Using the "propositions as types" correspondence, what proposition does this program prove?

```plaintext
let rec loop x = loop x
```

A. P  
B. P ⇒ P  
C. P = P  
D. P ⇒ Q  
E. (P ⇒ Q) ⇒ R
Review

Previously in 3110:
• Functional programming in Coq
• Logic in Coq
• Proofs are programs (Curry-Howard, BHK)

Today:
• Induction in Coq
INDUCTION ON NATURAL NUMBERS
Structure of inductive proof

Theorem: for all natural numbers \( n \), \( P(n) \).

Proof: by induction on \( n \)

Case: \( n = 0 \)
Show: \( P(0) \)

Case: \( n = k+1 \)
IH: \( P(k) \)
Show: \( P(k+1) \)

QED
Sum to $n$

```plaintext
let rec sum_to n =
    if n=0 then 0
    else n + sum_to (n-1)
```

Theorem:
for all natural numbers $n$,

\[ \sum_{i=0}^{n} i = n \times (n+1) / 2. \]

Proof: by induction on $n$

Discussion:  What is $P$?  Base case?  Inductive case?  Inductive hypothesis?
Proof

\[ P(n) \equiv (\text{sum}_n = n \cdot (n+1) / 2) \]

Case: \( n = 0 \)
Show:
\[ P(0) \]

Case: \( n = k+1 \)
IH: \( P(k) \equiv \text{sum}_k = k \cdot (k+1) / 2 \)
Show:
\[ P(k+1) \]

QED

let rec sum_to n =
  if n=0 then 0
  else n + sum_to (n-1)
INDUCTION ON LISTS
Structure of inductive proof

Theorem: for all natural numbers n, \( P(n) \).

Proof: by induction on n

Case: \( n = 0 \)
Show: \( P(0) \)

Case: \( n = k+1 \)
IH: \( P(k) \)
Show: \( P(k+1) \)

QED
Structure of inductive proof

Theorem:  
for all lists $\text{lst}$, $P(\text{lst})$.

Proof: by induction on $\text{lst}$

Case: $\text{lst} = []$
Show: $P(\text{[]} )$

Case: $\text{lst} = \text{h::t}$
IH: $P(\text{t} )$
Show: $P(\text{h::t} )$

QED
Append nil

let rec (@) lst1 lst2 =
  match lst1 with
  | []   -> lst2
  | h::t -> h :: (t @ lst2)

Theorem:
for all lists lst, lst @ [] = lst.

Proof: by induction on lst

Discussion: What is P?  Base case? Inductive case?  Inductive hypothesis?
**Base case**

\[ P(lst) \equiv lst @ [] = lst \]

**Case**: \( lst = [] \)

**Show**: 
\[ P([]) \]

**Case**: \( lst = h::t \)

**IH**: \( P(t) \equiv t @ [] = t \)

**Show**: 
\[ P(h::t) \]

**QED**

```haskell
let rec (@) lst1 lst2 =
  match lst1 with
  | [] -> lst2
  | h::t -> h :: (t @ lst2)
```
INDUCTION ON LISTS IN COQ
INDUCTION ON NATS
Inductive types

induction works on inductive types, e.g.

Inductive list (A : Type) : Type :=
  | nil : list A
  | cons : A -> list A -> list A

Need an inductive definition of natural numbers...
Naturals

Inductive nat : Set :=
  \mid O : nat (* zero *)
  \mid S : nat \to nat (* succ *)

type nat = O \| S of nat

0 is O
1 is S O
2 is S (S O)
3 is S (S (S O))

- unary representation
- Peano arithmetic
Induction on nat(ural)s

Theorem:
for all n:nat, P(n)

Proof: by induction on n

Case: n = 0
Show: P(0)

Case: n = S k
IH: P(k)
Show: P(S k)

QED

Theorem:
for all naturals n, P(n)

Proof: by induction on n

Case: n = 0
Show: P(0)

Case: n = k+1
IH: P(k)
Show: P(k+1)

QED
Goal: redo this proof in Coq

```coq
let rec sum_to n =
  if n=0 then 0
  else n + sum_to (n-1)
```

Theorem:
for all natural numbers n,

\[ \sum_{i=0}^{n} i \]

Proof: by induction on n

Demo
CONTROLLED RECURSION
Why no infinite loops?

In OCaml:

```ocaml
# let rec loop x = loop x
val loop : 'a -> 'b = <fun>
```

By propositions-as-types, these are the same:

- 'a → 'b
- A ⇒ B

What if A=True, B=False?

Infinite loops prove False!
CONTROLLED RECURSION
Induction and recursion

• Intense similarity between inductive proofs and recursive functions on variants
  – In proofs: one case per constructor
  – In functions: one pattern-matching branch per constructor
  – In proofs: uses IH on "smaller" value
  – In functions: uses recursive call on "smaller" value

• Proofs = programs
• Inductive proofs = recursive programs
Upcoming events

• A10 GIST: tonight, 8 pm, Gates 122

This is inductive.

THIS IS 3110