Next big topic: <u>Recursion</u>

Analogy: much like <u>proofs by induction</u>.

Normal proof:

Thm: $A \Rightarrow E$.

Proof: $A \Rightarrow B$ (Axiom 2)

$B \Rightarrow C$ (SAS Theorem)

$C \Rightarrow D$ (algebra ..)

$D \Rightarrow E$ (. —)

✓

Unlike the above meta thing, proofs by induction can involve an <u>unbounded</u> # of implications. Here's how it works.

Usual setting: want to prove that some property T of positive integers (1, 2, 3 ...) holds <u>for all</u> such values (1, 2, 3 ...)

Example: $\displaystyle\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. ($\leftarrow$ this eqn is true for n $\equiv T(n)$)

Outline of inductive proof:

① Show $T(1)$ explicitly.

$$\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2} \quad ✓$$

(2) Now prove a <u>parameterized</u> implication of the form $T(n-1) \Rightarrow T(n)$

(or maybe $\bigwedge_{i=1}^{n-1} T(i) \Rightarrow T(n)$)

(3) Combine (1) & (2) to get a proof for <u>any</u> $n \in \mathbb{Z}^+$ $(= \{1,2,3..\})$

$$\underset{(1)}{T(1)} \overset{(2)}{\Longrightarrow} T(2) \overset{(2)}{\Longrightarrow} T(3) \overset{(2)}{\Longrightarrow} \cdots \overset{(2)}{\Longrightarrow} T(n)$$

✓

What does this have to do w/ programming?

We can write programs w/ the same structure.

Here's an outline:

$T(n) \equiv$ "my program does the right thing on any input of size n".

Then we can write a <u>self-referencing</u> program like this:

```
int f(int x) {
  if (size(x) == 1)    //    n == 1
     // explicitly solve, or
     // hardcode answer.
  else
     // build answer to x from
     // f(x') where size(x') ≨ size(x)
```

// so, we use f itself as a subroutine!

$\equiv$ Step 2 of an inductive proof

$\equiv$ Step 1 of an inductive proof.