Proof Pattern 1: In implication formula($(A_1 \wedge A_2) \rightarrow C$) proofs by Induction, here is the recipe:
1. Choose an induction scheme
2. Apply the induction scheme to the formula to obtain proof obligations.
3. Simplify the base case and induction step proof obligations, by using the tautology($p \rightarrow ((a_1 \wedge a_2) \rightarrow c) \equiv p \wedge a_1 \wedge a_2 \rightarrow c$). So basically push A_1 and A_2 into the top-level implies like this:

```
(implies (and [Base case branch condition]
              A1
              A2)
         C)

(implies (and [Induction step branch condition]
              (implies (and A1' A2')
                       C')
              A1
              A2)
         C)
```

4. After simplifying the proofs, notice that in the induction step context, you want to use C', but you cant use it directly. You have to unlock it, by proving A_1', A_2'. This is easy, just use the rest of the context, that is, use A_1, A_2, *IScond*, to derive more context(D_1, D_2 etc) and in this way deduce A_1', A_2', so that they become part of your derived context , which in turn, deduces C'(using Modus Ponens Rule) and hence you can use it directly in your proof.

Proof Pattern 2: Which induction scheme to choose is a heuristic, and comes with practice. The heuristic to use in case of proving formulas which contain a (tail-recursive) accumulator style function, is to choose that **same** accumulator-style function to induct on, since it will help the most with its Induction Hypothesis in the course of opening up its definition. In general, a good heuristic to choose the induction scheme to use, is to choose the one you think is the most important function in your conjecture.

As an example of proof pattern 2, lets say you want to prove:

```
(equal (f*-acc x y acc)  (g x (f x y) acc))
```

Then you prove it by doing induction on `f*-acc`, i.e, you use the induction scheme that `(f*-acc x y acc)` gives rise to. If you try to use the induction scheme generated by `(f x y)`, you will get stuck.