Groupoid Infinity
Kostiantynivska 20/14, Kyiv, Ukraine 04071

Identity Type Encoding

Paul Lyutko, Maxim Sokhatsky

Groupoid Infinity
Kropyvnytsky 2016

Contents

1 Abstract

In this article Gropoid Infinit will show how to encode classical Identity Types used in Type Theory using EXE language. We provide brief explanation of Identity Type properties and our motivation why use Setoid Types instead of classical Identity Types from Type Theory.

```
record Id (A: Type): A → A → Type :=
       (refl (a: A): Id a a)
```


2 Identity Type

2.1 Typing Rules

$$\frac{a : A \quad b : A \quad A : \mathit{Type}}{Id(A, a, b) : \mathit{Type}} \tag{Id-formation}$$

$$\frac{a : A}{refl(A, a) : Id(A, a, a)} \tag{Id-intro}$$

$$\frac{p : Id(a, b) \quad x, y : A \quad u : Id(x, y) \vdash E : \mathit{Type} \quad x : A \vdash d : E\ [x/y,\ refl(x)/u]}{J(a, b, p, (x, y, u)\ d) : E\ [a/x,\ b/y,\ p/u]} \tag{J-elimination}$$

$$\frac{a, x, y : A, \quad u : Id(x, y) \vdash E : \mathit{Type} \quad x : A \vdash d : E\ [x/y,\ refl(x)/u]}{J(a, a, refl(a), (x, y, u)\ d) = d\ [a/x] : E\ [a/y,\ refl(a)/u]} \tag{Id-computation}$$

```
record Id (A: Type): Type :=
       (Id: A → A → Type)
       (refl (a: A): Id a a)
       (Predicate := ∀ (x,y: A) → Id x y → Type)
       (Forall (C: Predicate) := ∀ (x,y: A) → ∀ (p: Id x y) → C x y p)
       (Δ (C: Predicate) := ∀ (x: A) → C x x (refl x))
       (axiom-J (C: Predicate): Δ C → Forall C)
       (computation-rule (C: Predicate) (t: Δ C):
                           ∀ (x: A) → (J C t x x (refl x)) ==> (t x))
```

2.2 Symmetry and Transitivity

```
record Properties (A: Type): Type :=
       (Trans (a,b,c: A) : Id a b → Id b c → Id a c :=
                           Id.axiom-J (λ a b p1 → Id b c → Id a c) (λ x p2 → p2) a ab)
       (Sym (a,b: A)     : Id a b → Id b a :=
                           Id.axiom-J (λ a b p → Id b a) Id.refl a b)
```

2.3 Substitution in Predicates

```
record Subst (A: Type): Type :=
       (intro (P (a: A): Type) (a1, a2: A) : Id a1 a2 → P a1 → P a2 :=
        Id.axiom-J (λ a1 a2 p12 → P a1 → P a2) (λ a0 p0 → p0) a1 a2)
```

2.4 Uniqueness of Identity Proofs

```
record UIP (A: Type): Type :=
       (intro (A: Type) (a,b: A) (x,y: Id a b) : Id (Id A a b) x y)
```

2.5 Axiom K

```
record K (A: Type): Type :=
       (Predicate_K := ∀ (a: A) → Id a a → Type)
       (Forall_K (C: Predicate_K) := ∀ (a: A) → ∀ (p: Id a a) → C a p)
       (Δ_K (C: Predicate_K) := ∀ (a: A) → C a (Id.refl a))
       (axiom-K (C: Predicate): Δ_K C → Forall_K C)
```

2.6 Congruence

```
define Respect_1 (A,B: Type) (C: A → Type) (D: B → Type) (R: A → B → Prop)
       (Ro: ∀ (x: A) (y: B) → C x → D y → Prop) :
            (∀ (x: A) → C x) → (∀ (x: B) → D x) → Prop :=
                   λ (f g: Type → Type) → (∀ (x y) → R x y) → Ro x y (f x) (g y)

record Respect_2 (A: Type): Type :=
       (intro (A,B: Type) (f: A → B) (a,b: A) : Id A a b → Id B (f a) (f b) :=
              Id.axiom-J (λ a b p12 → Id B (f a) (f b)) (λ x → refl B (f x)) a b)
```

2.7 Setoid

```
record Setoid: Type :=
       (Carrier: Type)
       (Equ: Carrier → Carrier → Prop)
       (Refl:  ∀ (e_0: Carrier) → Equ e_0 e_0)
       (Trans: ∀ (e_1,e_2,e_3: Carrier) → Equ e_1 e_2 → Equ e_2 e_3 → Equ e_1 e_3)
       (Sym:   ∀ (e_1 e_2: Carrier) → Equ e_1 e_2 → Equ e_2 e_1)
```

2.8 Conclusion

As you can see EXE language has enough expressive power to be used for drawing MLTT axioms in computer science articles and papers.

References

[1] E.Bishop *Foundations of Constructive Analysis* 1967

[2] T.Streicher, M.Hofmann *The groupoid interpretation of type theory* 1996

[3] G.Barthe, V.Capretta *Setoids in type theory* 2003

[4] M.Sozeau, N.Tabareau *Internalizing Intensional Type Theory* 2013

[5] V.Voevodsky *Identity types in the C-systems defined by a universe category* 2015

[6] D.Selsam and L.de Moura *Congruence Closure in Intensional Type Theory* 2016