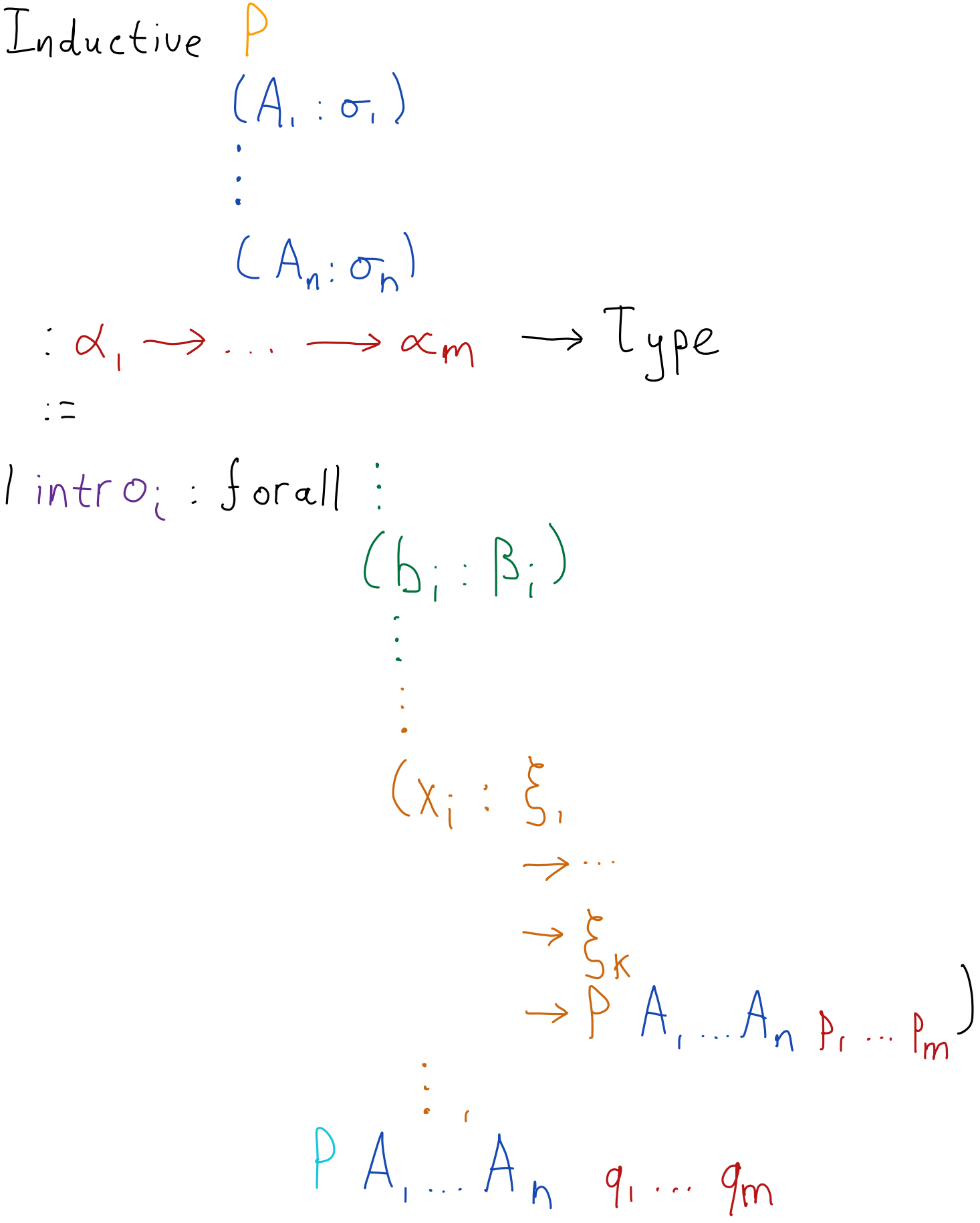
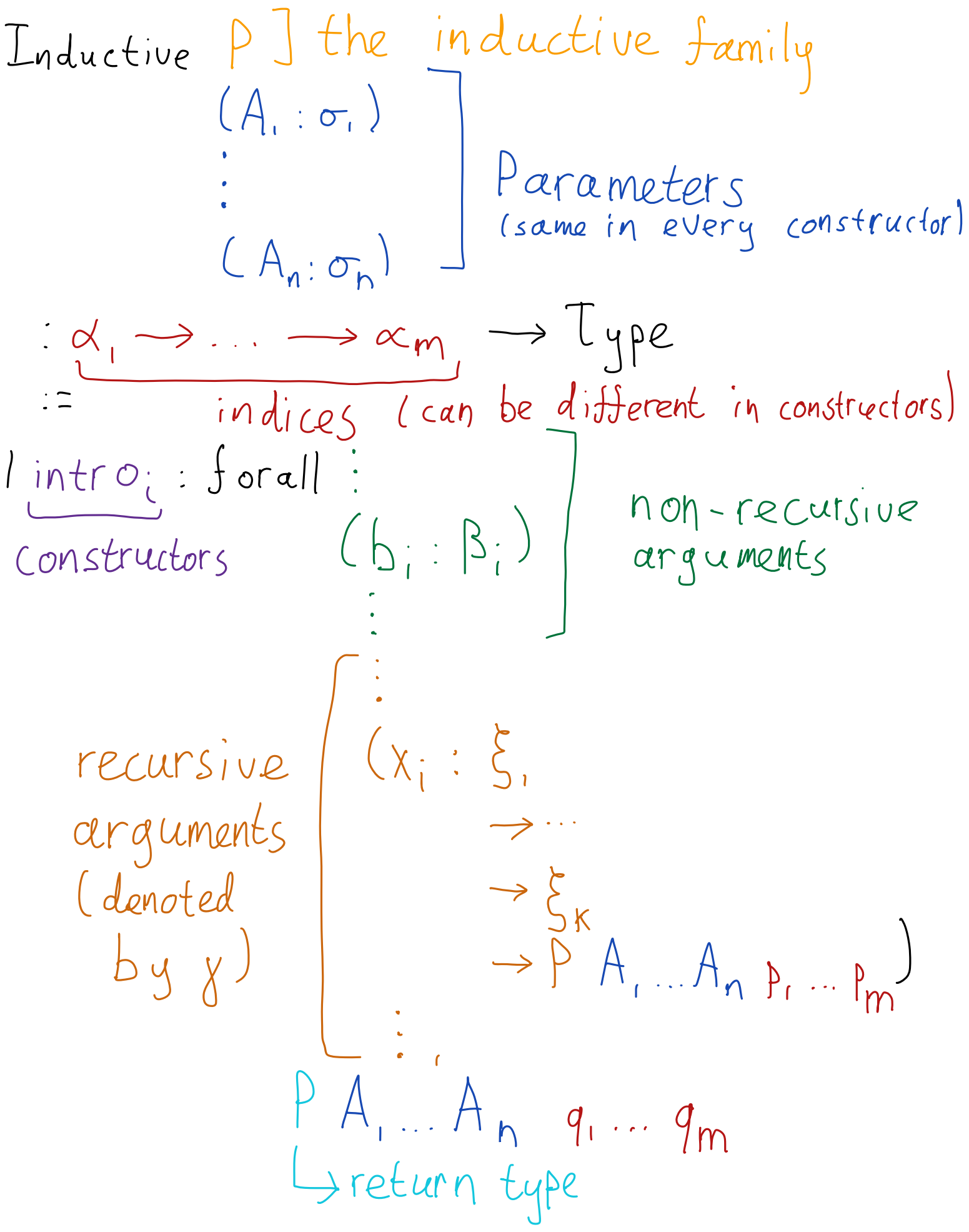

Inductive P
(A1 : σ1)
(An : σn)
: α1 → ... → αm → Type
:=
| introi : forall
(bi : βi)
(xi : ξ1
→ ...
→ ξk
→ P A1 ... An p1 ... pm)
P A1 ... An q1 ... qm


Inductive P] the inductive family
(A1 : σ1)
(An : σn)
Parameters
(same in every constructor)
: α1 → ... → αm → Type
:=
indices (can be different in constructors)
| introi : forall
constructors
(bi : βi)
non-recursive arguments
recursive arguments (denoted by y)
(xi : ξ1
→ ...
→ ξK
→ P A1 ... An p1 ... pm)
P A1 ... An q1 ... qm
return type

```
Inductive nat
  : Type

  :=
| Z : nat

| S :  nat

      → nat
```

A : empty list

Inductive nat

: Type

α : empty list

:=

| Z : nat { β : empty list, γ : empty list }

| S : nat { β : empty list, γ : single element, ξ : empty list }

→ nat

```
Inductive list
          (A: Type)
  : Type

  :=
| nil: list A

| cons: forall (a: A)
              (x: list A),
        list A
```

Inductive list (A: Type) / A is a singleton list

: Type

α : empty list

:=

| nil : list A { β : empty list
γ : empty list

| cons : forall (a: A) { β : singleton list of A
(x: list), { γ : single element
ξ : empty list

list A

```
Inductive Vec
          (A: Type)
  : nat → Type
  :=
| nil : Vec A 0

| cons : forall (b :A)
               (n: nat)
               (x: Vec A n),
         Vec A (S n)
```

Inductive Vec (A: Type) — A: singleton list

: nat → Type

:= α : singleton list

| nil : Vec A 0 { β : empty list; no recursive arguments }

| cons : forall (b : A) (n : nat) { β : list of A and nat } (x : Vec A n), { γ has one element; ξ : empty list }

Vec A (S n)

```
Inductive Tree
  : Type

  :=
| leaf : Tree
| node : forall (X : nat → Tree),
           Tree
```

A empty list

Inductive Tree

: Type

a empty list

:=

| leaf : Tree

| node : forall (X : nat → Tree), { B : empty ; γ : single element ; ξ : nat }

Tree