A Flexible Type Checker For Modal Type Theories

Philipp Stassen Daniel Gratzer Lars Birkedal

January 22nd 2022

Aarhus University

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]

Judiciously chosen mode theories recover some of the prior examples.

Contribution

We contribute `mitten`, a prototype implementation of MTT.

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

Concretely, we developed and implemented

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

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

$$\begin{array}{l}\textsf{type}\ \texttt{modality} =\\ \quad \mid \triangleright \mid \Box \mid \mathsf{id} \mid (\circ)\ \textsf{of}\ \texttt{modality} * \texttt{modality}\end{array}$$

and predicates

$$\begin{aligned}(=) &: \texttt{modality} \times \texttt{modality} \to \texttt{bool}\\ (\leq) &: \texttt{modality} \times \texttt{modality} \to \texttt{bool}\end{aligned}$$

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. We use $\leq$ to validate the correct usage of variables.

Summary

We implemented an adjustable type checker

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.
- Can we identify good class of decidable mode theories?

Thanks

Thanks to the organizers and all participants! ☺

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)
```