Formalization of 2LTT in Agda

Interplay Between
Strict Equality and Propositional Equality

Elif Uskuplu

University of Southern California

Outline of The Talk

- Motivation for the experiment
- `--two-level` and `--cumulativity` flags
- Exo-types, types, exo-equality ($=^e$)
- Interplay between ($=^e$) and Id-type ($\equiv$)

Motivation

- Two-level type theory (2LTT)[1] addresses the problem that certain higher-categorical structures cannot be suitably encoded in HoTT (e.g. semisimplicial types).
- A recent work[2] about a general version of univalence principle that applies to all set-based, categorical, and higher-categorical structures, makes use of 2LTT.
- The authors of the work left the formalization of their results as an open project. This is the current project[3] of the speaker ☺.

[1] ACKS19. 2LTT and Applications. arXiv:1705.03307
[2] ANST21. The Univalence Principle. arXiv:2102.06275
[3] `https://github.com/ElifUskuplu/2LTT-Agda`

Experimental Flags in Agda

2LTT offers another kind of universe for non-fibrant types. In Agda, the flag `--two-level` enables a new sort `SSet` of strict types. With the usual sort `Set`, we obtain a distinction between fibrant and non-fibrant types. Currently, there is no documentation about the flag.

```
{-# OPTIONS --two-level #-}

open import Agda.Primitive public

--exo-universe of exotypes
UUᵉ : (i : Level) → SSet (lsuc i)
UUᵉ i = SSet i

--universe of types
UU : (i : Level) → Set (lsuc i)
UU i = Set i
```

Experimental Flags in Agda

As in the UP paper, it makes sense to assume that any fibrant type is a 'non-fibrant' type. In other words, we can regard `Set` as a subtype of `SSet`. In Agda, the flag `--cumulativity` enables this property. Although, there is a documentation about it, the flag is still in progress and is subject to change.

```
{-# OPTIONS --two-level
            --cumulativity #-}

lift : {i : Level} → UU i → UUᵉ i
lift A = A
```

Experimental Flags in Agda

!!! Using `--two-level` and `--cumulativity` together causes some undesired results which can be considered as bugs. For the details, one can look at the issues 5948 and 5761 in Agda Github[4]. We'll discuss it later in details.

[4] https://github.com/agda/agda

Types & Exo-Types

- Following the notations in the UP paper, we refer to non-fibrant types as **exo-types**[5] and reserve the word **type** to refer to the fibrant ones.
- $\mathcal{U}$ stands for the universe of (fibrant) types, and $\mathcal{U}^e$ for the universe of exo-types.
- Each type former is defined twice; one for types, one for exo-types. Whenever it is needed, we make distinction between them using the superscript $-^e$.

[5] This term was originally suggested by Ulrik Buchholtz.

Example

```
{-# OPTIONS --two-level #-}

--Type former of dependent pairs for exotypes
record Σᵉ {i j}
        (A : UUᵉ i)
        (B : A → UUᵉ j) :
        UUᵉ (i ⊔ j) where
  constructor _,ᵉ_
  field
    pr1ᵉ : A
    pr2ᵉ : B pr1ᵉ

--Type former of dependent pairs for types
record Σ {i j}
        (A : UU i)
        (B : A → UU j) :
        UU (i ⊔ j) where
  constructor _,_
  field
    pr1 : A
    pr2 : B pr1
```

```
{-# OPTIONS --two-level
            --cumulativity #-}

module _
  {i : Level}
  {Aᵉ : UUᵉ i} {Bᵉ : Aᵉ → UUᵉ i}
  {A : UU i} {B : A → UU i}
  {Cᵉ : Aᵉ → UU i}
  {C : A  → UUᵉ i} where

--These two are usual.
Type-1 = Σᵉ Aᵉ Bᵉ
Type-2 = Σ A B
--These three are valid only
--when --cumulativity assumed.
Type-3 = Σᵉ A B
Type-4 = Σᵉ A C
Type-5 = Σᵉ Aᵉ Cᵉ
--This is by no means valid.
Type-6 = Σ Aᵉ Bᵉ
```

Two notions of equality

- Exo-equality ($=^e$)
- Usual identity type ($\equiv$)
- If A is a (fibrant) type and $a, b : A$, we have a map $=^e$`-to-`$\equiv : a =^e b \to a \equiv b$, but not vice-versa.
- We assume *"the axiom of uniqueness of identity proofs"* for $=^e$. Note that Agda allows us to prove it because `--without-K` flag only disables this for `Set` but still allows it for `SSet`.

```
{-# OPTIONS --without-K
            --two-level
            --cumulativity #-}

--exo(strict)equality for exotypes
data _=ᵉ_ {i : Level}{A : UUᵉ i}
          (x : A) : A → UUᵉ i where
  reflᵉ : x =ᵉ x

UIPᵉ : {i : Level}{A : UUᵉ i}{x y : A}
       (p q : x =ᵉ y) → p =ᵉ q
UIPᵉ reflᵉ reflᵉ = reflᵉ

--usual identity type
data _≡_ {i : Level}{A : UU i}
          (x : A) : A → UU i where
  refl : x ≡ x

--If two terms are exo-equal,
--they are also path equal.
=ᵉ-to-≡ : {i : Level}{A : UU i}{x y : A}
          → x =ᵉ y → x ≡ y
=ᵉ-to-≡ reflᵉ = refl
```

Details about the issue with flags

- Since 2LTT does not assume elimination from a fibrant type to a non-fibrant one, we expect that we are not able to define maps like $\mathbb{N} \to \mathbb{N}^e$ and $+ \to +^e$ in Agda.
- For example, the possible inverse of $=^e$`-to-`$\equiv$ would destroy the main motivation for 2LTT.
- With the power of `--two-level`, Agda knows the distinction between exo-types and types, and prevents defining the maps we don't desire.
- However, using `--cumulativity`, we can lift a type to the exo-universe that enables the maps we don't expect.
- As Andreas Abel said

  The sort of a type is no longer a well-defined concept.

Details about the issue with flags

```
{-# OPTIONS --two-level
            --cumulativity #-}

ℕᵉ-to-ℕ : ℕᵉ → ℕ
ℕᵉ-to-ℕ zeroᵉ = zero
ℕᵉ-to-ℕ (succᵉ n) = succ (ℕᵉ-to-ℕ n)

ℕ-to-ℕᵉ : ℕ → ℕᵉ
ℕ-to-ℕᵉ n = { }0
--Cannot eliminate fibrant type ℕ
--unless target type is also fibrant
--when checking that the expression {}0 has type ℕᵉ

liftℕ : UUᵉ lzero
liftℕ = ℕ

liftℕ-to-ℕᵉ : liftℕ → ℕᵉ
liftℕ-to-ℕᵉ zero = zeroᵉ
liftℕ-to-ℕᵉ (succ n) = succᵉ (liftℕ-to-ℕᵉ n)
```

Two notions of equality

- Exo-equality should regarded as a sort of "metatheoretic" or "syntactic" equality.
- Since exo-equality is assumed to satisfy UIP, we have all exo-types are h-sets in terms of type hierarchy.
- We can define each operation related to equalities as usual, but emphasizing the difference between $=^e$ and $\equiv$.
- We assume function extensionality (funext) for both equalities.
- We assume the univalence (UA) only for $\equiv$ because it is incompatible with UIP.

Exo-isomorphisms and Equivalences

- We say that a function $f : A \to B$ between exotypes is an **exo-isomorphism** if there is $g : B \to A$ such that $g \circ f =^e 1_A$ and $f \circ g =^e 1_B$.
- We say that a function $f : A \to B$ between (fibrant) types is an **equivalence** if there is $g : B \to A$ such that $g \circ f \equiv 1_A$ and $f \circ g \equiv 1_B$.
- Using $=^e$`-to-`$\equiv$ it is easy to see that an exo-isomorphism between (fibrant) types is also an equivalence.

- In the formalization, we make use of funext.

```
module _
  {i j : Level}
  {A : UUᵉ i} {B : UUᵉ j}
  {C : UU i} {D : UU j}
  where

  is-exo-iso : (f : A → B) → UUᵉ (i ⊔ j)
  is-exo-iso f = Σᵉ (B → A)
                    (λ g → ((a : A) → (g ∘ᵉ f) a =ᵉ a) ×ᵉ
                            ((b : B) → (f ∘ᵉ g) b =ᵉ b))
  _≅_ : UUᵉ (i ⊔ j)
  _≅_ = Σᵉ (A → B) is-exo-iso

  is-equiv : (f : C → D) → UU (i ⊔ j)
  is-equiv f = Σ (D → C)
                  (λ g → ((c : C) → (g ∘ f) c ≡ c) ×
                          ((d : D) → (f ∘ g) d ≡ d))
  _≃_ : UU (i ⊔ j)
  _≃_ = Σ (C → D) is-equiv
```

Fibrant Exo-types!

- We call an exo-type $A : \mathcal{U}^e$ **fibrant** if it is exo-isomorphic to a type $B : \mathcal{U}$.
- Clearly, every type is a fibrant exo-type.

```
{-# OPTIONS --without-K
            --two-level
            --cumulativity #-}

record isFibrant {i : Level}(B : UUᵉ i) :
       UUᵉ (lsuc i) where
  constructor isfibrant
  field
    fibrant-match : UU i
    fibrancy-witness : B ≅ fibrant-match

--every type is fibrant
type-isFibrant : {i : Level} (A : UU i)
                 → isFibrant A
type-isFibrant A = isfibrant A (id-iso A)
```

Fibrant Maps!

- If $f : A \to B$ is a map of fibrant exo-types, we can lift to a map between their fibrant matches

$$\begin{tikzcd} A \arrow[rr, "f"] \arrow[d, bend right, "\phi"'] & & B \arrow[d, bend right, "\psi"'] \\ FA \arrow[u, bend right, "\phi^{-1}"'] \arrow[rr, dashed, "\text{Fib-map}(f)"'] & & FB \arrow[u, bend right, "\psi^{-1}"'] \end{tikzcd}$$

- We call f equivalence if the lift $\text{Fib-map}(f) = \psi \circ f \circ \phi^{-1}$ is an equivalence.

```
Fib-map : {i j : Level} {A : UUᵉ i} {B : UUᵉ j}
          (P : isFibrant A) (Q : isFibrant B)
          → (F : A → B)
          → isFibrant.fibrant-match P
          → isFibrant.fibrant-match Q
Fib-map {A = A} {B = B} P Q F = ψ ∘ᵉ (F ∘ᵉ ϕ⁻¹)
   where
   ϕ⁻¹ : isFibrant.fibrant-match P → A
   ϕ⁻¹ = pr1ᵉ (pr2ᵉ (isFibrant.fibrancy-witness P))

   ψ : B → isFibrant.fibrant-match Q
   ψ = pr1ᵉ (isFibrant.fibrancy-witness Q)
```

Properties

- Fib-map(_) preserves identity maps and compositions.
- If f is an exo-isomorphism between fibrant exo-types, then f is an equivalence.
- If f and g are homotopic maps between fibrant exo-types, then f is an equivalence $\Leftrightarrow$ g is.
- 2-out-of-3, 3-out-of-4 properties, etc.
- All these properties above are obtained thanks to $=^e$`-to-`$\equiv$ conversion. The interplay between $=^e$ and $\equiv$ has many other useful corollaries. One of these is a new kind of function extensionality!

Function Extensionality

Depending on taking a type (family) or an exo-type (family), one can obtain a different $\prod / \prod^e$-type.

- If $A : \mathcal{U}$ and $B : A \to \mathcal{U}$, then we have $\prod_A B : \mathcal{U}$ and the function extensionality wrt $\equiv$.
- If $A : \mathcal{U}^e$ and $B : A \to \mathcal{U}^e$, we have $\prod_A^e B : \mathcal{U}^e$ and the function extensionality wrt $=^e$.
- What if $A : \mathcal{U}^e$ and $B : A \to \mathcal{U}$? We still have the function extensionality wrt $=^e$, but for $f, g : \prod_A^e B$, we also have $f(a) \equiv_{B(a)} g(a)$.

Question : What can be derived from $\prod_{a:A}^e f(a) \equiv_{B(a)} g(a)$?

Cofibrancy

- Recall that the funext for $\equiv$ is equivalent to that for any $B : A \to \mathcal{U}$ we have
$$\prod_{a:A} \text{isContr}(B(a)) \to \text{isContr}\left(\prod_{a:A} B(a)\right).$$
- In 2LTT, we have another notion weaker than fibrancy, which is called **cofibrancy**.
- An exo-type $A : \mathcal{U}^e$ is called cofibrant, if for any $B : A \to \mathcal{U}$, the exo-type $\prod^e_A B$ is fibrant, and moreover if each $B(a)$ is contractible, so is the fibrant match of $\prod^e_A B$.
- We can use the notion to give an answer for the previous question.

Nice example of the interplay between $=^e$ and $\equiv$

Proposition.(Funext for cofibrant exo-types) Assume $A : \mathcal{U}^e$ is a cofibrant exo-type and $B : A \to \mathcal{U}$. Let FM be the fibrant match of $\prod_A^e B$, and $\beta : \prod_A^e B \to FM$ be the exo-isomorphism. Then we have

$$\left[\prod_{a:A}^e \left(f(a) \equiv_{B(a)} g(a)\right)\right] \to (\beta(f) \equiv_{FM} \beta(g)).$$

Proof

```
module FUNEXT {i j : Level}{A : UUᵉ i}
      {B : A → UU j} {P : isCofibrant {i} A j}
      where
  FM = Π-fibrant-witness (P B)
  α : FM → Πᵉ A B
  β : Πᵉ A B → FM
  βα : (X : FM) → (β ∘ᵉ α) X =ᵉ X
  αβ : (X : Πᵉ A B) → (α ∘ᵉ β) X =ᵉ X

  FEP : {f g : Πᵉ A B}
      → ((x : A) → Id (f x) (g x))
      → Id (β f) (β g)
  FEP {f} {g} T = ?
```

```
Y : A → UU j
Y x = Σ (B x) (λ y → Id y (f x))

f' g' : Πᵉ A Y
f' x = (f x , refl)
g' x = (g x , (T x) ⁻¹)

fibers-of-Y-is-contr : (x : A) → is-contr (Y x)
fibers-of-Y-is-contr x =
  path-type-is-contr {j} {B x} (f x)

WFEP = contr-preserve-witness (P Y)

Πtype = Π-fibrant-witness (P Y)
α' : Πtype → Πᵉ A Y
β' : Πᵉ A Y → Πtype
βα' : (X : Πtype) → (β' ∘ᵉ α') X =ᵉ X
αβ' : (X : Πᵉ A Y) → (α' ∘ᵉ β') X =ᵉ X
```

Proof

```
p' : Id (β' f') (β' g')
p' = (pr2 (WFEP fibers-of-Y-is-contr) (β' f')) · ((pr2 (WFEP fibers-of-Y-is-contr) (β' g')) ⁻¹)

p : Id (β f) (β g)
p = =ᵉ-to-Id (exo-ap β (funextᵉ λ a → exo-inv (exo-ap pr1 (happlyᵉ (αβ' f') a))))
         · (ap (λ u → β (λ a → pr1 ((α' u) a))) p' ·
    =ᵉ-to-Id (exo-ap β (funextᵉ λ a → (exo-ap pr1 (happlyᵉ (αβ' g') a)))))
```

Thanks!