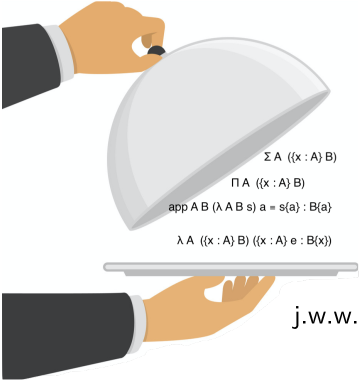
Andromeda 2 - your type theory à la carte



Anja Petković[1]

[1]University of Ljubljana, Slovenia

ICMS 2020,
July 15, 2020

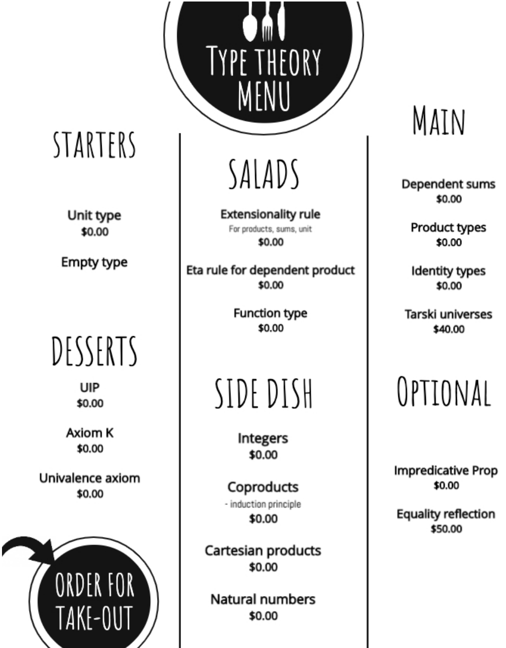
j.w.w. Andrej Bauer and Philipp G. Haselwarter

[1]This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0326.

Motivation

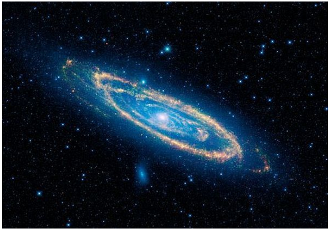
- Popular proof assistants have a fixed underlying type theory.
- What if we want to tailor it to our needs?
  - flags in Agda: `--with-K`, `---without-K`, `--no-eta-equality`, `--rewrite`, `--cubical`
  - flags in Coq: `--impredicative-set`, `--type-in-type`
  - Dedukti supports user-defined rewrite rules.

Make your own rules



Andromeda 2

- In Andromeda 2 type theory is user-definable.
- It supports finitary type theories.
- LCF-style proof assistant with Andromeda Meta-Language (AML).
- Trusted Nucleus, that governs constructors for judgements.
- Recent development: Equality checking algorithm.



Finitary Type Theories

General type theories with finitary rules and finitely many of them.

- 4 hypothetical judgement forms

$$\Gamma \vdash A \text{ type} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash A \equiv B \qquad \Gamma \vdash a \equiv b : A$$

- boundaries

$$\Gamma \vdash \Box \text{ type} \qquad \Gamma \vdash \Box : A \qquad \Gamma \vdash A \stackrel{?}{\equiv} B \qquad \Gamma \vdash a \stackrel{?}{\equiv} b : A$$

- well-presented inference rules

In AML there is an abstract datatype of judgements and of boundaries, whose constructors are controlled by the Nucleus.

Well-presented rules

Informally, well-presented rules are what we usually write in fraction-form:

Example

$$\frac{\Gamma \vdash A \ \textsf{type} \quad \Gamma, x{:}A \vdash B \ \textsf{type}}{\Gamma \vdash \Pi(x{:}A)\,.\,B \ \textsf{type}}$$

Well-presented rules

Informally, well-presented rules are what we usually write in fraction-form:

Example

$$\frac{\Gamma \vdash A \ \mathsf{type} \quad \Gamma, x{:}A \vdash B \ \mathsf{type}}{\Gamma \vdash \Pi(x{:}A) \,.\, B \ \mathsf{type}}$$

Non-example:

$$\frac{\Gamma, x{:}\mathbb{N} \vdash P \ \mathsf{type}}{\Gamma \vdash t : P[\mathsf{refl}(0)/x]}$$

Rules have to be given in a well-ordered form, so everything makes sense according to the previous set of rules.
Nucleus supports structural rules, substitution and inversion principles.

Equality rules

The user may specify rules for judgemental equality:

Example (Dependent functions)

$$\frac{\vdash A \text{ type} \quad \vdash \{x{:}A\}B \text{ type} \quad \vdash \{x{:}A\}s : B(x) \quad \vdash a : A}{\vdash \mathsf{app}(A, B, \lambda(A, B, s), a) \equiv s[a/x] : B(a)}$$

In Andromeda 2:

```
rule Π_beta (A type) ({x : A} B type)
({x : A} s : B{x}) (a : A)
: app A B (lambda A B s) a == s{a} : B{a} ;;
```

`B{a}` instantiates the bound variable `x` with `a`.
Terms are fully annotated with their types.

Definitions

Equational rules serve as a definition in object type theory.

Example

```
rule three : N ;;
rule three_def : three == s(s(s(z))) : N ;;
```

`let`-bindings are definitions in AML.

Example

```
let funs = derive (X type) ->  Π X ({x} X) ;;
```

`derive` makes a derived rule.

Congruence rules

Nucleus automatically generates congruence rules for term and type formers.

Example (Congruence rule for Π)

$$\frac{\Gamma \vdash A \equiv A' \quad \Gamma, x{:}A \vdash B(x) \equiv B'(x)}{\Gamma \vdash \Pi(A, \{x\}B(x)) \equiv \Pi(A', \{x\}B'(x))}$$

In Andromeda 2:

```
rule A_eq_A' : A == A' ;;
rule B_eq_B' (x : A) : B x == B' (convert x A_eq_A');;
congruence (Π A ({x} B x)) (Π A' ({x} B' x))
             A_eq_A' ({y : A} B_eq_B' y) ;;
```

Congreunce rules are left-leaning. Terms are explicitly converted.

Other features of AML

- Inversion principles: pattern-matching.

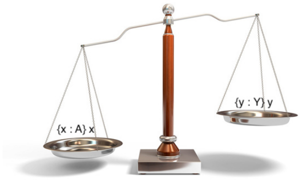
Example

```
let two = s(s(z));;

match two with
| s (?one) -> one
end;;
```

- Inductive types.
- Operations, exceptions, handlers.

Equality checking algorithm



We have designed and implemented a user-extensible equality checking algorithm, based on type-directed equality checking, e.g., Harper & Stone (2006).

Check equality of terms s and t of type A:

1. **type-directed phase:** normalize the type A and apply extensionality rules, if any.
2. **normalization phase:** if no extensionality rules apply, normalize s and t and structurally compare their normal forms using congruence rules.

Normalization

- Use computation rules as long as any apply.
- Normalize the *principal arguments*.

Normalization outputs a certified equation between the original and normalized expression.

Extensionality rules

$$\frac{P_1 \quad \cdots \quad P_n \qquad \vdash x : A \qquad \vdash y : A \qquad Q_1 \quad \cdots \quad Q_m}{\vdash x \equiv y : A},$$

where

- $P_1, \ldots, P_n$ are object premises,
- $Q_1, \ldots, Q_m$ are equality premises,

Example (Extensionality rule for dependent functions[1])

$$\frac{\begin{array}{c}\vdash A \textit{ type} \qquad \vdash \{x{:}A\}B \textit{ type} \\ \vdash f : \Pi(A, \{x\}B(x)) \qquad \vdash g : \Pi(A, \{x\}B(x)) \\ \vdash \{x{:}A\}\ \textit{app}(A, B, f, x) \equiv \textit{app}(A, B, g, x) : B(x)\end{array}}{\vdash f \equiv g : \Pi(A, \{x\}B(x))}$$

[1] not to be confused with function extensionality

Extensionality rules

$$\frac{P_1 \quad \cdots \quad P_n \qquad \vdash x : A \qquad \vdash y : A \qquad Q_1 \quad \cdots \quad Q_m}{\vdash x \equiv y : A},$$

where

- $P_1, \ldots, P_n$ are object premises,
- $Q_1, \ldots, Q_m$ are equality premises,

Example (Extensionality rule for dependent functions[1])

$$\frac{\begin{array}{c}\vdash A \text{ type} \qquad \vdash \{x{:}A\}B \text{ type} \\ \vdash f : \Pi(A, \{x\}B(x)) \qquad \vdash g : \Pi(A, \{x\}B(x)) \\ \vdash \{x{:}A\}\ \mathsf{app}(A, B, f, x) \equiv \mathsf{app}(A, B, g, x) : B(x)\end{array}}{\vdash f \equiv g : \Pi(A, \{x\}B(x))}$$

Note: Inter-derivable with η-rules.

[1] not to be confused with function extensionality

Computation rules

Computation rules take the forms

$$\frac{P_1 \quad \cdots \quad P_n}{\vdash u \equiv v : T} \qquad\qquad \frac{P_1 \quad \cdots \quad P_n}{\vdash A \equiv B}$$

where the P_i's are object premises.

- u has the form $\textsf{s}(e_1, \ldots, e_m)$
- A has the form $\textsf{S}(e_1, \ldots, e_m)$

Example (Dependent functions)

$$\frac{\vdash A \textit{ type} \quad \vdash \{x{:}A\}B \textit{ type} \quad \vdash \{x{:}A\}s : B(x) \quad \vdash a : A}{\vdash \textit{app}(A, B, \lambda(A, B, s), a) \equiv s[a/x] : B(a)}$$

Future work

- Add support for confluence and termination of normalization.
- Appraise efficiency and find opportunities for optimization.
- Extend the algorithm to cover more complex patterns.