Coq Meets Subtyping


```
Definition Eq_trans {A : Type} {a a' a'' : A} (e : Eq a a') (e' : Eq a' a'') : Eq a a''
 := match e in Eq _ a' return Eq a' a'' -> Eq a a'' with
      | Eq_refl => fun e' => e'
    end e'.
```



```
Fixpoint length_Vec2List {A : Type} {n : Nat} (v : Vec A n) : Eq n (length (Vec2List v))
 := match v as v in Vec _ n return Eq n (length (Vec2List v)) with
      | empty => Eq_refl 0
      | prep _ n v => pres_Eq S (length_Vec2List v)
    end.
```



```
Definition empty_Eq (A : Type) (v : Vec A 0) : Eq(empty A) v
 := match v as v in Vec _ n return match n as n return Vec A n -> Type with
                                     | 0 => fun v => Eq (empty A) v
                                     | _ => fun _ => unit
                                   end v with
      | empty => Eq_refl (empty A)
      | _ => tt
    end.
```



```
Definition if_nat (t : Typ) (T : Type) : Type
  := match t with
       | nat => T
       | _ => Unit
     end.
```



```
Fixpoint sub_trans {t t' t'' : Typ} (s : sub t t') (s' : sub t' t'') : sub t t''
 := match s in sub t t' return sub t' t'' -> sub t t'' with
      | sub_bool => fun s' => s'
      | sub_nat => fun s' => s'
      | sub_bn => fun s' => match s' in sub t' t'' return if_nat t' (sub bool t'') with
                              | sub_nat => sub_bn
                              | _ => tt
                            end
    end s'.
```