Структура доклада.

1. Структура доклада.
2. Введение в Agda и примеры.
3. Введение в Coq и примеры.
4. Элементы гомотопической теории типов в Coq.
5. Элементы гомотопической теории типов в Agda.

Синтаксис Agda.

- Сравним два модуля, в которых написаны одинаковые функции. Первый модуль написан на языке Haskell, второй — на Agda:

Синтаксис Agda.

```
data Nat = Zero | Suc Nat

fromNat :: Nat -> Integer
fromNat Zero = 0
fromNat (Suc n) = fromNat n + 1

toNat :: Integer -> Nat
toNat 0 = Zero
toNat x = Suc (toNat(x - 1))

add :: Nat -> Nat -> Nat
add = \x y -> toNat (fromNat x +
fromNat y)

mul :: Nat -> Nat -> Nat
mul = \x y -> toNat (fromNat x *
fromNat y)
```

```
data ℕ : Set where
 zero : ℕ
 suc : (n : ℕ) → ℕ

_+_ : ℕ → ℕ → ℕ
zero + n = n
suc m + n = suc (m + n)

_*_ : ℕ → ℕ → ℕ
zero * n = zero
suc m * n = m * n + m
```

Синтаксис Agda.

```
fib : ℕ → ℕ
fib zero = zero
fib (suc zero) = suc zero
fib (suc (suc n)) = fib (suc n) + fib n

fac : ℕ → ℕ
fac zero = suc zero
fac (suc n) = suc n * fac n
```

Аналогично в Haskell.

```
fib :: Nat -> Nat
fib = \ x -> toNat ( fibonacci ( fromNat x) where
 fibonacci n | n == 0 = 0
 | n == 1 = 1
 | n > 0 = fibonacci (n - 1) + fibonacci (n - 2)
 | otherwise = error ''arg must be >=0''

fac :: Nat -> Nat
fac = \x -> toNat ( factorial ( fromNat x)) where
 factorial n | n == 0 = 1
 | n > 0 = n * factorial(n-1)
 | otherwise = error ''arg must be >=0''
```

Синтаксис Agda.

```
data Vec {a} (A : Set a) : ℕ → Set a where
 [] : Vec A zero
 _::_ : ∀ {a} (x : A) (xs : Vec A n) → Vec A (suc n)

head : ∀ {a n} {A : Set a} → Vec A (1 + n) → A
head (x :: xs) = x

tail : ∀ {a n} {A : Set a} → Vec A (1 + n) → A
tail (x :: xs) = xs

[_] : ∀ {a} {A : Set a} → A → Vec A 1
[ x ] = x :: []

_++_ : ∀ {a m n} {A : Set a} → Vec A m → Vec A n → Vec A (m + n)
[] ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)
```

Как работает Agda.

Рассмотрим более сложный пример. Введем теорию первого порядка с сигнатурой $\langle e, \cdot, =, ^{-1} \rangle$ с аксиомами:

1. $\forall x(x = x)$;
2. $\forall x \, \forall y(x = y \supset y = x)$;
3. $\forall x \, \forall y \, \forall z(x = y \wedge y = z \supset x = z)$;
4. $\forall x_1 \, \forall x_2 \, \forall y_1 \, \forall y_2(x_1 = y_1 \wedge x_2 = y_2 \supset x_1 \cdot x_2 = y_1 \cdot y_2)$;
5. $\forall x \, \forall y \, \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$;
6. $\forall x \, \forall y(x \cdot y = y \cdot x)$;
7. $\forall x(x \cdot e = x = e \cdot x)$;
8. $\forall x(x \cdot x^{-1} = x^{-1} \cdot x = e)$.

Такая теория с заданными сигнатурой и аксиомами называется теорией абелевых групп.

Как работает Agda.

- Определим в синтаксисе Agda абелеву группу:

```
record AbelianGroup c ℓ : Set (suc (c ⊔ ℓ)) where
infix 8 _ ⁻¹
infixl 7 _·_
infix 4 _≈_
Carrier : Set c
_≈_ : Rel Carrier ℓ
_·_ : Op₂ Carrier
e : Carrier
_ ⁻¹ : Op₁ Carrier
isAbelianGroup : IsAbelianGroup _≈_ _·_ e _ ⁻¹
```

Как работает Agda.

```
record AbelianGroup (A : Setoid) : Set
where
open Setoid A using (_≈_) renaming (Carrier to G)
field
_·_ : G → G → G
cong· : (x y x' y' : G) → x ≈ x' → y ≈ y' → (x · y) ≈ (x' · y')
assoc· : (x y z : G) → (x · y) · z ≈ x · (y · z)
unit· : (x : G) → x · e ≈ e · x ≈ x
inv· : (x : G) → x · x⁻¹ ≈ x⁻¹ · x ≈ e
com· : (x y : G) → x · y = y · x
```

Как работает Agda.

```
int+abeliangroup : AbelianGroup Int.setoid
AbelianGroup = { _·_ = _+_, e = 0, _⁻¹ = -(_), cong· = cong+, assoc· =
assoc+, unit· = unit +, inv· = int+, com· = com+ }
_+_ : ℤ → ℤ → ℤ
-[1+ m ] + -[1+ n ] = -[1+ ℕ.suc (mℕ+ n) ]
-[1+ m ] + + n = n ⊖ ℕ.suc m
+ m + -[1+ n ] = m ⊖ ℕ.suc n
+ m + + n = + (m ℕ+ n)
assoc+ : (x y z : Int) -> (x + y) + z = x + (y + z)
assoc+ y z = refl
assoc+ (x + 1) y z =
begin
((x + 1) + y) + z
((x + y) + z) + 1
(x + (y + z)) + 1
(x + 1) + (y + z)
```

Определения в Coq из общей топологии (По А. Бауэру).

1'. Синглтон
2'. Подмножество:
$A \supset B \Leftrightarrow \forall\, x \in A(x \in A \to y \in A)$;
3'. Объединение (разделенное):
$A \sqcup B \Leftrightarrow \forall x \neg(x \in A \wedge x \in B)$;
4'. Объединение:
$A \cup B = \{x \mid x \in A \vee x \in B\}$;
5'. Пересечение: $A \cap B = \{x \mid x \in A \wedge x \in B\}$;
6. $\emptyset$.

```
Definition P (A : Type) := A -> Prop.
1. Definition singleton {A : Type} (x
: A) := { y : A | x = y }.
2. Definition subset {A : Type} (u v :
P A) :=
forall x : A, u x -> v x.
3. Definition disjoint {A : Type} (u v
: P A) :=
 forall x, ~ (u x /\ v x).
4. Definition union {A : Type} (S : P
(P A)) :=
{ x : A | some U : S , U x }.
5. Definition inter A : Type (u v : P
A) := { x : A | u x /\ v x }.
6. Definition empty {A : Type} := { x
: A | False }.
```

Определения в Coq из общей топологии (По А. Бауэру).

7. A — это множество, Ω — набор его подмножеств.
(A, Ω) — топологическое пространство, если выполнены три условия (аксиомы топологической структуры):
1) $A_1, A_2, A_3, ... \in \Omega \Leftrightarrow \bigcup\limits_{i=1} A_i \in \Omega$;
2) $A_1, A_2, A_3, ..., A_n \in \Omega \Leftrightarrow \bigcap\limits_{i=1}^{n} A_i \in \Omega$;
3) $A, \emptyset \in \Omega$

```
7'. Structure topology (A : Type) := {
 open :> P A -> Prop ;
 empty_open : open empty ;
 full_open : open full ;
 inter_open : all u : open, all v :
open, open (u * v) ;
 union_open : forall S, S <= open ->
open (union S)
}.
```

Определения в Coq из общей топологии (По А. Бауэру).

8. База топологии: совокупность открытых подмножеств A называется базой топологии, $\mathfrak{B}$ если любое открытое подмножество представимо как объединение определенного набора множеств $\mathfrak{B}$.

8'.
```
Definition base {A : Type} (B : P
(P A)) :
 B full -> (all u : B, all v : B, B
(u * v)) -> topology A.
```

Определения в Coq из общей топологии (По А. Бауэру).

9. Аксиома Фреше (аксиома T_1): для любых $x \neq y \in A$ можно найти такую окрестность $O_\varepsilon(x)$, что $y \notin O_\varepsilon(x)$;
10. Топологическое пространство называется хаусдорфовым, если каждая пара неравных точек обладает непересекающимися окрестностями (то есть выполняется аксиома отделимости T_2 (аксиома Хаусдорфа)) ;

9'.
```
Definition T1 {A : Type} (T :
topology A) :=
 forall x y : A,
 x <> y ->
 some u : T, (u x /\ ~ (u y)).
```
10'.
```
Definition hausdorff {A : Type} (T
: topology A) :=
 forall x y : A,
 x <> y ->
 some u : T, some v : T,
 (u x /\ v y /\ disjoint u v).
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
```

```
 1 subgoal
_____________________________(1/1)
forall n m : nat, n + m = m + n
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
```

```
1 subgoal
n, m : nat
_____________________________(1/1)
n + m = m + n
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
```

```
2 subgoals
n, m : nat
_____________________________(1/2)
0 + m = m + 0
_____________________________(2/2)
forall n0 : nat,
n0 + m = m + n0 -> S n0 + m = m + S
n0
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
elim m.
```

```
3 subgoals
n, m : nat
______________________________(1/3)
0 + 0 = 0 + 0
______________________________(2/3)
forall n0 : nat,
0 + n0 = n0 + 0 -> 0 + S n0 = S n0 +
0
______________________________(3/3)
forall n0 : nat,
n0 + m = m + n0 -> S n0 + m = m + S
n0
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
elim m.
exact (eq_refl (0 + 0)).
```

```
2 subgoals
n, m : nat
_____________________________(1/2)
forall n0 : nat,
0 + n0 = n0 + 0 -> 0 + S n0 = S n0 +
0
_____________________________(2/2)
forall n0 : nat,
n0 + m = m + n0 -> S n0 + m = m + S
n0
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
elim m.
exact (eq_refl (0 + 0)).
intros m'.
```

```
2 subgoals
n, m, m' : nat
_____________________________(1/2)
0 + m' = m' + 0 -> 0 + S m' = S m' +
0
_____________________________(2/2)
forall n0 : nat,
n0 + m = m + n0 -> S n0 + m = m + S
n0
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
elim m.
exact (eq_refl (0 + 0)).
 intros m'.
 intros inductive_hyp_m.
```

```
2 subgoals
n, m, m' : nat
inductive_hyp_m : 0 + m' = m' + 0
_____________________________(1/2)
0 + S m' = S m' + 0
_____________________________(2/2)
forall n0 : nat,
n0 + m = m + n0 -> S n0 + m = m + S
n0
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
elim m.
exact (eq_refl (0 + 0)).
 intros m'.
 intros inductive_hyp_m.
simpl.
```

```
2 subgoals
n, m, m' : nat
inductive_hyp_m : 0 + m' = m' + 0
_____________________________(1/2)
S m' = S (m' + 0)
_____________________________(2/2)
forall n0 : nat,
n0 + m = m + n0 -> S n0 + m = m + S
n0
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
elim m.
exact (eq_refl (0 + 0)).
 intros m'.
 intros inductive_hyp_m.
simpl.
 rewrite <- inductive_hyp_m.
```

```
2 subgoals
n, m, m' : nat
inductive_hyp_m : 0 + m' = m' + 0
_____________________________(1/2)
S m' = S (0 + m')
_____________________________(2/2)
forall n0 : nat,
n0 + m = m + n0 -> S n0 + m = m + S
n0
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
elim m.
exact (eq_refl (0 + 0)).
 intros m'.
 intros inductive_hyp_m.
simpl.
 rewrite <- inductive_hyp_m.
simpl.
```

```
2 subgoals
n, m, m' : nat
inductive_hyp_m : 0 + m' = m' + 0
_____________________________(1/2)
S m' = S m'
_____________________________(2/2)
forall n0 : nat,
n0 + m = m + n0 -> S n0 + m = m + S
n0
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
elim m.
exact (eq_refl (0 + 0)).
 intros m'.
 intros inductive_hyp_m.
simpl.
 rewrite <- inductive_hyp_m.
simpl.
exact (eq_refl (S m')).
```

```
1 subgoal
n, m : nat
_____________________________(1/1)
forall n0 : nat,
n0 + m = m + n0 -> S n0 + m = m + S
n0
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
elim m.
exact (eq_refl (0 + 0)).
 intros m'.
 intros inductive_hyp_m.
simpl.
 rewrite <- inductive_hyp_m.
simpl.
exact (eq_refl (S m')).
 intros n'.
```

```
1 subgoal
n, m, n' : nat
_____________________________(1/1)
n' + m = m + n' -> S n' + m = m + S
n'
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
elim m.
exact (eq_refl (0 + 0)).
 intros m'.
 intros inductive_hyp_m.
simpl.
 rewrite <- inductive_hyp_m.
simpl.
exact (eq_refl (S m')).
intros n'.
intros inductive_hyp_n.
```

```
1 subgoal
n, m, n' : nat
inductive_hyp_n : n' + m = m + n'
_____________________________(1/1)
S n' + m = m + S n'
```

Coq. Пример доказательства.

```
Theorem plus_sym : (forall n m, n + m
= m + n).
Proof.
intros n m.
elim n.
elim m.
exact (eq_refl (0 + 0)).
 intros m'.
 intros inductive_hyp_m.
simpl.
 rewrite <- inductive_hyp_m.
simpl.
exact (eq_refl (S m')).
intros n'.
intros inductive_hyp_n.
simpl.
```

```
1 subgoal
n, m, n' : nat
inductive_hyp_n : n' + m = m + n'
_____________________________(1/1)
S (n' + m) = m + S n'
```

Coq. Пример доказательства.

```
rewrite inductive_hyp_n
```

```
1 subgoal
n, m, n' : nat
inductive_hyp_n : n' + m = m + n'
_____________________________(1/1)
S (m + n') = m + S n'
```

Coq. Пример доказательства.

```
rewrite inductive_hyp_n
elim m.
```

```
2 subgoals
n, m, n' : nat
inductive_hyp_n : n' + m = m + n'
_____________________________(1/2)
S (0 + n') = 0 + S n'
_____________________________(2/2)
forall n0 : nat,
S (n0 + n') = n0 + S n' ->
S (S n0 + n') = S n0 + S n'
```

Coq. Пример доказательства.

```
rewrite inductive_hyp_n
elim m.
 simpl.
```

```
2 subgoals
n, m, n' : nat
inductive_hyp_ : n' + m = m + n'
_____________________________(1/2)
S n' = S n'
_____________________________(2/2)
forall n0 : nat,
S (n0 + n') = n0 + S n' ->
S (S n0 + n') = S n0 + S n'
```

Coq. Пример доказательства.

```
rewrite inductive_hyp_n
elim m.
 simpl.
 exact (eq_refl (S n')).
```

```
2 subgoals
n, m, n' : nat
inductive_hyp_n : n' + m = m + n'
_____________________________(1/1)
forall n0 : nat,
S (n0 + n') = n0 + S n' ->
S (S n0 + n') = S n0 + S n'
```

Coq. Пример доказательства.

```
rewrite inductive_hyp_n
elim m.
 simpl.
 exact (eq_refl (S n')).  intros m'.
```

```
1 subgoal
n, m, n' : nat
inductive_hyp_n : n' + m = m + n'
m' : nat
______________________________(1/1)
S (m' + n') = m' + S n' ->
S (S m' + n') = S m' + S n'
```

Coq. Пример доказательства.

```
rewrite inductive_hyp_n
elim m.
 simpl.
 exact (eq_refl (S n')).  intros m'.
 intros inductive_hyp_m.
```

```
1 subgoal
n, m, n' : nat
inductive_hypn : n' + m = m + n'
m' : nat
inductive_hyp_m : S (m' + n') = m' +
S n'
_____________________________(1/1)
S (S m' + n') = S m' + S n'
```

Coq. Пример доказательства.

```
rewrite inductive_hyp_n
elim m.
 simpl.
 exact (eq_refl (S n')).  intros m'.
 intros inductive_hyp_m.
 simpl.
```

```
1 subgoal
n, m, n' : nat
inductive_hypn : n' + m = m + n'
m' : nat
inductive_hyp_m : S (m' + n') = m' +
S n'
_____________________________(1/1)
S (S (m' + n')) = S (m' + S n')
```

Coq. Пример доказательства.

```
rewrite inductive_hyp_n
elim m.
 simpl.
 exact (eq_refl (S n')).  intros m'.
 intros inductive_hyp_m.
 simpl.
 rewrite inductive_hyp_m.
```

```
1 subgoal
n, m, n' : nat
inductive_hypn : n' + m = m + n'
m' : nat
inductive_hyp_m : S (m' + n') = m' +
S n'
______________________________(1/1)
S (m' + S n') = S (m' + S n')
```

Coq. Пример доказательства.

```
rewrite inductive_hyp_n
elim m.
 simpl.
 exact (eq_refl (S n')).  intros m'.
 intros inductive_hyp_m.                    No more subgoals
 simpl.
 rewrite inductive_hyp_m.
 exact (eq_refl (S (m' + S n')))
Qed.
```

Coq. Пример доказательства.

```
plus_sym =
fun n m : nat =>
nat_ind (fun n0 : nat => n0 + m = m + n0)
 (nat_ind (fun m0 : nat => 0 + m0 = m0 + 0) eq_refl
 (fun (m' : nat) (inductive_hyp_m : 0 + m' = m' + 0) =>
 eq_ind (0 + m') (fun n0 : nat => S m' = S n0) eq_refl
 (m' + 0) inductive_hyp_m) m)
 (fun (n' : nat) (inductive_hyp_n : n' + m = m + n') =>
 eq_ind_r (fun n0 : nat => S n0 = m + S n')
 (nat_ind (fun m0 : nat => S (m0 + n') = m0 + S n') eq_refl
 (fun (m' : nat) (inductive_hyp_m : S (m' + n') = m' + S n') =>
 eq_ind_r (fun n0 : nat => S n0 = S (m' + S n')) eq_refl inductive_hyp_m)
 m) inductive_hyp_n) n
 : forall n m : nat, n + m = m + n
```

Гомотопическая теория типов в Coq.

```
Definition idmap A := fun x : A => x.
Definition compose {A B C} (g : B -> C) (f : A -> B)(x : A) := g (f x).
Inductive paths {A} : A -> A -> Type := idpath : forall x, paths x x.
Inductive total {B : Type} (A : B -> Type) : Type := pair (x : B)(y : A x).
Definition dirprod {A B : Type} : Type := total (fun x : A => B).
```

Гомотопическая теория типов в Coq.

```
Ltac path_induction :=
intros; repeat progress (
match goal with
 | [p: _ ~~> _ |-_] => induction p
 | _ => idtac
 end
); auto.
```

Гомотопическая теория типов в Coq.

```
Definition concat {A} {x y z : A} : (x ∼∼> y) -> (y ∼∼> z) -> (x ∼∼> z).
Proof.
intros p q.
induction p.
induction q.
apply idpath.
Defined.
```

Гомотопическая теория типов в Coq.

```
Lemma concat_associativity A (w x y z : A) (p : w ~~> x) (q : x ~~> y) (r
: y ~~> z) : (p @ q) @ r ~~> p @ (q @ r).
Proof.
path_induction.
Defined.
Definition homotopy_concat A (x y z : A)(p p': x ~~> y)(q q': y ~~> z) :
(p ~~> p') -> (q ~~> q') -> (p @ q ~~> p' @ q').
Proof.
path_induction.
Defined.
Lemma map {A B} {x y : A} (f : A -> B) (p : x ~~> y) : f x ~~> f y.
Proof.
path_induction.
apply idpath.
Defined.
```

Гомотопическая теория типов в Coq.

```
Theorem transport {A} {P : A -> Type} {x y : A}(p : x ∼∼> y) : P x -> P y.
Proof.
pathinduction.
Defined.
```

Гомотопическая теория типов в Coq.

```
fun (A : Type) (w x y z : A) (p : w ~~> x) (q : x ~~> y) (r : y ~~> z) =>
paths_rect A (fun (y0 z0 : A) (r0 : y0 ~~> z0) => forall q0 : x ~~> y0,
(p @ q0) @ r0 ~~> p @ (q0 @ r0))  (fun (x0 : A) (q0 : x ~~> x0) =>
 paths_rect A (fun (x1 x2 : A) (q1 : x1 ~~> x2) => forall p0 : w ~~> x1,
(p0 @ q1) @ idpath x2 ~~> p0 @ (q1 @ idpath x2))
 (fun (x1 : A) (p0 : w ~~> x1) =>
 paths_rect A (fun (w0 x2 : A) (p1 : w0 ~~> x2) => (p1 @ idpath x2) @
idpath x2 ~~> p1 @ (idpath x2 @ idpath x2))
 (fun x2 : A => idpath_right_unit A x2 x2 (idpath x2 @ (idpath x2 @ idpath
x2))) w x1 p0) x x0 q0 p) y z r q
 : forall (A : Type) (w x y z : A) (p : w ~~> x) (q : x ~~> y) (r : y ~~>
z), (p @ q) @ r ~~> p @ (q @ r)
```

Гомотопическая теория типов в Coq.

```
map = fun (A B : Type) (x y : A) (f : A -> B) (p : x ∼∼> y) =>
paths_rect A (fun (x0 y0 : A) (_ : x0 ∼∼> y0) => f x0 ∼∼> f y0) (fun x0 :
A => idpath (f x0)) x y p
 : forall (A B : Type) (x y : A) (f : A -> B), x ∼∼> y -> f x ∼∼> f y

transport = fun (A : Type) (P : A -> Type) (x y : A) (p : x ∼∼> y) (X : P
x) =>
paths_rect A (fun (x0 y0 : A) (_ : x0 ∼∼> y0) => P x0 -> P y0) (fun (x0 :
A) (X0 : P x0) => X0) x y p X
 : forall (A : Type) (P : A -> Type) (x y : A), x ∼∼> y -> P x -> P y
```

Гомотопическая теория типов в Coq.

```
Ltac path_tricks :=
first
[apply homotopy_concat
| apply opposite_map
| apply opposite_opposite
| apply opposite_concat
| apply map_cancel
| idtac] ; auto with path_hints.
Ltac path_via x := apply @ concat with (y := x); path_tricks.
```

Гомотопическая теория типов Coq.

```
Definition hfiber {A B}(f : A -> B)(y : B) := x : A f x ~~> y.
Definition contractible A := {x : A /\ forall y : A, y ~~> x}.
Ltac contract_hfiber y p :=
match goal with
| [|- contractible (@hfiber _ _ ?f ?x )] =>
 eexists (existT (fun z => f z ~~> x) y p);
 let z := fresh ''z'' in
 let q := fresh ''q'' in
 intros [z q]
end.
```

Гомотопическая теория типов Coq.

```
Lemma transport_hfiber A B (f : A ->
B)(x y : A)(z : B)(p : x ~~> y)(q :
f x ~~> z) : transport (P := fun x
=> f x ~~> z)p q ~~> !(map f p) @ q.
Proof.
```

```
1 subgoal
A : Type
B : Type
f : A -> B
x, y : A
z : B
p : x ~~> y
q : f x ~~> z
______________________________(1/1)
transport p q ~~> ! map f p @ q
```

Гомотопическая теория типов Coq.

```
Lemma transport_hfiber A B (f : A ->
B)(x y : A)(z : B)(p : x ~~> y)(q :
f x ~~> z) : transport (P := fun x
=> f x ~~> z)p q ~~> !(map f p) @ q.
Proof.
induction p.
```

```
1 subgoal
A : Type
B : Type
f : A -> B
z : B
x : A
q : f x ~~> z
_____________________________(1/1)
transport (idpath x) q ~~> ! map f
(idpath x) @ q
```

Гомотопическая теория типов Coq.

```
Lemma transport_hfiber A B (f : A ->
B)(x y : A)(z : B)(p : x ~~> y)(q :
f x ~~> z) : transport (P := fun x
=> f x ~~> z)p q ~~> !(map f p) @ q.
Proof.
induction p.
path_via q.
```

```
2 subgoals
A : Type
B : Type
f : A -> B
z : B
x : A
q : f x ~~> z
_____________________________(1/2)
transport (idpath x) q ~~> q
_____________________________(2/2)
q ~~> ! map f (idpath x) @ q
```

Гомотопическая теория типов Coq.

```
Lemma transport_hfiber A B (f : A ->
B)(x y : A)(z : B)(p : x ~~> y)(q :
f x ~~> z) : transport (P := fun x
=> f x ~~> z)p q ~~> !(map f p) @ q.
Proof.
induction p.
path_via q.
path_via (!(idpath (f x)) @ q).
```

```
3 subgoals
A : Type
B : Type
f : A -> B
z : B
x : A
q : f x ~~> z
_____________________________(1/3)
transport (idpath x) q ~~> ! idpath
(f x) @ q
_____________________________(2/3)
! idpath (f x) @ q ~~> q
_____________________________(3/3)
q ~~> ! map f (idpath x) @ q
```

Гомотопическая теория типов Coq.

```
Lemma transport_hfiber A B (f : A ->
B)(x y : A)(z : B)(p : x ~~> y)(q :
f x ~~> z) : transport (P := fun x
=> f x ~~> z)p q ~~> !(map f p) @ q.
Proof.
induction p.
path_via q.
path_via (!(idpath (f x)) @ q).
path_via (idpath (f x) @ q).
```

```
5 subgoals
A : Type
B : Type
f : A -> B
z : B
x : A
q : f x ~~> z
_____________________________(1/5)
transport (idpath x) q ~~> idpath (f
x) @ q
_____________________________(2/5)
idpath (f x) ~~> ! idpath (f x)
_____________________________(3/5)
q ~~> q
_____________________________(4/5)
! idpath (f x) @ q ~~> q
_____________________________(5/5)
```

Гомотопическая теория типов Coq.

```
Lemma transport_hfiber A B (f : A ->
B)(x y : A)(z : B)(p : x ~~> y)(q :
f x ~~> z) : transport (P := fun x
=> f x ~~> z)p q ~~> !(map f p) @ q.
Proof.
induction p.
path_via q.
path_via (!(idpath (f x)) @ q).
path_via (idpath (f x) @ q).
apply opposite; auto.
```

```
4 subgoals
A : Type
B : Type
f : A -> B
z : B
x : A
q : f x ~~> z
_____________________________(1/4)
idpath (f x) ~~> ! idpath (f x)
_____________________________(2/4)
q ~~> q
_____________________________(3/4)
! idpath (f x) @ q ~~> q
_____________________________(4/4)
q ~~> ! map f (idpath x) @ q
```

Гомотопическая теория типов Coq.

```
Lemma transport_hfiber A B (f : A ->
B)(x y : A)(z : B)(p : x ~~> y)(q :
f x ~~> z) : transport (P := fun x
=> f x ~~> z)p q ~~> !(map f p) @ q.
Proof.
induction p.
path_via q.
path_via (!(idpath (f x)) @ q).
path_via (idpath (f x) @ q).
apply opposite; auto.
apply idpath.
apply idpath.
```

```
2 subgoals
A : Type
B : Type
f : A -> B
z : B
x : A
q : f x ~~> z
______________________________(1/2)
! idpath (f x) @ q ~~> q
______________________________(1/2)
q ~~> ! map f (idpath x) @ q
```

Гомотопическая теория типов Coq.

```
Lemma transport_hfiber A B (f : A ->
B)(x y : A)(z : B)(p : x ~~> y)(q :
f x ~~> z) : transport (P := fun x
=> f x ~~> z)p q ~~> !(map f p) @ q.
Proof.
induction p.
path_via q.
path_via (!(idpath (f x)) @ q).
path_via (idpath (f x) @ q).
apply opposite; auto.
apply idpath.
apply idpath.
path induction.
```

```
1 subgoal
A : Type
B : Type
f : A -> B
z : B
x : A
q : f x ~~> z
_____________________________(1/1)
q ~~> ! map f (idpath x) @ q
```

Гомотопическая теория типов Coq.

```
Lemma transport_hfiber A B (f : A ->
B)(x y : A)(z : B)(p : x ~~> y)(q :
f x ~~> z) : transport (P := fun x
=> f x ~~> z)p q ~~> !(map f p) @ q.
Proof.
induction p.
path_via q.
path_via (!(idpath (f x)) @ q).
path_via (idpath (f x) @ q).
apply opposite; auto.
apply idpath.
apply idpath.
path induction.
apply opposite.
```

```
1 subgoal
A : Type
B : Type
f : A -> B
z : B
x : A
q : f x ~~> z
_____________________________(1/1)
! map f (idpath x) @ q ~~> q
```

Гомотопическая теория типов Coq.

```
Lemma transport_hfiber A B (f : A -> B)(x y : A)(z : B)(p : x ~~> y)(q : f
x ~~> z) : transport (P := fun x => f x ~~> z)p q ~~> !(map f p) @ q.
Proof.
induction p.
path_via q.
path_via (!(idpath (f x)) @ q).
path_via (idpath (f x) @ q).
apply opposite; auto.
apply idpath.
apply idpath.
path induction.
apply opposite.
apply idmap_map.
Defined.
```

Гомотопическая теория типов. Формализация в Agda.

1. Уровни универсов:

```
postulate
 Level : Set
 Zero : Set
 suc : Level -> Level
 max : Level -> Level -> Level
```

2. Сигма-тип:

```
record Σ { i j } (A : Set i) (B : Set j) where
 constructor _,_
 record
 π1: A
 π2: P(π1).
```

$2\frac{1}{2}$. В частности, прямое произведение типов:

```
_×_ : ∀ { i j } (A : Set i) (B : Set j) → Set : (max i j)
A×B = Σ ( λ _ B )
```

Гомотопическая теория типов. Формализация в Agda.

3. Зависимое произведение:

```
Π : ∀ { i j } (A : Set i) (B : Set j) → Set : (max i j)
Π A B = (x : A) → P x
```

4. Функция тождества:

```
id : ∀ { i } (A : Set i) → ( A → A )
id A = λx → x
```

5. Композиция функций:

```
_∘_ : ∀ { i j k } { A : Set } { B : A → Set j } { C : (a : A) → ( B a → Set k )
} → ( g : { a : A } → Π (B a )(C a)) → (f : Π A B) → Π A (λa → C a (f a))
g ∘ f = λx  → g (f x)
```

Гомотопическая теория типов. Формализация в Agda.

6. Определение путей.

```
data _≡_ {i} {A : Set i} (a : A) : A → Set i where
 refl : a ≡ a
```

7. `! : ∀{i}{A : Set i}{x y : A} → (x ≡ y → y ≡ x)`

```
! refl = refl
```

8. `_∘_ : ∀{i}{A : Set i}{x y z : A} → (x ≡ y → y ≡ z → x ≡ z)`

```
refl ∘ q = q
```

9. `ap : ∀{i j}{A : Set i}{B : Set j}(f : A → B){x y : A} → (x ≡ y → f x ≡ f y)`

```
ap f refl = refl
```

10.

```
transport :∀{i j}{A : Set i}(P : A → Set j){x y : A} → (x ≡ y → P x → P y)
transport P refl t = t
```

Определение группоида в Agda.

```
record Groupoid ∘ p : Set(suc(∘ ⊔ )) where
 infix 8 _⁻¹
 infixr 7 _∘_
 infix 4 _≡_
 field
 Object : Set∘
 _≡_: Object → Object  →  Set p
 id : ∀{x} → x ≡ x
_⁻¹ : ∀{x y} → x ≡ y → y ≡ x
_∘_ :  ∀{x y z} → x ≡ y → y ≡ z → x ≡ z
left-id : ∀{ x y }(p : x ≡ y) → id ∘ p ≡ p
right-id : ∀{ x y ( p : x ≡ y) → p ∘ id ≡ p
assoc : ∀{ f x y z }(p : x ≡ y)(q: y ≡ z)(r: x ≡ z) → (p ∘ q) ∘ r ≡ p ∘ (q ∘ r)
left-inv : ∀{x y}(p : x ≡ y) → p⁻¹ ∘ p ≡ id
right-int : ∀{x y}(p : x ≡ y) → p ∘ p⁻¹ ≡ id
```

$\Omega S^1 \simeq \mathbb{Z}$ в Agda.

```
S¹ : Set
base : S¹
loop : base == base

S¹-induction : (X : S¹ → Type)
base' : X base
loop' : Path (transport X loop base') base'
→ (y : S¹) → X y

S¹Cover : S¹ → Type₀
S¹Cover = S¹Cover.f

encode : {x : S¹} (p : base == x) → S¹Cover x
encode p = transport S¹Cover p 0
```

$\Omega S^1 \cong \mathbb{Z}$ в Agda.

```
loop^ : (n : ℤ) → base = base
loop^ 0 = idp
loop^ (pos 0) = loop
loop^ (pos (S n)) = loop^ (pos n) · loop
loop^ (neq 0) = ! loop
loop^ (neg (S n)) = loop^ (neg n) · ! loop
```

$\Omega S^1 \cong \mathbb{Z}$ в Agda.

```
loop ^ succ (n : ℤ) → loop ^ n · loop = loop ^ (succ n)
loop ^ succ 0 = loop
loop ^ succ (pos n) = loop · loop ^ n
loop ^ succ (neg 0) = !-inv-l loop
loop ^ succ (neg (S n)) =
 (loop ^ (neg n) · ! loop) · loop
 =⟨ ·-assoc (loop ^ (neg n)) (! loop) loop ⟩
 loop ^ (neg n) · (! loop · loop)
 =⟨ !-inv-l loop |in-ctx (λ u → loop ^ (neg n) · u) ⟩
 loop ^ (neg n) · idp
 =⟨ ·-unit-r _ ⟩
 loop ^ (neg n)
```

$\Omega S^1 \simeq \mathbb{Z}$ в Agda.

```
encode-loop ^ : (n : ℤ) → encode (loop ^ n) = n
encode-loop ^ 0 = idp
encode-loop ^ (pos 0) = S¹Cover.coe-loop-β 0
encode-loop ^ (pos (S n)) =
 encode (loop ^ (pos n) · loop) = ⟨ idp ⟩
 coe (ap S¹Cover (loop ^ (pos n) · loop)) 0
 =⟨ ap-· S¹Cover (loop ^ (pos n)) loop |in-ctx (λ u → coe u 0) ⟩
 coe (ap S¹Cover (loop ^ (pos n)) · ap S¹Cover loop) 0
 =⟨ coe-· (ap S¹Cover (loop ^ (pos n)))
 (ap S¹Cover loop) 0 ⟩
 coe (ap S¹Cover loop) (coe (ap S¹Cover (loop ^ (pos n))) 0)
 = ⟨ encode-loop ^ (pos n) |in-ctx coe (ap S¹Cover loop) ⟩
 coe (ap S¹Cover loop) (pos n)
 =⟨ S¹Cover.coe-loop-β (pos n) ⟩
 pos (S n)
```

$\Omega S^1 \cong \mathbb{Z}$ в Agda.

```
decode : (x : S¹) → (S¹Cover x → base == x)
decode {x}=
S¹-induction
(λ x → S¹Cover x → base == x)
loop^
transport (λ x → S¹Cover x → base == x) loop loop ^
≃ transport (λ x → base == x) loop o loop ^ o transport S¹Cover (! loop)
≃ (λ p → loop ∘ p) o loop ^ transport S¹Cover (! loop)
≃ (λ p → loop ∘ p) o loop ^ o pred
≃ (λ n → loop ∘ (loop ^ (pred n)))
≃ (λ n → loop ^ n)

decode-encode : (x : S¹) (p : base == x) → decode x (encode p) == p
decode-encode {x} α = path-induction (λ x : S¹ → λ α : base == x → decode
x (encode p) == id p
ΩS1 ≅ ℤ : (base == base) ≅ ℤ
```

Спасибо за внимание!