–150.


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. (1994).
Coming to Terms with Modal Logic: On the interpretation of modalities in typed lambda calculus.
PhD thesis, Eindhoven University of Technology.


Borghuis, T. (1998).
Modal pure type systems: Type theory for knowledge representation.
Journal of Logic, Language, and Information, 7(3):pp. 265–296.

References III


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


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


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.


Pfenning, F. and Wong, H. C. (1995).
On a modal λ-calculus for $s4*$.
In *Proceedings of the Eleventh Conference on Mathematical Foundations of Programming Sematics*. Elsevier.

References IV


Primiero, G. (2011).
A multi-modal type system and its procedural semantics for safe distributed programming.
In *Intuitionistic Modal Logic and Applications Workshop (IMLA11)*, Nancy.