Defining and relating theories

Håkon Gylterud

Plan

- Look at some example theories.
- Discuss how to represent these.
- Look at Myott.
- Discuss future directions.

Myott

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

A multitude of theories

Type theories:

- Different versions of MLTT
- Extensions and variations
  - Inductive families, induction-recursion, $\cdots$
  - HITs
  - Cubical
  - Modalities
  - $\cdots$

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.

Problems when relating different kinds of theories

There are several ways to translate set theory into type theory (Aczel's model, HoTT-book model, $\cdots$).

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

Problems when relating different kinds of theories

There are several ways to translate set theory into type theory (Aczel's model, HoTT-book model, $\cdots$).

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

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?

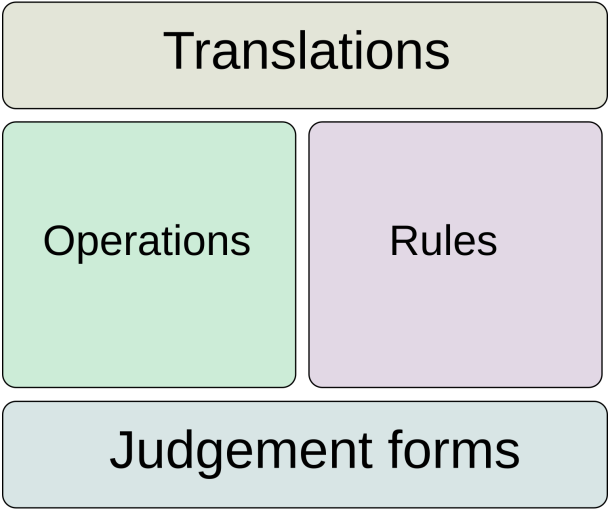
Related work

- 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



Extensions

$$\begin{tikzcd} \partial R \arrow[r, "d_R"] \arrow[d, "\alpha"'] & yR \\ \Gamma & \end{tikzcd}$$

Figure 2: Both assumptions and constructions can be viewed as pushouts.

Extensions

$$
\begin{tikzcd}
\partial R \arrow[r, "d_R"] \arrow[d, "\alpha"'] & yR \arrow[d] \\
\Gamma \arrow[r] & \Gamma.(R\alpha) \arrow[ul, phantom, "\lrcorner", very near start]
\end{tikzcd}
$$

Figure 3: Both assumptions and constructions can be viewed as pushouts.

Examples

Type theory: Judgement forms

Martin-Löf type theory has four kinds of judgement:

- `A : type`

Type theory: Judgement forms

Martin-Löf type theory has four kinds of judgement:

- `A : type`
- `a : element A`

Type theory: Judgement forms

Martin-Löf type theory has four kinds of judgement:

- `A : type`
- `a : element A`
- `A` $\equiv$ `B`

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` $\equiv$ `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.

Type theory: Presuppositions

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` $\equiv$ `B`, **presupposing `A type` and `B type`.**
- `a` $\equiv$ `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 : object`
- `f : hom A B`

Categories: Judgement forms

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

- `A : object`
- `f : hom A B`
- `f ≡ g : hom A B`

Categories: Presuppositions

Again, we have presuppositions:

- `A : object`

Categories: Presuppositions

Again, we have presuppositions:

- `A : object`
- `f : hom A B`, **presupposing** `A,B object`.

Categories: Presuppositions

Again, we have presuppositions:

- `A : object`
- `f : hom A B`, **presupposing** **`A,B object.`**
- `f ≡ g : hom A B`, **presupposing** **`A,B : object`** **and** **`f,g : hom A B`**

Set theory

One might expect set theory to have judgement forms:

- `A set`
- $\mathtt{A} \in \mathtt{B}$
- `A = B`

Set theory

One might expect set theory to have judgement forms:

- `A set`
- $\texttt{A} \in \texttt{B}$
- `A = B`

But, actually formulas are an integral part of set theory

Set theory: Judgement forms

This means we get the following:

- `A : set`
- ϕ `: formula`
- ϕ `true` (propsition)

Set theory: Judgement forms

This means we get the following:

- $\mathtt{A : set}$
- ϕ $\mathtt{: formula}$
- ϕ $\mathtt{true}$ (propsition)

Elementhood and equality would then be a term-forming operation for $\mathtt{formula}$:

$\mathtt{A\ B\ set} \vdash \mathtt{A} \in \mathtt{B : formula}$

Judgement forms

In each example theory, we have:

- a set of judgement forms.
- some judgements are propositional.
- judgements have presuppositions.

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 $a \prec b \Leftrightarrow \exists f : b \to a.\ f \neq \mathrm{id}_a$, 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 $a \prec b \Leftrightarrow \exists f : b \to a.\ f \neq \mathrm{id}_a$, is wellfounded.
- P is a set of ($\prec$-maximal) elements of Ob_C.

Example: Category judgement form signature

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

Example: Category judgement form signature

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

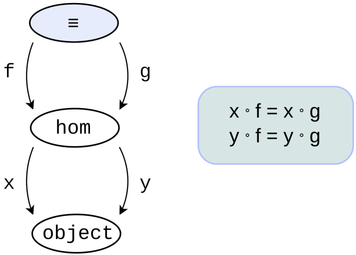
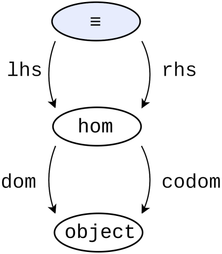


Figure 4: The above signature as a DSV.

Example: Category judgement form signature (alt)

- `⊢ object sort`
- `dom, codom : object ⊢ hom(dom,codom) sort`
- `dom, codom : object, lhs,rhs : hom(dom,codom) ⊢ lhs ≡ rhs proposition`




```
  dom ∘ lhs =   dom ∘ rhs
codom ∘ lhs = codom ∘ rhs
```

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

- $\vdash$ `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`

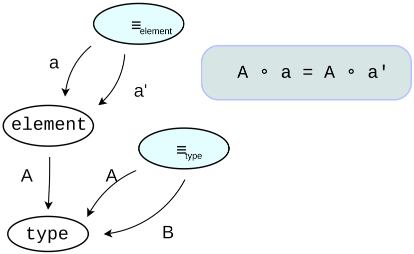


Figure 6: The above signature as a DSV.

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`

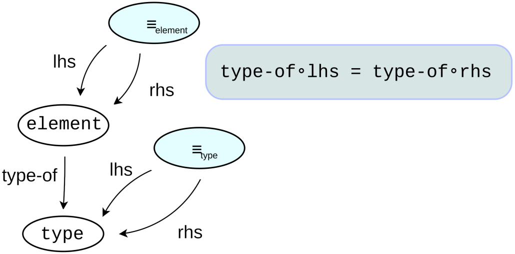


Figure 7: The above signature as a DSV

Judgements in Myott

Operations

Categories: Operations

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

```
x : object ⊢ id x : hom x x

  x, y, z : object,
  f : hom x y, g : hom y z
⊢ g ∘ f : hom x z
```

Categories: Equations

Equations can be seen as operations as well:

```
   x:ob, y:ob, z:ob, w:ob,
   f:hom x y, g:hom y z, h:hom z w
⊢ assoc x y z w f g h :
             h ∘ (g ∘ f)
           ≡ (h ∘ g) ∘ f
```

Judgements in context as pure operations

Remember, compositions in categories:

```
  x, y, z : object,
  f : hom x y, g : hom y z
⊢ g ∘ f : hom x z
```

The signature of this operation is a **purely judgemental** context.

Operations depending on operations

Consider the two operations:

- $\vdash$ `1 : object`
- `x : object` $\vdash$ `! : hom(x,1)`

Notice:

- The second rule depends on the first.
- The first rule must be used in a well-formed way.

Operations

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.

Operations

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:

$$\begin{tikzcd}
\mathtt{assoc} \arrow[d, bend right=40] \arrow[d, bend right=15] \arrow[d, bend left=15] \arrow[d, bend left=40] & \mathtt{lid} \arrow[dl] \arrow[d] & \mathtt{rid} \arrow[dll] \arrow[dl] \\
\mathtt{comp} & \mathtt{id} &
\end{tikzcd}$$

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:

- $\vdash$ `1 : object`
- `x : object` $\vdash$ `! : hom(x,1)`

Term expansion

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

- $\vdash$ `1 : object`
- `x : object, y` $:=$ `1() : object` $\vdash$ `! : hom(x,y)`

Operations in Myott

Rules

Type theory: Rules

Rules can express variable binding:

$$\dfrac{\begin{array}{r}\vdash \texttt{A} \quad : \texttt{type} \\ \texttt{x : element A} \vdash \texttt{B x} : \texttt{type}\end{array}}{\vdash \prod \texttt{(x:A) (B x) : type}} \; \prod\texttt{-formation}$$

Type theory: Rules

Rules can express variable binding:

$$\dfrac{\begin{array}{rl} & \vdash \mathtt{A} \quad\; : \mathtt{type} \\ \mathtt{x : element\ A} & \vdash \mathtt{B\ x} : \mathtt{type} \end{array}}{\vdash \prod\ \mathtt{(x{:}A)\ (B\ x)} : \mathtt{type}}\ \prod\text{-}\mathtt{formation}$$

Notice that the assumptions on the rules form a signature of operations.

First-order logic

Similar rules handles quantifiers in FOL:

$$\frac{\texttt{x : set} \vdash \phi(\texttt{x}) \texttt{ : formula}}{\vdash \forall \texttt{ x. } \phi(\texttt{x}) \texttt{ : formula}} \ \forall\texttt{-form}$$

Rule elaboration

```
                ⊢ A   : Type
x : Element A ⊢ B x : Type
x : Element A ⊢ b x : Element B x
———————————————————————————————————————— λ-abstract
⊢ λ(x : A) (b x) : Element (Π (x : A) (B x))
```

Rule elaboration

```
                                         ⊢ A     : Type
A₀:=A(), x : Element A₀                  ⊢ B x : Type
                                         ⊢ C := (Π (x : A) (B x)) : Type
A₀:=A(), x : Element A₀, B₀=B(x) ⊢ b x : Element B₀
————————————————————————————————————————— λ-abstract
C₀ := C() : Type ⊢ λ(x : A) (b x) : Element C
```

Rules in Myott

Translations

A translation converts:

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

Example: Setoid model

```
τ(type) := {
                    ⊢ E : type;
   x,y : element E ⊢ R x y : type;
   x : element E   ⊢ refl x : element (R x x);
   ···}
τ(element) (E, R , refl,···) ↦ {
           ⊢ el : element E ; }
...
```

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)