Programming language semantics in modal type theories

Thesis Defence

Philipp Stassen

August 9th 2024

Aarhus University

Introduction

The topics of my Thesis

- Modelling FPC in guarded type theory
- Modelling Probabilistic FPC in guarded type theory
  Philipp Stassen, Rasmus Møgelberg, Maaike Zwart, Alejandro Aguirre, Lars Birkedal
- `mitten` : a flexible multimodal proof assistant
  Philipp Stassen, Daniel Gratzer, Lars Birkedal

Proving equivalence of programs

There are three approaches to reason about program equivalence [Mog91]

1. Operational approach
2. Denotational approach
3. Logical approach

Operational Approach

- Operational semantics relates programs with values (partial function eval)
- Programs $M, N : \mathbb{N}$ are equivalent, if $\text{eval}(M) = \text{eval}(N)$, e.g. $\text{eval}(2 + 4) = \text{eval}(2 \cdot 3)$
- How about functions and terms with free variables?
  $\rightarrow$ Contextual Equivalence captures this (Definition later)

Denotational Approach

- A programming language is interpreted as a certain mathematical domain.

Theorem (Soundness)
If $[\![M]\!]_\rho = [\![N]\!]_\rho$, then M and N are contextually equivalent.

Theorem (Adequacy)
If M and N are contextually equivalent, then $[\![M]\!]_\rho = [\![N]\!]_\rho$

Side Effects

- **Nontermination** (Partiality)
- Concurrency
- **Probabilistic Choice**
- Error handling
- State

Example
Prob(coin $= 0) = \frac{1}{2}$ and Prob(coin $= 1) = \frac{1}{2}$

Modelling Effects with Monads

- Monads endow a set of values A with some extra structure (write $\mathcal{M}(A)$)
- Probabilistic choice by a distribution monad
- Nontermination by a partiality monad.

Programming language vs Type Theory

- Certain programming languages double as mathematical logic (Curry-Howard isomorphism)
- Often referred to as *type theories*.
- Nontermination breaks the Curry-Howard isomorphism

Modelling Probabilistic FPC in Guarded Type Theory

Contributions

- Modelling Probabilistic FPC in guarded type theory
  1. Syntax, operational and denotational semantics
  2. Finite distribution monad
  3. Guarded convex delay monad
  4. Convex delay monad
  5. Theory of lifting relations to the convex delay monad

Meta Theory - Guarded Type Theory

Requirements

- We want to use *guarded type theory*
- Type theory does not support non-termination natively
- Programs are probabilistic, and thus terminate to distributions over values

Clocked Cubical Type Theory ticks all the boxes.

Finite Distribution Monad

Definition

For any set A, let $\mathcal{D}(A)$ is generated by

$$\delta : A \to \mathcal{D}(A) \qquad \oplus : (0,1) \to \mathcal{D}(A) \to \mathcal{D}(A) \to \mathcal{D}(A)$$

validating, *idempotency*, *commutativity* and *associativity*.

$$\underbrace{\mu}_{p} \oplus_p \underbrace{\nu}_{1-p}$$

How to model partiality constructively?

Idea: Define monad L such that $\mathsf{L}(A) \simeq A + \mathsf{L}(A)$

$$\eta : A \to \mathsf{L}(A) \qquad\qquad \mathsf{step} : \mathsf{L}(A) \to \mathsf{L}(A)$$

- step is an explicit computation step.
- Supposed to be a coinductive type
- We can define the infinite (unproductive) loop $\bot : \mathsf{L}(A)$.
  $\bot = \mathsf{step}(\bot)$

Clocked Cubical Type Theory (CCTT)

CCTT has the "later-modality" $\triangleright^\kappa$ (indexed by formal clock κ).

- Elements of $\triangleright^\kappa A$ are available after one step of computation
- We can delay an element $a : A$ to $\text{next}^\kappa(a) : \triangleright^\kappa A$
- Guarded fixpoint combinator $\text{fix}^\kappa : (\triangleright^\kappa A \to A) \to A$.
- *clock quantification* $\forall \kappa . A$ allows us to realize coinductive types.

Guarded Convex Delay Monad

Definition (Guarded Convex Delay Monad)
Let D^κ be such that $\mathsf{D}^\kappa A \simeq \mathcal{D}(A + \triangleright^\kappa(\mathsf{D}^\kappa A))$. We define

$$\delta^\kappa : A \to \mathsf{D}^\kappa A$$

$$\mathsf{step}^\kappa : \triangleright^\kappa(\mathsf{D}^\kappa A) \to \mathsf{D}^\kappa A$$

$$- \oplus^\kappa_- - : (0,1) \times \mathsf{D}^\kappa A \times \mathsf{D}^\kappa A \to \mathsf{D}^\kappa A$$

Convex Delay Monad

Definition (Convex Delay Monad)

$$\mathsf{D}^{\forall} A \triangleq \forall \kappa. \mathsf{D}^{\kappa} A \qquad\qquad \mathsf{D}^{\forall} A \simeq \mathcal{D}(A + \mathsf{D}^{\forall} A)$$

$$\delta^{\forall} : A \to \mathsf{D}^{\forall} A$$

$$\mathsf{step}^{\forall} : \mathsf{D}^{\forall} A \to \mathsf{D}^{\forall} A$$

$$- \oplus^{\forall}_{-} - : (0,1) \to \mathsf{D}^{\forall} A \to \mathsf{D}^{\forall} A \to \mathsf{D}^{\forall} A$$

Coinduction - Convex Delay Monad

Example (Geometric Process)

We can define the semantic geometric process $\mathsf{geo}_p : \mathbb{N} \to \mathsf{D}^{\forall}(\mathbb{N})$ satisfying

$$\begin{aligned}\mathsf{geo}_p(0) &= (\delta^{\forall} 0) \oplus_p^{\forall} \mathsf{step}^{\forall}(\mathsf{geo}_p(1)) \\ &= (\delta^{\forall} 0) \oplus_p \mathsf{step}^{\forall}((\delta^{\forall} 1) \oplus_p \mathsf{step}^{\forall}(\mathsf{geo}_p(2))) \\ &= \ldots\end{aligned}$$

Probability of Termination

Definition
We can define a function

$$\mathsf{PT}_n : \mathsf{D}^{\forall} A \to [0, 1]$$

measuring the probability of termination after n computation steps.

The language $\mathsf{FPC}_{\oplus}$

$$
\begin{array}{llll}
\textit{(types)} & \sigma, \tau & ::= & 1 \mid \mathsf{Nat} \mid \mu X.\tau \mid \ldots \\
\textit{(values)} & V, W & ::= & \langle\rangle \mid \underline{n}\ (n : \mathbb{N}) \mid \mathsf{lam}\ x.M \mid \mathsf{fold}\ V \mid \langle V, W \rangle \\
 & & & \mid \mathsf{inl}\ V \mid \mathsf{inr}\ V \\
\textit{(terms)} & L, M, N & ::= & x \mid \langle\rangle \mid \underline{n}\ (n : \mathbb{N}) \mid \mathsf{suc}\ M \mid \mathsf{lam}\ x.M \\
 & & & \mid MN \mid \langle M, N \rangle \mid \mathsf{inl}\ M \mid \mathsf{inr}\ M \\
 & & & \mid \mathsf{fold}\ M \mid \mathsf{unfold}\ M \mid \mathsf{choice}^p(M, N) \mid \ldots
\end{array}
$$

The language $\mathsf{FPC}_{\oplus}$

$\Gamma \vdash M : \sigma$ denotes well-formed terms with variables bound in Γ.

Definition ($\vdash$ relation)

$$\frac{\Gamma \vdash M : \tau[\mu X.\tau/X]}{\Gamma \vdash \mathsf{fold}\, M : \mu X.\tau} \qquad \frac{\Gamma \vdash M : \mu X.\tau}{\Gamma \vdash \mathsf{unfold}\, M : \tau[\mu X.\tau/X]}$$

$$\frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \sigma \qquad p : (0,1)}{\Gamma \vdash \mathsf{choice}^{p}(M, N) : \sigma} \qquad \ldots$$

The language $\mathrm{FPC}_{\oplus}$

Example (Y-combinator)
For any types σ and τ, we may define the Y-combinator

$$\cdot \vdash \mathsf{Y} : ((\sigma \to \tau) \to (\sigma \to \tau)) \to \sigma \to \tau.$$

Example (Geometric process)
For any $p : (0, 1)$ we define the function

$$\mathsf{geo}_p : \mathsf{Nat} \to \mathsf{Nat}$$
$$\mathsf{geo}_p = \mathsf{lam}\ x.\mathsf{choice}^p(x, \mathsf{geo}_p(x + 1))$$

Collection of Terms and Values

- $\mathsf{Tm}_\sigma^\Gamma$ denotes the collection of well-typed terms of type σ in context Γ
- Val_σ denotes the collection of well-typed values of type σ in the empty context.
- We write Tm_σ if Γ is empty

Call-by-value Operational Semantics

We may define operational semantics of type:

$$\mathsf{eval} : \{\sigma : \mathsf{Ty}\} \to \mathsf{Tm}_\sigma \to \mathsf{D}^\forall(\mathsf{Val}_\sigma)$$

Example

$$\mathsf{eval}(\mathsf{choice}^{\frac{1}{2}}(\underline{0}, 2)) = \delta(0) \oplus^\forall_{\frac{1}{2}} \delta(2)$$

$$\mathsf{eval}(\mathsf{id}(0)) = \mathsf{step}^\forall(\delta(0))$$

Contextual Equivalence

A closing context is a function $C : \mathsf{Tm}_\sigma^\Gamma \to \mathsf{Tm}_1$.

Definition (Contextual Refinement)

Let $\Gamma \vdash M, N : \tau$ be terms. We say that N *contextually refines* M if for any closing context, and for any m there exists an n such that

$$\mathsf{PT}_m(\mathsf{eval}(C[M])) \leq \mathsf{PT}_n(\mathsf{eval}(C[N]))$$

. In this case, write $M \preceq_{\mathsf{Ctx}} N$. We say that M and N are contextually equivalent ($M \equiv_{\mathsf{Ctx}} N$) if $M \preceq_{\mathsf{Ctx}} N$ and $N \preceq_{\mathsf{Ctx}} M$.

Guarded Denotational Semantics

$$[\![-]\!]^\kappa : \mathsf{Ty} \to \mathsf{U}$$

$$\vdots$$

$$[\![\mu X.\tau]\!]^\kappa \triangleq \triangleright^\kappa [\![\tau[\mu X.\tau/X]]\!]^\kappa$$

- Environments $\rho : [\![\Gamma]\!]^\kappa$ list semantic values for types in Γ.

$$[\![-]\!]^\kappa_- : \{\Gamma : \mathsf{Ctx}\} \to \{\sigma : \mathsf{Ty}\} \to \mathsf{Tm}^\Gamma_\sigma \to [\![\Gamma]\!]^\kappa \to \mathsf{D}^\kappa([\![\sigma]\!]^\kappa)$$

$$[\![\mathsf{unfold}\ M]\!]^\kappa_\rho \triangleq [\![M]\!]^\kappa_\rho >>=^\kappa \lambda v.\mathsf{step}^\kappa(\lambda\alpha.\delta^\kappa(v[\alpha]))$$

$$[\![\mathsf{choice}^p(M,N)]\!]^\kappa_\rho \triangleq [\![M]\!]^\kappa_\rho \oplus^\kappa_p [\![N]\!]^\kappa_\rho$$

Denotational Semantics

Definition (Semantics)

$$[\![-]\!] : \mathsf{Ty} \to \mathsf{U}$$
$$[\![-]\!]_- : \{\Gamma : \mathsf{Ctx}\} \to \{\sigma : \mathsf{Ty}\} \to \mathsf{Tm}^{\Gamma}_{\sigma} \to [\![\Gamma]\!] \to \forall \kappa. \mathsf{D}^{\kappa}([\![\sigma]\!]^{\kappa})$$

- Environments $\rho : [\![\Gamma]\!]$ are lists of semantic values for types in Γ

Relating Syntax and Semantics

- Soundness of the semantics $[\![M]\!]_\rho = [\![N]\!]_\rho \to M \equiv_{\mathsf{Ctx}} N$
- Adequacy of the semantics: $M \equiv_{\mathsf{Ctx}} N \to [\![M]\!]_\rho = [\![N]\!]_\rho$
- Defining logical relation s.t. : $M \preceq N \to M \preceq_{\mathsf{Ctx}} N$
- Necessitates lifting a relation $\mathcal{R} : A \to B \to \mathsf{Prop}$ to $\overline{\mathcal{R}} : \mathsf{D}^{\forall} A \to \mathsf{D}^{\forall} B \to \mathsf{Prop}$.

Properties of relational lifting

$$\frac{\nu \rightsquigarrow \nu' \qquad \mu \; \overline{\mathcal{R}}^{\kappa} \; \nu}{\mu \; \overline{\mathcal{R}}^{\kappa} \; \nu'} \qquad \frac{\nu \rightsquigarrow \nu' \qquad \mu \; \overline{\mathcal{R}}^{\kappa} \; \nu'}{\mu \; \overline{\mathcal{R}}^{\kappa} \; \nu}$$

$$\frac{(\mathsf{step}^{\kappa} \mu_1) \oplus_p^{\kappa} (\mathsf{step}^{\kappa} \mu_2) \; \overline{\mathcal{R}}^{\kappa} \; \nu}{\mathsf{step}^{\kappa}(\lambda(\alpha : \kappa).(\mu_1 \, [\alpha]) \oplus_p^{\kappa} (\mu_2 \, [\alpha])) \; \overline{\mathcal{R}}^{\kappa} \; \nu}$$

$$\frac{\mu_1 \; \overline{\mathcal{R}}^{\kappa} \; \nu_1 \qquad \mu_2 \; \overline{\mathcal{R}}^{\kappa} \; \nu_2}{(\mu_1 \oplus_p^{\kappa} \mu_2) \; \overline{\mathcal{R}}^{\kappa} \; (\nu_1 \oplus_p^{\forall} \nu_2)} \qquad \frac{a \, \mathcal{R} \, b}{(\delta^{\kappa} a) \; \overline{\mathcal{R}}^{\kappa} \; (\delta^{\forall} b)}$$

$$\frac{\mu \; \overline{\mathcal{R}}^{\kappa} \; \overline{\nu} \qquad \forall a, b. a \, \mathcal{R} \, b \to f(a) \; \overline{\mathcal{S}}^{\kappa} \; g(b)}{\overline{f}(\mu) \; \overline{\mathcal{S}}^{\kappa} \; \overline{g}(\nu)}$$

Logical Relation

Definition

For $M, N : \mathsf{Tm}^{\Gamma}_{\sigma}$ we define a relation $M \preceq^{\kappa,\Gamma}_{\sigma} N$ using guarded recursion, the relational lifting and induction on types.

Soundness of logical relation

Theorem

For any terms $\Gamma \vdash M : \sigma$ *and* $\Gamma \vdash N : \sigma$ *we have*

$$(\forall \kappa. M \preceq_{\sigma}^{\kappa,\Gamma} N) \to M \preceq_{\mathsf{Ctx}} N$$

Further Directions

1. Combining and extending the present work with the account of nondeterminism.
2. Extend the logical relation to account for approximate relational reasoning (up to a small ϵ), which would allow us, *e.g.*, to show that constant functions are refinements of their approximations. ✓
3. Remove the unnecessary computation steps in the denotational semantics — simplifying many calculations. ✓

Logical relation

Value relation

$$\frac{}{n \preceq^{\kappa,\mathsf{Val}}_{\mathsf{Nat}} \underline{n}} \qquad \frac{}{\star \preceq^{\kappa,\mathsf{Val}}_{1} \langle\rangle} \qquad \frac{\triangleright(\alpha : \kappa).(v[\alpha] \preceq^{\kappa,\mathsf{Val}}_{\tau[\mu X.\tau/X]} V)}{v \preceq^{\kappa,\mathsf{Val}}_{\mu X.\tau} \mathsf{fold}\ V}$$

$$\frac{\forall w, V.w \preceq^{\kappa,\mathsf{Val}}_{\sigma} V \to v(w) \preceq^{\kappa,\mathsf{Tm}}_{\tau} \mathsf{eval}(M[V/x])}{v \preceq^{\kappa,\mathsf{Val}}_{\sigma\to\tau} \mathsf{lam}\ x.M}$$

Expression relation

$$\mu \preceq^{\kappa,\mathsf{Tm}}_{\sigma} d \triangleq \mu \overline{\preceq^{\kappa,\mathsf{Val}}_{\sigma}}^{\kappa} d$$

Final logical relation For $M, N : \mathsf{Tm}^{\Gamma}_{\sigma}$ we define

$$M \preceq^{\kappa,\Gamma}_{\sigma} N \triangleq \left(\forall \rho, \delta.(\rho \preceq^{\kappa,\mathsf{Val}}_{\Gamma} \delta) \to [\![M]\!]^{\kappa}_{\rho} \preceq^{\kappa,\mathsf{Tm}}_{\sigma} \mathsf{eval}(N[\delta])\right)$$

Proof (Sketch)

1. **Fundamental lemma**: $\forall(M : \mathsf{Tm}^{\Gamma}_{\sigma}) \to M \preceq^{\kappa,\Gamma}_{\sigma} M$.
2. **Congruence theorem** : For any well-formed context C and terms M, N, if $\forall\kappa.M \preceq^{\kappa,\Gamma}_{\sigma} N$, then also $\forall\kappa.C[M] \preceq^{\kappa,\Delta}_{\tau} C[N]$.
3. **Soundness theorem** of the denotational semantics: For any well typed closed expression $\cdot \vdash M : \sigma$ we have that

$$\mathsf{D}^{\kappa}\Big(\llbracket - \rrbracket^{\mathsf{Val},\kappa}\Big)\,(\mathsf{eval}^{\kappa} M) \equiv \llbracket M \rrbracket^{\kappa}$$

4. $(\forall\kappa.M \preceq^{\kappa,\cdot}_{1} N) \to (\mathsf{eval}(M)\,\overline{\mathsf{eq}_1}\,\mathsf{eval}(N))$
5. $\forall M, N : \mathsf{Tm}_1$ with $\mathsf{eval}(M)\,\overline{\mathsf{eq}_1}\,\mathsf{eval}(N)$ it follows that

$$\forall n : \mathcal{N}\,\exists m : \mathcal{N}\,.\mathsf{PT}_n(\mathsf{eval}(M)) \leq \mathsf{PT}_m(\mathsf{eval}(N)).$$

Proof (Sketch)

Definition (Contextual refinement)
We write that $M \preceq_{\mathsf{Ctx}} N$, if for any closing context C it follows

$$\forall n : \mathcal{N}\, \exists m : \mathcal{N}.\mathsf{PT}_n(\mathsf{eval}(C[M])) \leq \mathsf{PT}_m(\mathsf{eval}(C[N]))$$

.

Proof.
Assume $\forall \kappa. M \preceq_{\sigma}^{\kappa,\Gamma} N$, and let C be a closing context.

- It follows that $\forall \kappa. C[M] \preceq_{1}^{\kappa,\Gamma} C[N]$.
- This implies that $(\mathsf{eval}(C[M])\, \overline{\mathsf{eq}_1}\, \mathsf{eval}(C[N]))$
- and finally it follows
  $\forall n : \mathcal{N}\, \exists m : \mathcal{N}.\mathsf{PT}_n(\mathsf{eval}(C[M])) \leq \mathsf{PT}_m(\mathsf{eval}(C[N]))$

□

A Flexible Type Checker For Modal Type Theories

Before we start: A word on proof assistants

- Mathematical proofs are difficult and error-prone
- Idea: Write proof in semi-decidable formal language
  $\rightarrow$ Computer can validate correctness
- Many problems benefit from specialized languages

Modalities in Computer Science

Modalities provide abstraction for programming languages

- Information Flow [Kav19]
- Distributed Systems
- Synchronous Programming [Gua18]
- Coinductive Data Types [Clo+15]

as well as reasoning principles in mathematics

- Axiomatic Cohesion [Shu18]
- Guarded Recursion [Nak00]
- Monads/Comonads/Adjunctions

MTT — a machine that produces modal type theories

MTT takes as input a description of the modal situation – a mode theory – and produces a modal type theory

Importantly, MTT has a well developed meta-theory. In particular:

- MTT is sound [Gra+20]
- there is a normalization algorithm for MTT [Gra21]
- MTT enjoys canonicity. [Gra21]

Carefully chosen mode theories recover some of the prior examples.

Contribution

`mitten` is a prototype implementation of MTT.

Like MTT, `mitten` easily adapts to different modal situations.

Contributions:

1. A normalization algorithm for MTT
2. A bidirectional type checking algorithm for MTT

`mitten` — a type checker with a hole

Without a concrete mode theory, MTT is not a type theory and `mitten` not a type checker.

An implementation of a mode theory completes `mitten`

One has to implement a structure to describe the mode theory:

1. Abstract type of modalities
2. Preorder and equality relation defined on modalities

Example: Guarded Recursion – implementation

Instantiating MTT with the modalities

$$\begin{aligned}&\mathsf{type}\ \mathtt{modality} = \\ &\quad \mid \triangleright \mid \Box \mid \mathsf{id} \mid (\circ)\ \mathsf{of}\ \mathtt{modality} * \mathtt{modality}\end{aligned}$$

and predicates

$$(=) : \mathtt{modality} \times \mathtt{modality} \to \mathtt{bool}$$

$$(\leq) : \mathtt{modality} \times \mathtt{modality} \to \mathtt{bool}$$

allows us to formalize guarded recursion.

Mode Theory Implementations

- This code is both necessary and sufficient: the general word problem for mode theories is undecidable!
- Equality for MTT is decidable iff the mode theory is
- **In practice:** Implementing a mode theory requires relatively few lines of code.

Normalization

- Normalization for MTT has been proven by Gratzer [Gra21].
- Although the proof is constructive, it is not clear how to extract an algorithm.
- Restricting the mode theories allowed us to implement an algorithm based on normalization-by-evaluation [Abe13].
- A *weak-head normal form* algorithm seems more promising.

Bidirectional Type Checking Algorithm

At this point, we utilize the entire ML-signature of the mode theory:

1. Equality of terms uses normalization and equality of modalities
2. At every stage, we carefully check modes and modalities
3. Use $\leq$ to validate the correct usage of variables.

Summary — What is `mitten`?

`mitten` is an adjustable type checker and:

Flexible The underlying normalization algorithm and type checker do not depend on specifics of the modalities

Expressive MTT extends MLTT (cubical variants already exist).

Simple Implementing a type checker is reduced to a simpler problem.

Going forward

- Generalize `mitten`
- Make it usable by integrating this technique into a main stream proof assistant.
- Good class of decidable mode theories?

Thanks

Github

`https://github.com/logsem/mitten_preorder`

Usage

Input

```
let next : (A : U<0>) -> A -> << l | A >> @ T =
    fun A -> fun x -> mod l x

normalize next Nat 2 at << l | Nat >> @ T
```

Output

```
Computed normal form of
    (ap () (ap () next Nat) 2)
as
    (mod (l) 2)
```