

Coq Meets Subtyping



```

Definition Eq_trans (A : Type) (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'.
    
```

so import context in return chose

Context's type needs to change depending on match

use context with changed type

supply context

```

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 a n v => pres_Eq S (length_Vec2List v)
end.
    
```

n changes to S n

changes to prep a n v where v has type Vec A n

length (Vec2List (prep a n v)) simplifies to S (length (Vec2List v))

we need something of type Eq (S n) (S (length (Vec2List v)))

so Vec A n → Eq n (length (Vec2List v))

NO. Eq S

```

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.
    
```

1. unless it so that you only have to provide something useful when n is 0

2. convert v to appropriate type

3. has well typed type Vec A 0

4. has type Vec A n

5. n is S_

6. n is 0 and v is empty A

7. n is S_ so evaluates to Unit

so contributes to Eq (empty A) (empty A)

```

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

you only need to provide a T when t is nat

otherwise you can just provide Unit

```

Fixpoint sub_trans (t t' t'' : Type) (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'.
    
```

change t and t' in each case as appropriate

convert s' in the context to the appropriate type

return to be not into you only need to handle nat cases

otherwise just return a unit