Groupoid Infinity
Kostiantynivska 20/14, Kyiv, Ukraine 04071

Identity Type Encoding

Paul Lyutko, Maxim Sokhatsky

Groupoid Infinity
Kropyvnytsky 2016

Contents

1 Abstract **3**

2 Identity Type **3**

2.1 Typing Rules 3
2.2 Symmetry and Transitivity 4
2.3 Substitution in Predicates 4
2.4 Uniqueness of Identity Proofs 4
2.5 Congruence 4
2.6 Setoid 4
2.7 Conclusion 4

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

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

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

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

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

```
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 Congruence

```
define Respect (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 (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)
```

2.6 Setoid

```
record Setoid: Type :=
       (Carrier: Type)
       (Equ: Carrier → Carrier → Prop)
       (Refl:  ∀ (e0: Carrier) → Equ e0 e0)
       (Trans: ∀ (e1,e2,e3: Carrier) → Equ e1 e2 → Equ e2 e3 → Equ e1 e3)
       (Sym:   ∀ (e1 e2: Carrier) → Equ e1 e2 → Equ e2 e1)
```

2.7 Conclusion

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