1. Mostrar que sem usar a táctica `auto` ou equivalente.
   (a) `forall A B:Prop, (A /\ B) -> (A \/ B).`
   (b) `forall P Q R: Prop, P->Q-> R-> P/\Q/\R.`
   (c) `forall A B C:Prop, ((A /\ B) -> C) -> (A  -> (B -> C)).`
   (d) `forall A B:Prop, (A -> B) -> ~B -> ~A`
   (e) `forall A B C:Prop, A \/ (B \/ C) ->  (A \/ B) \/ C`
   (f) `forall P:Prop, ~~~ P -> ~P`
   (g) `forall P:Prop, ~~ (P \/ ~P)`

2. Mostra que
   (a) `forall A:Set, forall P Q: A-> Prop, (forall x, P x) \/ (forall x, Q x) -> forall x, P x \/ Q x`
   (b) `~ (forall A:Set, forall P Q: A -> Prop, (forall x, P x \/ Q x) -> (forall x, P x) \/ (forall y, P y))`

3. Mostra que
   (a) `forall (A:Type)(P Q:A->Prop), (exists x:A, P x \/ Q x) -> (ex P)\/ (ex Q)`
   (b) `forall (A:Type)(P Q:A->Prop),(ex P) \/ (ex Q) -> exists x: A,  P x \/ Q x`

4. Considerando a secção seguinte demonstra o teorema.

```
Section Ref.
Variable D: Set.
Variable R: D-> D-> Prop.
Hypothesis Anti_sym : forall x y: D, R x y -> ~ R y x.

Theorem not_ref:forall x:D,  ~ R x x.
End Ref.
```

5. Considerando o tipo indutivo

```
Inductive bin : Set :=
L : bin | N: bin-> bin->bin.
```

e as funcões:

```
Fixpoint size (t:bin):nat :=
match t with
  L => 1
| N t1 t2 => 1 + size t1 + size t2
```

```
end.

Definition add_one (t:bin): bin :=
match t with
  L => N L L
  | N t1 t2 => N L (N t1 t2)
end.
```

Mostra que

```
forall t:bin, size(add_one(t))= S (S (size t)).
```

6. Considerando o tipo indutivo `list A` definido na biblioteca `Lists` do COQ e considerando as funções:

```
Fixpoint length (A:Set) (l: list A){struct l}: nat :=
match l with
nil => 0 |
cons a l' => 1 + (length A l')
end.

Definition add_one (A:Set) (l:list A) (a: A):list A:= a::l.
```

Mostra que

```
 forall (A:Set) (l:list A) (a:A), (length A (add_one A l a)) =
  (length A l ) +1.
```