Unconventional

An Introduction To Coq

Recursive Data Type

Defining a new type

```
Inductive Nat : Type
 := O : Nat
  | S : Nat -> Nat.
```

O is a constant

with two constructors

S takes an argument

fixed

dependent

```
Inductive Le (n : Nat) : Nat -> Type
 := Le_refl : Le n n
  | Le_S (n' : Nat) : Le n n' -> Le n (S n').
```

$n \le n' \Rightarrow n \le n'+1$

Le 5 8

Le 5 7

Le 5 6

Le 5 5

: Le 5 5

Le_refl 5

arbitrary expression

variable

translating type

```
Fixpoint Le_O (n : Nat) : Le O n
 := match n as n return Le O n with
      | O => Le_refl O
      | S n => Le_S O n (Le_O n)
    end.
```

Le O n

Le O (S n)

n : nat

Le O n

S

S

O

Le_O ⇒

Le 0 2

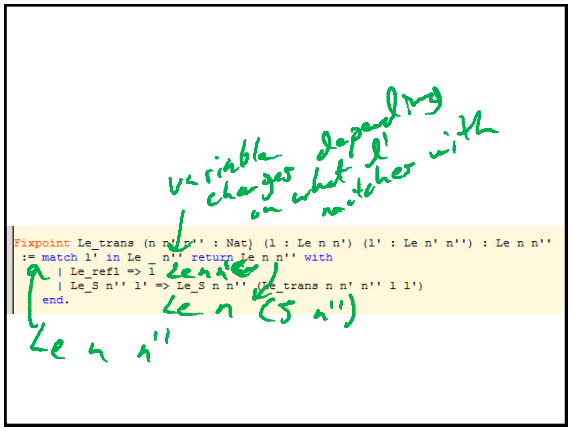
Le_S

Le 0 1

Le_S

Le 0 0

Le_refl


Fixpoint Le_trans (n n' n'' : Nat) (l : Le n n') (l' : Le n' n'') : Le n n''
:= match l' in Le _ n'' return Le n n'' with
| Le_refl => l
| Le_S n'' l' => Le_S n n'' (Le_trans n n' n'' l l')
end.
variable changes depending on what it matches with
Le n (S n'')
Le n n''