A Multi-Modal Dependent Type Theory for Representing Data Accessibility in a Network

Giuseppe Primiero

FWO- Flemish Research Foundation
Centre for Logic and Philosophy of Science
Ghent University, Belgium
IEG, Oxford, UK
giuseppe.primiero@ugent.be

Abstract

In this paper we present a multi-modal polymorphic constructive type theory for a computational interpretation of programs equipped with locations for data accessibility in the context of distributed processing.

1 Introduction

Constructive modalities ([2, 1]) and modal type theories ([9, 8]) have recently been studied to give operational versions of modal logics. Their interpretation of programs via the Curry-Howard isomorphism allows to reason about distributed and staged computation ([4, 3, 5, 6]). Two major approaches to modal logics for distributed computing can be recognised: the syntactical treatment of a type theoretical language in [3] to establish global/local validity of modal propositions; and the semantic approach of an intuitionistic modal logic $ML5$ for Grid Computing in [5, 6], enriched with the concept of location derived from modalities. Varying on the former theme, a polymorphic constructive type theory with judgemental (rather than propositional) modalities is developed in [10] for studying derivability from open assumptions in a constructive setting. In this language, modalities express conditional validity of propositions in terms of contextual extensions for dependent types. In this paper we present the extension to multi-modalities in structured contexts to explore reasoning about ordered distributed computing.

2 System

Contextual dependency provides all that is needed to express syntactically the notion of truth relativized to states typical of modal logics. Dependent judgements of the form $a : A[\Gamma]$, with $\Gamma = [x_1 : A_1, \ldots, x_n : A_n]$ and a a proof of A, hold under appropriate substitutions $a : A[x_1/a_1 : A_1, \ldots, x_n/a_n : A_n]$, corresponding to β-reductions for proof terms and the explicit evaluation at run-time of the codes from which a program depends. To formulate partially evaluated specifications, i.e. codes missing their β-redex, requires the admission of separate constructors for the types expressed in context . We shall use a set $\mathcal{T}$ built from indexed term constructors $a_i, b_j, \ldots$ and variable constructors $x_i, y_j, \ldots$, such that each is a constructor for an appropriate type $A, B, \ldots$ and $i, j \in \mathcal{G}$ range over an enumerable set $\mathcal{G}$ of distinct locations. An indexed term constructor a_i for a type A is intended as the program for specification A with signature i; $a_i : A$ says that A is presented with an α-term a signed by its issuer i; to ensure that this is a canonical proof-term that can be used under any other signature, we enrich the language with an appropriate modality $\Box_{i \in \mathcal{G}}(A\ true)$. An indexed variable constructor x_i for a type A,

on the other hand, is intended as the admissible program for specification A with signature i: $x_i : A$ says that a type A is admissible from source i, but cannot be indexed at any other source. To express this we enrich the language with an appropriate modality $\Diamond_{i\in\mathcal{G}}(A\ true)$. This explanation makes the necessity judgement $\Box(A\ true)$ equivalent to $a : A[\emptyset]$, i.e. where all contextual expressions have been evaluated and the related code program is valid from *any* accessible location; the possibility judgement $\Diamond(A\ true)$ is equivalent to $a : A[\Gamma]$, where Γ stands for a set of programs using code admissible at *some* locations as specified by the relevant indices included in Γ. Our starting point of view is therefore that assertion conditions in their modal translation correspond to information on the internal structure of a running program.

2.1 Language

Our set $\mathcal{K}$ of kinds includes two elements: $\mathcal{K} := \{(A, B, \ldots type); (A, B, \ldots type_{inf})\}$; *type* is the kind of all valid specifications defined by term constructors (executed program), and $type_{inf}$ is the kind of all information chuncks defined by variable constructors (admissible commands) used to define further elements of the kind *type*. An expression $x_i : A$ is an assumption with x_i in the set of terms and $A\ type_{inf}$, declaring that instructions to execute a program for A are available at source i. A context Γ is a finite sequence of assumptions $[x_i : A, \ldots, x_n : N]$ all with distinct subjects, expressing a network to execute those routines. A judgement $J[x_i : A, \ldots, x_n : N]$ says that a program – as specified in J – is executed provided each of the routines $A, \ldots, N$ is called at the appropriate location, each call depending on the forecoming ones in the same Γ. Each $x_i : \alpha$ depends on the assumptions $x_1 : \alpha$ up to $x_{i-1} : \alpha$ (with α a metavariable for possibly distinct types). If $\Gamma = \{x_i : A, \ldots, x_n : N\}$, an extended context $\Delta = \{\Gamma, x_{n+1} : N + 1\}$ is equivalent to $\Delta = \{x_i : A, \ldots, x_{n+1} : N + 1\}$. When the formulation of a fresh declaration $x_{n+1} : N + 1$ is meant to be independent of the order in Γ, we use a separator $\Gamma \mid x_{n+1} : N + 1$. The rules for signed expressions in the kind *type* are (omitting elimination rules for brevity):

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

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

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

$$\frac{}{\Gamma, a_i : A, \Delta \vdash A\ type.}\ \text{Global Validity Rule} \qquad \frac{\Gamma \vdash B\ type \quad \Gamma \vdash A\ type}{\Gamma \mid a_i : A \vdash B\ type.}\ \text{Weakening}$$

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

The Global Validity Rule uses premise generation to say that if a program for A is generated at source i, its validity is global to the relevant $\mathcal{G}$ accessible from i. This property makes it possible to validate the other structural rules for expressing modularity. The rules for signed expressions in the kind $type_{inf}$ are:

$$\frac{\neg(A \to \bot)\ type}{A\ type_{inf}}\ Type_{inf}\ \text{Formation} \qquad \frac{A\ type_{inf} \quad b_j : B[x_i : A]}{((x_i)b_j) : A \supset B\ type_{inf}}\ \text{Functional abstraction}$$

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

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

$$\frac{}{\Gamma, x_i : A, \Delta \vdash A\ type_{inf}}\ \text{Local Validity Rule} \qquad \frac{\Gamma \vdash B\ type_{inf} \quad x_i : A \vdash A\ type_{inf}}{\Gamma \mid x_i : A \vdash B\ type_{inf}.}\ \text{Weakening}$$

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

The Local Validity Rule says that the execution of a program for A depending on source i makes its validity bounded (starred) to that point in $\mathcal{G}$, until discharged (by β-conversion). Notice that in this case, validity of the structural rules is restricted to assumptions that are not in a relation order within a context.

2.2 Multi-Modalities

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

$$\frac{a_i : A}{\Box_i(A\ true)}\ \Box - \text{Formation} \qquad \frac{x_i : A}{\Diamond_i(A\ true)}\ \Diamond - \text{Formation}$$

Context extension mimicks accessibility on worlds, so that judgemental modal operators express terms executing somewhere or everywhere, with respect to locations. Contexts, as formal counterparts of networks, are disegned according to the relevant signatures. A context Γ_i is a context signed for i iff any declaration in Γ has signature i and all have distinct subjects $\{A, \ldots, N\} \in type_{inf}$. $\Box_i(A\ true)$ is a modal premise generated by $\Box$-Formation from $x_i[/a_i] : A$ with $x_i, a_i \in \mathcal{T}$, and $A\ type$; a modal premise is the declaration that A is valid for any extension of context Γ_i. $\Diamond_i(A\ true)$ is a modal assumption generated by $\Diamond$-Formation from $x_i : A$ with $x_i \in \mathcal{T}$ and $A\ type_{inf}$, declaring that A is a locally valid routine in some extension of context Γ_i. For any context Γ_i, $\Box_i\Gamma$ is given by $\bigcup\{\Box_i(A\ true) \mid \text{ for all } A \in \Gamma\}$; $\Diamond_i\Gamma$ is given by $\bigcup\{\circ_i(A\ true) \mid \circ = \{\Box, \Diamond\}$ and $\Diamond_i(A\ true)$ for at least one $A \in \Gamma\}$. A signed (modal) context $\circ\Gamma_i$ can be extended by a differently signed (modal) context $\circ\Delta_j$. Modal judgements derivable from multi-signed contexts are defined as follows:

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

A multi-modal context $\Sigma_{i,j}$ is a context extension

$$\circ_i\Gamma \mid \circ_j\Delta = \{\circ_i(A\ true) \ldots, \circ_i(N\ true), \circ_j(O\ true)\};$$

($\circ = \{\Box; \Diamond\}$ and $\Sigma_{i,j}$ is abbreviated as $\Sigma_{\mathcal{G}}$). A context extension $\circ_i\Gamma \mid \circ_j\Delta$ is admissible if, for any judgement $J \in \Delta$ such that $J = A\ type_{inf}$, $\Gamma \nvdash (A \to \bot)$.

The type-theoretical expression $\Diamond_{\mathcal{G}}\Sigma \vdash J$ is obtained by $\Box_i\Gamma \mid \Diamond_j\Delta \vdash J$ and expresses the local validity of J from source i and j in view of the information that source j makes available when accessed from source i and which can be lost when accessing source k:

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

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

Extending Γ_i with information accessible locally at j, validity holds by calling upon relevant sources (i.e. at their intersection). Elimination starts from a similarly derivable judgement $\Diamond_{i,j}(A\ true)$ to infer its variable constructor, then deriving local validity of B without the additional location of A.

The type-theoretical expression $\Box_{\mathcal{G}}\Sigma \vdash J$ is obtained by $\Box_i\Gamma \mid \Box_j\Delta \vdash J$ and expresses the global validity of J from source i and j, in view of the information that source j makes available when accessed from source i and which persists when accessing any other source k:

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

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

I$\Box$ turns local validity into global validity by instantiation of all premises (execution of codes). Elimination starts from a similarly derived $\Box_{\mathcal{G}}(B\ true)$ to decompose its locations.

2.3 Properties

Inference from $x_i : A$ to $\Diamond_i(A\ true)$ says that if A is admissible in the system, then a program can be executed using a subroutine for A at a given location. Considering a single node in a network, Reflexivity holds; admitting $\mathcal{G} = \{1, \ldots, n\}, n > 1$, a basic requirement is that processing happens in a strictly ordered way, then (ordered) Transitivity is enforced: if a process at k takes compuational information J at j, and J uses processual information J' at i, then the process at k also uses J' at i (for $(i < j < k \in \mathcal{G})$). Symmetry for such relation is not admitted.

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

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

β-conversion is reformulated as $\Box$-Import to enforce execution of partially evaluated processes, obtained by an instance of the Premise Rule and $\Box$-Formation; $\Diamond$-Import is a modal version of abstraction on terms expressing the formulation of processes that can be used at their sources at runtime:

$$\frac{\Gamma_i, x_j : A \vdash B\ true^* \quad a_j : A \vdash A\ true}{\Box_i\Gamma, a_j : A \vdash \Box_{i,j}(B\ true)}\ \Box\ \text{Import}$$

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

If we force the order relation and use accessibility to a new location in the first premise of $\Diamond$-Import, Common Seriality is satisfied, which says that each staged program can be traced back to some processual information:

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

Transmission of processual information is enforced by convergence (from which symmetry can be obtained as a special case): if there is a program executed at i and j uses such process (for the usual $i < j$ and using Common Seriality), then the information used at j is executable at i (Convergence satisfies Semi-Adjunction or Seriality for the monomodal B):

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

Derivability under $\Box_n\Sigma$ allows admissibility of $\Box_k(A\ true)$ by any $\Gamma_i, \Delta_j \in \Box_n\Sigma$ and $i < j < k \in \mathcal{G}$, from which follows *Upper Inclusion*: if a program is actually executed in a network, then it can be accessed from any location within that network; *Lower Inclusion* expresses accessibility of executed programs at any lower admissible location in $\mathcal{G}$ (accessibility of the lower point considered is satisfied by Convergence):

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

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

By Inclusion, validity of a program implies its admissibility at each location (which makes easy to validate forms of Equivalence and Union). By the multimodal version of $\Box_{1,2}$-Formation, we can instantiate ordered iteration, which gives us an equivalent of $S4$: the ascending part says that if a program for A is executed at k using processes at i, j, then the execution of program for A at i, j is accessible at k; the descending part says that if a program for A is executed at k using processes at i, j, then the execution of a program for A at k is accessible at i, j (a sort of code mobility without seriality, see [7]). This is easily derivable from Convergence and β-reduction.

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

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

3 Conclusion

We have presented a polymorphic modal type theory for reasoning about distributed computing. Computationally, values of $\Box_{\mathcal{G}}(A\ true)$ is everywhere accessible code and values of $\Diamond_{\mathcal{G}}(A\ type)$ is locally accessible code. Local validity and completeness are provable generalizing from the monomodal case presented in [10]. Code sources are ordered as to mimick their functional aspect and the modal interation simulates the validity of code at distinct locations.

References

[1] 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.

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

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

[4] 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.

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

[6] 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.

[7] T. Murphy, K. Crary, R. Harper, and F. Pfenning. A symmetric modal lambda calculus for distributed computing. In H. Ganzinger, editor, *Proceedings of the 19th Annual Symposium on Logic in Computer Science (LICS'04)*, pages 286–295. IEEE Computer Society Press, 2004.

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

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

[10] G. Primiero. A contextual type theory with judgemental modalities for reasoning from open assumptions. In *Logique & Analyse*. Forthcoming (2012).