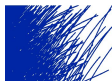


Formalizing Correctness & Interaction by Modal Types

Giuseppe Primiero

FWO – Research Foundation Flanders
Centre for Logic and Philosophy of Science, Ghent University
Information Ethics Group - Oxford



Giuseppe.Primiero@ugent.be

<http://www.philosophy.ugent.be/giuseppeprimiero/>

October 4th, 2010

European Conference on Computing and Philosophy
Philosophy of Computer Science Track

- 1 Introduction
- 2 Polymorphism and Modal Types
- 3 Interaction in Contextual Computing

- 1 Introduction
- 2 Polymorphism and Modal Types
- 3 Interaction in Contextual Computing

The background

2 major issues in the Philosophy of Computer Science (since [De Millo et al.(1979)]):

- **Correctness:** *can computer systems satisfy correctly their designers aim?*
- **Interaction:** *what role have feedback and user-oriented properties in program execution/evaluation?*

The background

2 major issues in the Philosophy of Computer Science (since [De Millo et al.(1979)]):

- **Correctness:** *can computer systems satisfy correctly their designers aim?*
- **Interaction:** *what role have feedback and user-oriented properties in program execution/evaluation?*

The background

2 major issues in the Philosophy of Computer Science (since [De Millo et al.(1979)]):

- **Correctness:** *can computer systems satisfy correctly their designers aim?*
- **Interaction:** *what role have feedback and user-oriented properties in program execution/evaluation?*

Correctness

Primiero @ ECAP2009:

- **Syntactic correctness:** *is it possible to formulate correct structural procedures to satisfy given specifications?*
 - 1 an appropriate language: modal dependent types
 - 2 embedded operational semantics + treatment of information sources;
 - 3 limits of correctness: decidability;
 - 4 internal vs. external levels of failure.

Correctness

Primiero @ ECAP2009:

- **Syntactic correctness:** *is it possible to formulate correct structural procedures to satisfy given specifications?*
 - 1 an appropriate language: modal dependent types
 - 2 embedded operational semantics + treatment of information sources;
 - 3 limits of correctness: decidability;
 - 4 internal vs. external levels of failure.

Interaction

Extension of today's talk:

- **Syntactic interaction:** *is it possible to formulate interactive processes as algorithms?*
 - 1 same functional interpretation;
 - 2 reading interaction as a property for process execution and metadata control;
 - 3 crucial properties: typology and location of data.

Interaction

Extension of today's talk:

- **Syntactic interaction:** *is it possible to formulate interactive processes as algorithms?*
 - 1 same functional interpretation;
 - 2 reading interaction as a property for process execution and metadata control;
 - 3 crucial properties: typology and location of data.

Interaction: the background

Interaction goes beyond algorithms and computability;

- [Copeland(1997)], [Copeland(2001)]: **Coupled Turing Machines**: TMs with additional input lines for accepting data from the world during computation: implement non-computable functions (addition over reals);
- [Wegner(1997)], [Wegner(1998)]: **Interaction Machines**: TMs with addition of input/output streams that support dynamic interaction with external environment:
 - 1 Input for Interactive Machines are not reducible to finite tapes;
 - 2 Open systems \approx Interaction Machines vs. Closed systems \approx Algorithmic Machines;
 - 3 Interactivness is non-monotonic.

Interaction: the background

Interaction goes beyond algorithms and computability;

- [Copeland(1997)], [Copeland(2001)]: **Coupled Turing Machines**: TMs with additional input lines for accepting data from the world during computation: implement non-computable functions (addition over reals);
- [Wegner(1997)], [Wegner(1998)]: **Interaction Machines**: TMs with addition of input/output streams that support dynamic interaction with external environment:
 - 1 Input for Interactive Machines are not reducible to finite tapes;
 - 2 Open systems \approx Interaction Machines vs. Closed systems \approx Algorithmic Machines;
 - 3 Interactiviness is non-monotonic.

Interaction: the background (2)

- Paraconsistent (Goldin, Wegner (2002)), Pluralist approaches;
- Super-recursive algorithms ([Burgin(2005)]);
- Concurrent Computing/ time and time-dependent spatial distribution ([Milner(1989)]; [Wegner(1998)]);
- Open algorithms: e.g. weak Turing machines with infinite input to the left and right of the actual dataword (Watanabe, Neary & Woods).

Contextual Computing

Thesis (Contextual Computing)

Interaction can be reduced to a property of processes within a contextual approach to computing s.t. it admits:

- *logical distinction among data (data are not all the same);*
- *metadata (the when/how/where of data is relevant);*
 - *local conditions on network (processes occur as events);*
 - *originating locations (processes are user originated events);*
- *restricted well-formedness and termination (not every process satisfies these properties).*

Contextual Computing

Thesis (Contextual Computing)

Interaction can be reduced to a property of processes within a contextual approach to computing s.t. it admits:

- *logical distinction among data (data are not all the same);*
- *metadata (the when/how/where of data is relevant);*
 - *local conditions on network (processes occur as events);*
 - *originating locations (processes are user originated events);*
- *restricted well-formedness and termination (not every process satisfies these properties).*

Contextual Computing

Thesis (Contextual Computing)

Interaction can be reduced to a property of processes within a contextual approach to computing s.t. it admits:

- *logical distinction among data (data are not all the same);*
- *metadata (the when/how/where of data is relevant);*
 - *local conditions on network (processes occur as events);*
 - *originating locations (processes are user originated events);*
- *restricted well-formedness and termination (not every process satisfies these properties).*

Contextual Computing

Thesis (Contextual Computing)

Interaction can be reduced to a property of processes within a contextual approach to computing s.t. it admits:

- *logical distinction among data (data are not all the same);*
- *metadata (the when/how/where of data is relevant);*
 - *local conditions on network (processes occur as events);*
 - *originating locations (processes are user originated events);*
- *restricted well-formedness and termination (not every process satisfies these properties).*

Interaction: the contextual setting

processes interaction via data **accessibility** and **categorization**:

- 1 types with full information guarantee termination and execution of distributed commands;
- 2 admitted types carry *external information* about (mutable) conditions for termination;
- 3 validity relations are defined over distinct user-states.

contexts allocate metadata (\neq infinite input):

- 1 needed processes at execution;
- 2 + revisable external data (when/how/where of processes);
- 3 = non-monotonic reductions, in the scope of algorithmic computation.

Interaction: the contextual setting

processes interaction via data **accessibility** and **categorization**:

- 1 types with full information guarantee termination and execution of distributed commands;
- 2 admitted types carry *external information* about (mutable) conditions for termination;
- 3 validity relations are defined over distinct user-states.

contexts allocate metadata (\neq infinite input):

- 1 needed processes at execution;
- 2 + revisable external data (when/how/where of processes);
- 3 = non-monotonic reductions, in the scope of algorithmic computation.

- 1 Introduction
- 2 Polymorphism and Modal Types**
- 3 Interaction in Contextual Computing

Dependent Types: some known facts

- **Curry-Howard Isomorphism**: propositions-as-types and proofs-as-programs identities;
- **Dependent Types**: extension to the first-order setting, functional language (MLTT; LF);
- The **program-meets-specification** variant: dependency as routine-subroutines relation;
 - **Requirements**: wellformedness of values and termination of terms (β - η -conversions)

Overview on the Formal Language

- Semantics for IML:
 - [Simpson(1994)];
 - [Bierman and de Paiva(2000)], [Alechina, Mendler, de Paiva(2001)];
 - *Lambda5* and *M/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)];

Overview on the Formal Language

- Semantics for IML:
 - [Simpson(1994)];
 - [Bierman and de Paiva(2000)], [Alechina, Mendler, de Paiva(2001)];
 - *Lambda5* and *M/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)];

Extension to Modal Types

([Primiero(2010)];[Primiero(2010b)])

- 1 type-inhabitation is reconsidered for extensions of contextual types:
 - \square : validity under all executions; termination guaranteed;
 - \diamond : validity under restricted conditions; termination not guaranteed.
- 2 Contexts:
 - contain localized data and express interaction of commands;
 - describe (variable) resources of networks;
 - structured to mimick the composition of commands.

Extension to Modal Types

([Primiero(2010)];[Primiero(2010b)])

- 1 type-inhabitation is reconsidered for extensions of contextual types:
 - \square : validity under all executions; termination guaranteed;
 - \diamond : validity under restricted conditions; termination not guaranteed.
- 2 Contexts:
 - contain localized data and express interaction of commands;
 - describe (variable) resources of networks;
 - structured to mimick the composition of commands.

The language

- polymorphic language: $\mathcal{K} : \{type, type_{inf}\}$
- *Type*: computations with complete instructional informations.
 - *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 $\square_i(A \text{ true})$: computations that can be safely run everywhere;
- *Type_{inf}*: computational instructions admissible to execute a command.
 - Admissibility defined from local execution;
 - $x_i : A$ induces $\diamond_i(A \text{ true})$;
 - address-bounded computations: the given location needs to be called upon to produce safely a value;

The language

- polymorphic language: $\mathcal{K} : \{type, type_{inf}\}$
- *Type*: computations with complete instructional informations.
 - *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 $\square_i(A \text{ true})$: computations that can be safely run everywhere;
- *Type_{inf}*: computational instructions admissible to execute a command.
 - Admissibility defined from local execution;
 - $x_i : A$ induces $\diamond_i(A \text{ true})$;
 - address-bounded computations: the given location needs to be called upon to produce safely a value;

The language

- polymorphic language: $\mathcal{K} : \{type, type_{inf}\}$
- *Type*: computations with complete instructional informations.
 - *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 $\square_i(A \text{ true})$: computations that can be safely run everywhere;
- *Type_{inf}*: computational instructions admissible to execute a command.
 - Admissibility defined from local execution;
 - $x_i : A$ induces $\diamond_i(A \text{ true})$;
 - address-bounded computations: the given location needs to be called upon to produce safely a value;

Modal Language

- 1 Extension of Modalities to Contextual Reasoning;
 - Terms in $\Box_i\Gamma$ refer to commands that can be broadcasted from original address i to any other address in the Network for execution;
 - Terms in $\Diamond_i\Gamma$ refer to commands that need to be executed explicitly at the originating address i .

- 2 Distinct derivability relations:
 - $\Box_k(A \text{ true})$ iff for all $\Gamma_j \in \text{Context}$, $\emptyset \mid \Box_j\Gamma \vdash \Box_k(A \text{ true})$, where $j = \bigcup\{1, \dots, k-1\} \in \mathcal{G}$;
 - $\Diamond_k(A \text{ true})$ iff for some $\Gamma_j, \Delta_j \in \text{Context}$, $\Box_j\Gamma \mid \Diamond_j\Delta \vdash \Diamond_k(A \text{ true})$, where $j = \bigcup\{1, \dots, k-1\} \in \mathcal{G}$;

Modal Language

- 1 Extension of Modalities to Contextual Reasoning;
 - Terms in $\Box_i\Gamma$ refer to commands that can be broadcasted from original address i to any other address in the Network for execution;
 - Terms in $\Diamond_i\Gamma$ refer to commands that need to be executed explicitly at the originating address i .

- 2 Distinct derivability relations:
 - $\Box_k(A \text{ true})$ iff for all $\Gamma_j \in \text{Context}$, $\emptyset \mid \Box_j\Gamma \vdash \Box_k(A \text{ true})$, where $j = \bigcup\{1, \dots, k-1\} \in \mathcal{G}$;
 - $\Diamond_k(A \text{ true})$ iff for some $\Gamma_i, \Delta_j \in \text{Context}$, $\Box_i\Gamma \mid \Diamond_j\Delta \vdash \Diamond_k(A \text{ true})$, where $j = \bigcup\{1, \dots, k-1\} \in \mathcal{G}$;

- 1 Introduction
- 2 Polymorphism and Modal Types
- 3 Interaction in Contextual Computing**

Interaction: formal definition

Definition (Interacting processes)

We say that a process P interacts with a process P' iff at its execution, P is capable of controlling

- 1 access to location(s) of P' ;
- 2 commands (reading/writing/exec/broadcasting) of P' ;
- 3 validity of P' (global/local w.r.t. its locations).

Interaction: reduction to information control

Definition (Interaction and Control)

We say that a language L expresses process interaction iff

- 1 L allows to represent a function $Int(P, P')$ for interaction among processes P, P' ;
- 2 $Int(P, P')$ for L allows treatment of
 - program code accessibility;
 - information priority;
 - source security.

Definition (Interaction function in CMmTT)

In $CMmTT$ the dependency relation of multi-modal types expressed by contextual derivability formulates an interaction function.

Interaction: reduction to information control

Definition (Interaction and Control)

We say that a language L expresses process interaction iff

- ① L allows to represent a function $Int(P, P')$ for interaction among processes P, P' ;
- ② $Int(P, P')$ for L allows treatment of
 - program code accessibility;
 - information priority;
 - source security.

Definition (Interaction function in CMmTT)

In $CMmTT$ the dependency relation of multi-modal types expressed by contextual derivability formulates an interaction function.

Conditions for Interaction

- **program code accessibility:**
 - full expression of conditions for executing a program (up to identity);
 - conditions can include user- location- machine-based properties;
 - validity under changing conditions (contextual dynamics);
 - transmission of data to locations;
- **information priority:**
 - structured/ordered presentation of the located informations (simulates distribution/parallelism);
- **source security:**
 - expresses reliability referring to complete/incomplete data.

Conditions for Interaction

- **program code accessibility:**
 - full expression of conditions for executing a program (up to identity);
 - conditions can include user- location- machine-based properties;
 - validity under changing conditions (contextual dynamics);
 - transmission of data to locations;
- **information priority:**
 - structured/ordered presentation of the located informations (simulates distribution/parallelism);
- **source security:**
 - expresses reliability referring to complete/incomplete data.

Conditions for Interaction

- **program code accessibility:**
 - full expression of conditions for executing a program (up to identity);
 - conditions can include user- location- machine-based properties;
 - validity under changing conditions (contextual dynamics);
 - transmission of data to locations;
- **information priority:**
 - structured/ordered presentation of the located informations (simulates distribution/parallelism);
- **source security:**
 - expresses reliability referring to complete/incomplete data.

Operational Semantics

Definition (Operational Model)

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

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

Terms $\mathcal{T} := \mid \text{synchro}(.(.)) \mid \text{run}_i(\alpha) \mid \text{exec}(\alpha) \mid \text{GLOB}(\cdot) \mid \text{BROAD}(\cdot) \mid$

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

Operational Semantics (II)

	$(L) \mapsto (L')$
<i>run</i>	$[\Gamma]x_i \mapsto [\diamond\Gamma]run_i(\alpha)$
<i>exec</i>	$[\Gamma]a_i \mapsto [\square\Gamma]exec(\alpha)$
\rightarrow	$[\Gamma]exec(\alpha) \vdash b_j \mapsto [\square\Gamma]synchro(b_j(exec(\alpha)))$
\supset	$[\Gamma]run_i(\alpha) \vdash b_j \mapsto [\diamond\Gamma]synchro(b_j(run_i(\alpha)))$
\wedge	$[\Gamma]run_i(\alpha), run_j(\beta) \mapsto [\square\Gamma]exec(\alpha, \beta)$
$\square 1$	$[\Gamma] \vdash run_i(\alpha) \mapsto GLOB[\square\Gamma, \alpha]$
$\square 2$	$[\square_i\Gamma]exec(\alpha) \mapsto RET[\Gamma, \alpha_{i \cup j}]$
$\diamond 1$	$[\Gamma] \vdash run_i(\alpha) \mapsto BROAD[\diamond\Gamma, \alpha_{i \cap j}]$
$\diamond 2$	$[\diamond\Gamma] \vdash run_{i,j}(\alpha) \mapsto SEND[\Gamma, \alpha_{i \cap j}]$

Evaluation in the Semantics

- **Closure:** Evaluation of syntactic transformation requires all closed expressions ($exec(\alpha)$ and $[\Box\Gamma]\alpha$) – satisfies Safety, Preservation, Progress;
- **Openess:** Occurrences of run_i and \diamond_i require a side condition on preservation of indices – guarantees dynamics and data broadcasting;
- **Structure:** Evaluation on contexts proceeds on the ordering induced by $i < j$ – simulates temporal/logical priority;
- **Reduction:** 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 instruction execution in one of the rules to obtain \mathcal{L}' .

Evaluation in the Semantics

- **Closure:** Evaluation of syntactic transformation requires all closed expressions ($exec(\alpha)$ and $[\Box\Gamma]\alpha$) – satisfies Safety, Preservation, Progress;
- **Openess:** Occurrences of run_i and \diamond_i require a side condition on preservation of indices – guarantees dynamics and data broadcasting;
- **Structure:** Evaluation on contexts proceeds on the ordering induced by $i < j$ – simulates temporal/logical priority;
- **Reduction:** 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 instruction execution in one of the rules to obtain \mathcal{L}' .

Evaluation in the Semantics

- **Closure:** Evaluation of syntactic transformation requires all closed expressions ($exec(\alpha)$ and $[\Box\Gamma]\alpha$) – satisfies Safety, Preservation, Progress;
- **Openness:** Occurrences of run_i and \diamond_i require a side condition on preservation of indices – guarantees dynamics and data broadcasting;
- **Structure:** Evaluation on contexts proceeds on the ordering induced by $i < j$ – simulates temporal/logical priority;
- **Reduction:** 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 instruction execution in one of the rules to obtain \mathcal{L}' .

Evaluation in the Semantics





- **Closure:** Evaluation of syntactic transformation requires all closed expressions ($exec(\alpha)$ and $[\Box\Gamma]\alpha$) – satisfies Safety, Preservation, Progress;
- **Openness:** Occurrences of run_i and \diamond_i require a side condition on preservation of indices – guarantees dynamics and data broadcasting;
- **Structure:** Evaluation on contexts proceeds on the ordering induced by $i < j$ – simulates temporal/logical priority;
- **Reduction:** 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 instruction execution in one of the rules to obtain \mathcal{L}' .

Conclusion




Ideas for a reductionist model on algorithmic interaction:

- 1 Output correctness for programs requires data completeness *and* accessibility;
- 2 Interaction is defined among processes with open, modifiable and broadcastable data;
- 3 It admits a rich operational transition-state semantics;
- 4 Reduction requires a dynamic correctness check procedure on context extensions;
- 5 Users are not substituted, metadata fulfill processes.

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.
-  M. Burgin.
Super-recursive algorithms, Monographs in computer science,
Springer, 2005.

References

-  B.J. Copeland.
The broad conception of computation.
American Behavioral Scientist, pp.690–716, vol.40, n.6, 1997.
-  B.J. Copeland.
Hypercomputation
Minds & Machines, pp.461–502, vol.12, 2002.
-  R. Davies and F. Pfenning.
A modal analysis of staged computation.
Journal of the ACM, 48(3):555–604, 2001.

References

 R. De Millo and J.R. Lipton and A.J. Perlis

Social processes and Proofs of Theorems and Programs.
Communications of the ACM, pp. 271–280, vol.22(15), 1979.

 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.

 , R. Milner.

Communication and Concurrency, Prentice-Hall, 1989.

 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.

References



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.






A. Nanevski, F. Pfenning, and B. Pientka.

Contextual modal type theory.

ACM Transactions on Computational Logic, 9(3):1–48, 2008.

References

-  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.

References



G. Primiero..

A multi-modal dependent type theory for representing data accessibility in a network.

In A. Simpson (ed.), *Proceedings of the Proof Systems for Program Logics Workshop (LICS Affiliated at FLOC 2010, Edinburgh, UK)*, EasyChair Electronic Proceedings, 2010.



A.K. Simpson.

The Proof Theory and Semantics of Intuitionistic Modal Logic. PhD thesis, University of Edinburgh. College of Science and Engineering. School of Informatics, 1994.



P. Wegner.

Why Interaction is more powerful than algorithms.

Communications of the ACM, pp.81-91, vol.40(5), 1997.

References



P. Wegner.

Interactive foundation of computing.

Theoretical Computer Science, pp.315-351, vol.192, 1998.