Defining and relating theories

Håkon Gylterud
Plan

- Look at some example theories.
- Discuss how to represent these.
- Look at Myott.
- Discuss future directions.
What is Myott (going to be)?

- Stand-alone specification tool:
  - for theories.
  - for translations between theories.
  - Code generation from theories.
- Haskell API for working with theories.

https://git.app.uib.no/Hakon.Gylterud/myott
Motivation

Håkon Gylterud
Defining and relating theories
A multitude of theories

Type theories:

- Different versions of MLTT
- Extensions and variations
  - Inductive families, induction-recursion, ···
  - HITs
  - Cubical
  - Modalities
  - ···
A multitude of theories

Other kinds of theories:

- Set theory
- First-order logic
- Higher-order logic
- Category theory
- Linear logic
- Software specification
Problems when relating different kinds of theories

In what sense are set theory and type theory both theories?
Problems when relating different kinds of theories

In what sense are set theory and type theory both theories?

Classical answer:

- A theory is a set of theorems.
Problems when relating different kinds of theories

In what sense are set theory and type theory both theories?

Classical answer:

- A theory is a set of theorems.

This does not...

- ...explain how to define a theory.
- ...explain what kind of objects a theory is about.
- ...explain how to relate different theories.
Problems when relating different kinds of theories

In what sense are set theory and type theory both theories?

Classical answer:

- A theory is a set of theorems.

This does not...

- ...explain how to define a theory.
- ...explain what kind of objects a theory is about.
- ...explain how to relate different theories.

For the sake of implementing Myott, we need a opinionated notion of theory, which settle these questions.
There are several ways to translate set theory into type theory (Aczel’s model, HoTT-book model, ···).

\[ \sigma(\exists x \phi) = \sum_{x : M} \sigma(\phi) \]

\[ \tau(\exists x \phi) = \| \sum_{x : M} \tau(\phi) \|_{-1} \]
There are several ways to translate set theory into type theory (Aczel’s model, HoTT-book model, ···).

\[ \sigma(\exists x \; \phi) = \sum_{x:M} \sigma(\phi) \]
\[ \tau(\exists x \; \phi) = \| \sum_{x:M} \tau(\phi) \|_{-1} \]

In each case it is obvious how to define the translation, but when formalising this has to be done:

- either by hand
- or internally to type theory.
Related work

- Per Martin-Löf’s: About Models for Intuitionistic Type Theories and the notion of Definitional Equality
Related work

- Per Martin-Löf’s: About Models for Intuitionistic Type Theories and the notion of Definitional Equality
- Cartmell’s generalised algebraic theories.
Related work

- Per Martin-Löf’s: About Models for Intuitionistic Type Theories and the notion of Definitional Equality
- Cartmell’s generalised algebraic theories.
- Makkai’s FOLDS
Related work

- Per Martin-Löf’s: About Models for Intuitionistic Type Theories and the notion of Definitional Equality
- Cartmell’s generalised algebraic theories.
- Makkai’s FOLDS
- Bauer, Haselwarter & Lumsdaine: What are we thinking when we present a type theory?
Related work

- Per Martin-Löf’s: About Models for Intuitionistic Type Theories and the notion of Definitional Equality
- Cartmell’s generalised algebraic theories.
- Makkai’s FOLDS
- Bauer, Haselwarter & Lumsdaine: What are we thinking when we present a type theory?
- Uemura’s abstract type theories.
Related work

- Per Martin-Löf’s: About Models for Intuitionistic Type Theories and the notion of Definitional Equality
- Cartmell’s generalised algebraic theories.
- Makkai’s FOLDS
- Bauer, Haselwarter & Lumsdaine: What are we thinking when we present a type theory?
- Uemura’s abstract type theories.
- Lumsdaine & Subramaniam’s dependent operads.
Overview

Translations

Operations

Rules

Judgement forms

Defining and relating theories
Figure 2: Both assumptions and constructions can be viewed as pushouts.
Extensions

Figure 3: Both assumptions and constructions can be viewed as pushouts.
Examples
Martin-Löf type theory has four kinds of judgement:

- \( A : \text{type} \)
Type theory: Judgement forms

Martin-Löf type theory has four kinds of judgement:

- A : type
- a : element A
Martin-Löf type theory has four kinds of judgement:

- \( A : \text{type} \)
- \( a : \text{element } A \)
- \( A \equiv B \)
Type theory: Judgement forms

Martin-Löf type theory has four kinds of judgement:

- $A : \text{type}$
- $a : \text{element } A$
- $A \equiv B$
- $a \equiv a' : A$
Type theory: Judgement forms

Martin-Löf type theory has four kinds of judgement:

- A : type
- a : element A
- A \equiv B
- a \equiv a' : A
Type theory: Judgement forms

Martin-Löf type theory has four kinds of judgement:

- A : type
- a : element A
- A ≡ B
- a ≡ a' : A

Notice: The equality judgement are *propositional* – no variables introduced.
Each kind of judgement come with presuppositions:

- A type
Type theory: Presuppositions

Each kind of judgement come with presuppositions:

- A type
- a : A, presupposing A type.
Type theory: Presuppositions

Each kind of judgement come with **presuppositions**:

- A type
- \( a : A \), presupposing A type.
- \( A \equiv B \), presupposing A type and B type.
Type theory: Presuppositions

Each kind of judgement come with presuppositions:

- A type
- a : A, presupposing A type.
- A ≡ B, presupposing A type and B type.
- a ≡ a' : A, presupposing A type, a : A and a' : A.
Categories: Judgement forms

When working inside a particular category, we would have the following judgement forms:

- A : object
Categories: Judgement forms

When working inside a particular category, we would have the following judgement forms:

- \( A : \text{object} \)
- \( f : \text{hom} \ A \ B \)
Categories: Judgement forms

When working inside a particular category, we would have the following judgement forms:

- $A : \text{object}$
- $f : \text{hom } A B$
- $f \equiv g : \text{hom } A B$
Categories: Presuppositions

Again, we have presuppositions:

- $A : \text{object}$
Categories: Presuppositions

Again, we have presuppositions:

- A : object
- f : hom A B, presupposing A,B object.
Categories: Presuppositions

Again, we have presuppositions:

- \( A : \text{object} \)
- \( f : \text{hom} \ A \ B, \text{presupposing} \ A, B \ \text{object} \).
- \( f \equiv g : \text{hom} \ A \ B, \text{presupposing} \ A, B : \text{object} \ \text{and} \ f, g : \text{hom} \ A \ B \)
Set theory

One might expect set theory to have judgement forms:

- A set
- $A \in B$
- $A = B$
Set theory

One might expect set theory to have judgement forms:

- A set
- $A \in B$
- $A = B$

But, actually formulas are an integral part of set theory
Set theory: Judgement forms

This means we get the following:

- $A : \text{set}$
- $\phi : \text{formula}$
- $\phi$ true (proposition)
Set theory: Judgement forms

This means we get the following:

- $A : \text{set}$
- $\phi : \text{formula}$
- $\phi$ true (proposition)

Elementhood and equality would then be a term-forming operation for formula:

$A \ B \ \text{set} \vdash A \in B : \text{formula}$
In each example theory, we have:

- a set of judgement forms.
- some judgements are propositional.
- judgements have presuppositions.
<table>
<thead>
<tr>
<th>Myott</th>
<th>Motivation</th>
<th>Examples</th>
<th>Makkai's dependent sort vocabularies</th>
<th>Judgements in Myott</th>
<th>Operations</th>
<th>Operations in Myott</th>
</tr>
</thead>
</table>

**Makkai’s dependent sort vocabularies**
Definition

A dependent sort vocabulary is a pair $\langle C, P \rangle$ where

- $C$ is a (finite/with finite out-degree) category.
Definition

A dependent sort vocabulary is a pair $\langle C, P \rangle$ where

- $C$ is a (finite/with finite out-degree) category.
- The relation $a \prec b$ on objects of $C$, defined by $a \prec b \iff \exists f : b \to a. f \neq id_a$, is wellfounded.
Definition

A dependent sort vocabulary is a pair $\langle C, P \rangle$ where

- $C$ is a (finite/with finite out-degree) category.
- The relation $a \prec b$ on objects of $C$, defined by $a \prec b \iff \exists f : b \to a. f \neq id_a$, is wellfounded.
- $P$ is a set of ($\prec$-maximal) elements of $\text{Ob}_C$. 
Example: Category judgement form signature

\[ \vdash \text{object sort} \]
\[ \vdash x,y : \text{object} \vdash \text{hom}(x,y) \text{ sort} \]
\[ \vdash x,y : \text{object}, f,g : \text{hom}(x,y) \vdash f \equiv g \text{ proposition} \]
Example: Category judgement form signature

- $\vdash \text{object sort}$
- $x,y : \text{object} \vdash \text{hom}(x,y) \text{ sort}$
- $x,y : \text{object}, f,g : \text{hom}(x,y) \vdash f \equiv g \text{ proposition}$

Figure 4: The above signature as a DSV.
Example: Category judgement form signature (alt)

- \( \vdash \) object sort
- \( \text{dom, codom : object} \vdash \text{hom(dom,codom)} \) sort
- \( \text{dom, codom : object, lhs,rhs : hom(dom,codom)} \vdash \)
- \( \text{lhs \equiv rhs} \) proposition

Figure 5: The above signature as a DSV.
Example: The judgements of type theory

- $\vdash$ type sort
- $A : type \vdash$ element $A$ sort
- $A, B : type \vdash A \equiv B$ sort
- $A : type, a, a' : element A \vdash a \equiv a'$ proposition
Example: The judgements of type theory

- $\vdash \text{type sort}$
- $A : \text{type} \vdash \text{element } A \text{ sort}$
- $A,B : \text{type} \vdash A \equiv B \text{ sort}$
- $A : \text{type}, a,a' : \text{element } A \vdash a \equiv a' \text{ proposition}$

Figure 6: The above signature as a DSV.
Example: The judgements of type theory (alt)

- ▫ ⊢ type sort
- ▫ type-of : type ⊢ element type-of sort
- ▫ lhs,rhs : type ⊢ lhs ≡ rhs sort
- ▫ A : type, lhs,rhs : element A ⊢ lhs ≡ rhs

proposition

Figure 7: The above signature as a DSV
Judgements in Myott
Operations
Categories: Operations

Categories can be formulated as a generalised algebraic theory, where the operations are:

\[ x : \text{object} \vdash \text{id} x : \text{hom} x x \]

\[ x, y, z : \text{object}, \quad f : \text{hom} x y, g : \text{hom} y z \]

\[ \vdash g \circ f : \text{hom} x z \]
Categories: Equations

Equations can be seen as operations as well:

\[
\begin{align*}
x &: \text{ob}, \ y &: \text{ob}, \ z &: \text{ob}, \ w &: \text{ob}, \\
f &: \text{hom} \ x \ y, \ g &: \text{hom} \ y \ z, \ h &: \text{hom} \ z \ w \\
\vdash \text{assoc} \ x \ y \ z \ w \ f \ g \ h : \\
& \quad h \circ (g \circ f) \\
& \equiv (h \circ g) \circ f
\end{align*}
\]
Judgements in context as pure operations

Remember, compositions in categories:

\[ x, y, z : \text{object}, \]
\[ f : \text{hom } x \to y, \quad g : \text{hom } y \to z \]
\[ \vdash g \circ f : \text{hom } x \to z \]

The signature of this operation is a \text{purely judgemental} context.
Operations depending on operations

Consider the two operations:

1. \( \vdash 1 : \text{object} \)
2. \( x : \text{object} \vdash ! : \text{hom}(x,1) \)

Notice:

- The second rule depends on the first.
- The first rule must be used in a well-formed way.
The operations of a theory are organised in a well-founded category.

- Arrows in the operation category represent applications of the rule in the signature of another rule.
The operations of a theory are organised in a well-founded category.

- Arrows in the operation category represent applications of the rule in the signature of another rule.

For categories it looks like:

![Diagram of operations](image)

Figure 8: The category of operations for categories
Term expansion

A rule application is a map between finite structures, hence all subterms must be present in context:

- ⊢ 1 : object
- x : object ⊢ ! : hom(x,1)
Term expansion

A rule application is a map between finite structures, hence all subterms must be present in context:

- ⊢ \(1 : \text{object}\)
- \(x : \text{object}, y := 1() : \text{object} \vdash ! : \text{hom}(x,y)\)
Operations in Myott
Rules
Type theory: Rules

Rules can express variable binding:

\[ \vdash A : \text{type} \]
\[ x : \text{element } A \vdash B x : \text{type} \]

\[ \Pi \text{-formation} \]

\[ \vdash \prod (x:A) (B x) : \text{type} \]
Rules can express variable binding:

\[ \vdash A : \text{type} \]
\[ \vdash x : \text{element } A \vdash B x : \text{type} \]

\[ \prod \text{-formation} \]
\[ \vdash \prod (x:A) (B x) : \text{type} \]

Notice that the assumptions on the rules form a signature of operations.
First-order logic

Similar rules handles quantifiers in FOL:

\[ x : \text{set} \vdash \phi(x) : \text{formula} \]

\[ \frac{}{\vdash \forall x. \phi(x) : \text{formula}} \]
Rule elaboration

\[ \vdash A : \text{Type} \]
\[ x : \text{Element } A \vdash B x : \text{Type} \]
\[ x : \text{Element } A \vdash b x : \text{Element } B x \]

\[ \vdash \lambda(x : A) (b x) : \text{Element } (\Pi (x : A) (B x)) \]
Rule elaboration

\[
\begin{align*}
A_0 := & A(), \ x : \text{Element } A_0 \\
B_0 := & B(x) \\
C_0 := & C() : \text{Type}
\end{align*}
\]

\[
\begin{align*}
\vdash & A : \text{Type} \\
\vdash & B x : \text{Type} \\
\vdash & C := \prod (x : A) \ (B x) : \text{Type} \\
\vdash & b x : \text{Element } B_0 \\
\end{align*}
\]

\[
\begin{align*}
\vdash & \lambda (x : A) \ (b x) : \text{Element } C
\end{align*}
\]
Rules in Myott
Translators

A translation converts:

- judgement forms to derived judgement forms
- operations to derived operations
- rule to derived rules (derivations)

Example: Setoid model

\[ \tau(\text{type}) := \{ \]
\[ \quad \vdash E : \text{type}; \]
\[ \quad x,y : \text{element } E \vdash R x y : \text{type}; \]
\[ \quad x : \text{element } E \quad \vdash \text{refl } x : \text{element } (R x x); \]
\[ \quad \ldots \} \]
\[ \tau(\text{element}) (E, R, \text{refl}, \ldots) \mapsto \{ \]
\[ \quad \vdash \text{el} : \text{element } E ; \}
\[ \quad \ldots \]
Future directions

- All the theoretical parts need to be properly written down.
Future directions

- All the theoretical parts need to be properly written down.
- Parsing / checking for rules.
Future directions

- All the theoretical parts need to be properly written down.
- Parsing / checking for rules.
- Built-in notion of equation/reductions.
Future directions

- All the theoretical parts need to be properly written down.
- Parsing / checking for rules.
- Built-in notion of equation/reductions.
- Translations between theories.
Future directions

- All the theoretical parts need to be properly written down.
- Parsing / checking for rules.
- Built-in notion of equation/reductions.
- Translations between theories.
- Usability:
Future directions

- All the theoretical parts need to be properly written down.
- Parsing / checking for rules.
- Built-in notion of equation/reductions.
- Translations between theories.
- Usability:
  - Module system
Future directions

- All the theoretical parts need to be properly written down.
- Parsing / checking for rules.
- Built-in notion of equation/reductions.
- Translations between theories.
- Usability:
  - Module system
  - Inferrable arguments and premisses
Future directions

- All the theoretical parts need to be properly written down.
- Parsing / checking for rules.
- Built-in notion of equation/reductions.
- Translations between theories.
- Usability:
  - Module system
  - Inferrable arguments and premisses
  - Custom grammars
Future directions

- All the theoretical parts need to be properly written down.
- Parsing / checking for rules.
- Built-in notion of equation/reductions.
- Translations between theories.
- Usability:
  - Module system
  - Inferrable arguments and premisses
  - Custom grammars
- More liberal notions of rules (example: binding many variables)