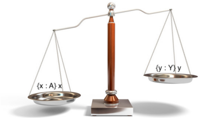


Equality Checking for Finitary Type Theories

Anja Petković[1]

[1] University of Ljubljana, Slovenia

HoTTEST Conference 2020,
June 18, 2020

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

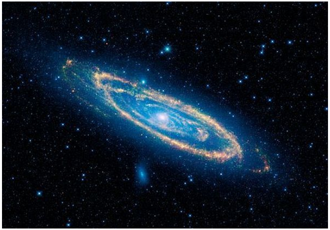
[1] This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0326.

Motivation

- Equality checking algorithms are essential parts of proof assistants.
- Most popular proof assistants provide them for their underlying type theory.
- Extensions to the equality checking.

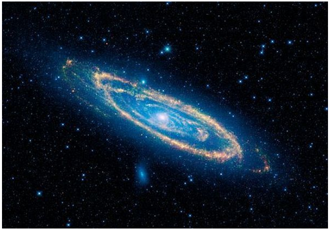
Motivation

What happens with user-definable type theory like in Andromeda 2?



Motivation

What happens with user-definable type theory like in Andromeda 2?



What we did:

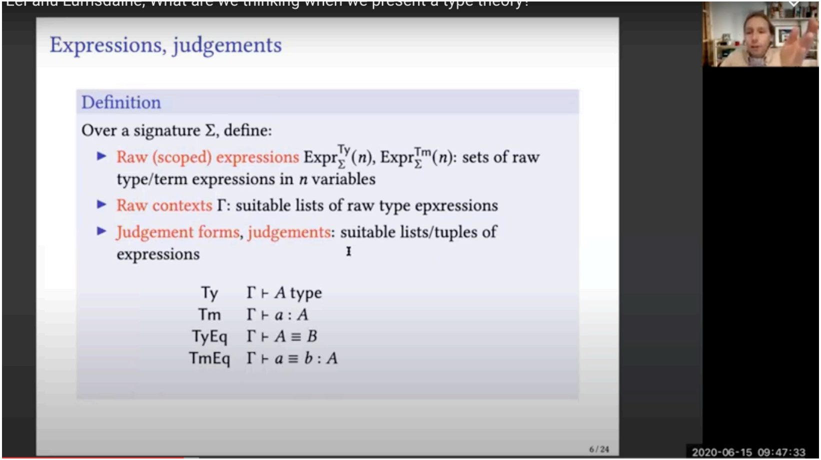
- Designed a user-extensible equality checking algorithm, based on type-directed equality checking, e.g., Harper & Stone (2006).
- Implementation in Andromeda 2.

Talk overview

- Finitary Type Theories (as implemented in Andromeda 2).
- Overview of the algorithm:
  - type-directed phase,
  - normalization phase,
  - normal forms.
- Live demo: using the implementation of the equality checker in Andromeda 2.

Finitary Type Theories

An adaptation of *general type theories* that Peter Lumsdaine talked about,



but finitary rules and finitely many of them.

Finitary Type Theories

- 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

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

- well-presented rules (finitary and finitely many)

Context-free presentation of finitary type theories

Andromeda 2 is an LCF-style proof assistant:

Context-free presentation of finitary type theories

Andromeda 2 is an LCF-style proof assistant:

no proof state

Context-free presentation of finitary type theories

Andromeda 2 is an LCF-style proof assistant:

$$\text{no proof state} \implies \text{no global contexts.}$$

Context-free presentation of finitary type theories

Andromeda 2 is an LCF-style proof assistant:

$$\text{no proof state} \implies \text{no global contexts.}$$

Context-free presentation:

- Previous work: Γ_∞ by Geuvers et al. for Calculus of Constructions.
- No explicit contexts.
- Free variables are tagged with their types: a^A.

Context-free presentation of finitary type theories

Andromeda 2 is an LCF-style proof assistant:

$$\text{no proof state} \implies \text{no global contexts.}$$

Context-free presentation:

- Previous work: Γ_∞ by Geuvers et al. for Calculus of Constructions.
- No explicit contexts.
- Free variables are tagged with their types: a^A.

Details: Philipp Haselwarter's dissertation.

Context-free presentation of finitary type theories

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

Context-free presentation of finitary type theories

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

$$\downarrow$$

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

Abstraction is a primitive notion.

Context-free presentation of type theories

4 judgement forms:

$$j := \qquad A\ \mathsf{type} \qquad a{:}A \qquad A \equiv B\ \mathsf{by}\ \alpha \qquad a \equiv b : A\ \mathsf{by}\ \alpha$$

boundaries:

$$\mathsf{b} := \qquad \Box\ \mathsf{type} \qquad \Box : A \qquad A \equiv B\ \mathsf{by}\ \Box \qquad a \equiv b : A\ \mathsf{by}\ \Box$$

Abstracted judgements and boundaries:

$$\{x_1{:}A_1\} \dots \{x_n{:}A_n\} j \quad \{x_1{:}A_1\} \dots \{x_n{:}A_n\} \mathsf{b}$$

.

Assumption sets

Contexts keep track of:

Assumption sets

Contexts keep track of:

1. Types of variables.

Assumption sets

Contexts keep track of:

1. Types of variables.
2. Which variables are available.

Assumption sets

Contexts keep track of:

1. Types of variables.
2. Which variables are available.

Annotations solve 1, but 2 needs care, e.g., if the user poses equality reflection rule

$$\frac{\Gamma \vdash A \;\mathsf{type} \quad \Gamma \vdash s : A \quad \Gamma \vdash t : A \quad \Gamma \vdash p : \mathsf{Eq}(A, s, t)}{\Gamma \vdash s \equiv t : A}$$

then p (and its potential variables) is not recorded in the conclusion.

Assumption sets

Contexts keep track of:

1. Types of variables.
2. Which variables are available.

Annotations solve 1, but 2 needs care, e.g., if the user poses equality reflection rule

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma \vdash s : A \quad \Gamma \vdash t : A \quad \Gamma \vdash p : \mathsf{Eq}(A, s, t)}{\Gamma \vdash s \equiv t : A}$$

then p (and its potential variables) is not recorded in the conclusion. Tracking used variables: **assumption sets**.

$$A \equiv B \text{ by } \alpha \qquad a \equiv b \,:\, A \text{ by } \alpha$$

Assumption sets α consist of:

- free variables
- bound variables
- meta-variables

Conversions

Explicit conversion in terms:

$$\frac{\vdash A \text{ type} \qquad \vdash B \text{ type} \qquad \vdash t : A \qquad \vdash A \equiv B \text{ by } \alpha}{\vdash (t : B \text{ by } \alpha) : B}$$

Conversions

Explicit conversion in terms:

$$\frac{\vdash A \textit{ type} \qquad \vdash B \textit{ type} \qquad \vdash t : A \qquad \vdash A \equiv B \textit{ by } \alpha}{\vdash (t : B \textit{ by } \alpha) : B}$$

Choices:

Example (Congruence rule for Π)

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

Conversions

Explicit conversion in terms:

$$\frac{\vdash A \textit{ type} \qquad \vdash B \textit{ type} \qquad \vdash t : A \qquad \vdash A \equiv B \textit{ by } \alpha}{\vdash (t : B \textit{ by } \alpha) : B}$$

Choices:

Example (Congruence rule for Π)

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

$$\frac{\vdash A \equiv A' \textit{ by } \alpha \quad \vdash \{x{:}A\}B(x) \equiv B'(x) \textit{ by } \beta}{\vdash \Pi(A, \{x\}B(x)) \equiv \Pi(A', \{x\}B'(x : A \textit{ by } \alpha))}$$

$$\frac{\vdash A \equiv A' \textit{ by } \alpha \quad \vdash \{x{:}A\}B(x) \equiv B'(x : A' \textit{ by } \alpha) \textit{ by } \beta}{\vdash \Pi(A, \{x\}B(x)) \equiv \Pi(A', \{x\}B'(x))}$$

Finitary Type Theories

Summary:

- Free variables annotated with their types a^A.
- Bound variables abstracted with an explicit abstraction.
- Assumption sets.
- Explicit conversions in terms.

Overview of the algorithm

Mutually recursive sub-algorithms:

- **Normalize a type** A
- **Normalize a term** t **of type** A
- **Check equality of types** $A \equiv B$
- **Check equality of normal types** $A \equiv B$
- **Check equality of terms** s **and** t **of type** A
  1. **type-directed phase**
  2. **normalization phase**
- **Check equality of normal terms** s **and** t **of type** A

Normalization

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

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

Equality checking

- **Check equality of types $A \equiv B$:** A and B are normalized and their normal forms are compared.
- **Check equality of normal types $A \equiv B$:** compare structurally - apply a congruence rule. Proceed recursively on the (normalizing) arguments.
- **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 compare their normal forms.
- **Check equality of normal terms s and t of type A:** normal terms are compared structurally.

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

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

where

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

Example (Extensionality rule for dependent functions[1])

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

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

Example (Dependent functions)

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

Normal forms

Definition

An expression is in *normal form* if

- no computation rules apply,
- its *normalizing arguments* are in normal form.

Normal forms

Definition

An expression is in *normal form* if

- no computation rules apply,
- its *normalizing arguments* are in normal form.

Selecting normalizing arguments specifies what is a (weak) normal form.

In Andromeda 2: normalizing arguements for $\mathsf{s}(u_1, \ldots, u_n)$ are those u_i's that are *not* meta-variables.

Example (Computation rule for *app*)

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

Andromeda marks just the third argument of *app* as normalizing argument.

Normalizing abstracted arguments

Example

How to normalize $\prod(A, \{x\}\, B(x))$.

Normalizing abstracted arguments

Example

How to normalize $\prod(A, \{x\}\, B(x))$.

1. Normalize A to get $\vdash A \equiv A'$ *by* α.

Normalizing abstracted arguments

Example

How to normalize $\prod(A, \{x\}\, B(x))$.

1. Normalize A to get $\vdash A \equiv A'$ *by* α.
2. Normalize $\{x{:}A\}\, B(x)$ to get $\vdash \{x{:}A\}B(x) \equiv B'(x)$ *by* β

Normalizing abstracted arguments

Example

How to normalize $\prod(A, \{x\}\, B(x))$.

1. Normalize A to get $\vdash A \equiv A'$ *by* α.
2. Normalize $\{x{:}A\}\, B(x)$ to get $\vdash \{x{:}A\}B(x) \equiv B'(x)$ *by* β
3. Convert x in $B'(x)$ to get

$$\vdash \prod(A', \{x\}\, B'[(x : A \textit{ by } \alpha)/x]) \textit{ type}$$

Normalizing abstracted arguments

Example

How to normalize $\prod(A, \{x\}\, B(x))$.

1. Normalize A to get $\vdash A \equiv A'$ *by* α.
2. Normalize $\{x{:}A\}\, B(x)$ to get $\vdash \{x{:}A\}B(x) \equiv B'(x)$ *by* β
3. Convert x in $B'(x)$ to get

$$\vdash \prod(A', \{x\}\, B'[(x : A \textit{ by } \alpha)/x]) \textit{ type}$$

4. Apply congruence rule and combine into

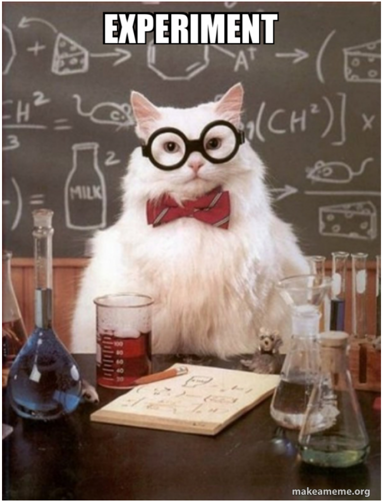
$$\vdash \prod(A, \{x\}\, B(x)) \equiv \prod(A', \{x\}\, B'(x : A \textit{ by } \alpha)) \textit{ by } (\beta \backslash \{x\})$$

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.

Demo in Andromeda

- Implemented-in-Ocaml in 1300 lines.
- Outside of trusted nucleus.
- Each equality step certified by nucleus.



Demo in Andromeda

```
require eq ;;
rule Π (A type) ({x : A} B type) type ;;
rule lambda (A type) ({x : A} B type) ({x : A} e : B{x}) : Π A B ;;
rule app (A type) ({x : A} B type) (s : Π A B) (a : A) : B{a} ;;

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} ;;

eq.add_rule Π_beta;;

rule Π_ext (A type) ({x : A} B type) (f : Π A B) (g : Π A B) ({x : A} app A B f x == app A B g x : B{x})
 : f == g : Π A B;;

eq.add_rule Π_ext;;

let eta = derive (A type) ({x : A} B type) (f : Π A B) ->
 eq.prove (f == lambda A B ({a : A} app A B f a) : Π A B by ??) ;;
```