The Induction Principle

To prove that a statement S(n) is true for every natural number n, it suffices to:

1. *Base Case*: Prove that the statement S(0) is true;
2. *Induction Step*: Assuming S(n) is true, prove that S(n+1) is true.

When proving the induction step, the assumption S(n) is called the *induction hypothesis*.

Often we need to prove that a statement S(n) is true not exactly for every n, but for every n starting from a given number k. The base case is then S(k); the induction step is the same.

Example 1

Problem. Prove that the sum of first n odd numbers is equal to n^2.

Proof. The statement S(n) is $1 + 3 + \ldots + (2n-1) = n^2$, I.e. ($\sum_{i=1,n}(2i-1) = n^2$) and we want to prove it is true for every $n \geq 1$.

Base Case. S(1) is the statement 1=1.

Induction Step. Assume the induction hypothesis

$$1 + 3 + \ldots + (2n-1) = n^2$$

The goal is to prove

$$1 + 3 + \ldots + (2n-1)+(2n+1) = (n+1)^2$$

Using the IH, the goal can be rewritten as

$$n^2 + (2n+1) = (n+1)^2,$$

which is directly verified.

qed

Complete (Strong) Induction

To prove that a statement S(n) is true for every natural number n, it suffices to:

1. *Base Case*: Prove that the statement S(0) is true.
2. *Induction Step*: Assuming n>0 and that S(k) is true for all numbers k smaller than n, prove that S(n) is true.

Example 2

Problem. Let f : N $\rightarrow \mathrm{N}$ be defined recursively by

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 2f\left(\dfrac{n}{2}\right) & \text{if } n \text{ is even} \\ f(n-1)+1 & \text{if } n \text{ is odd} \end{cases}$$

Prove that f(n)=n for every n.

Proof.

Base Case. f(0)=0; true by definition of f.

Induction Step. Suppose n>0 and f(k)=k for all k<n. To derive f(n)=n, we consider separately the cases when n is even and odd.

- If n is even, we have f(n/2) = (n/2) by IH (note (n/2)<n). Therefore, f(n) = 2f(n/2) = 2 $*$ (n/2) = n.
- If n is odd, the IH gives us f(n-1)=n-1, so we get

  f(n) = f(n-1)+1=(n-1)+1=n.

qed

Example 3

Problem. Suppose two strings u and v satisfy the relation uv=vu. Prove that u and v are powers of the same string.

Proof. Induction on |u|+|v| Strictly speaking, the statement S(n) is this: If uv=vu and |u|+|v|=n then u and v are powers of the same string.

B*ase Case*. |u|+|v|=0. This implies u=v=ε, and the statement is true.

Induction Step. We're arguing by complete induction. Suppose |u|+|v|= n and n>0 and suppose that the statement is true for every u',v' such that |u'|+|v'|<n.

Proof continued

If $|u|=|v|$, the statement is true. Assume $|u|<|v|$. (The third case $|u|>|v|$ is symmetric and does not need to be considered separately.)

Then $v=uw$ for some w and we have $uuw=uwu$. This implies $uw=wu$. Since $|w|<|v|$, we have $|u|+|w|<|u|+|v|$ and the IH applies giving us that u and w are powers of the same string z.

Clearly then, $v=uw$ is also a power of z.

qed

Structural Induction

A method for proving properties of objects (trees, expressions, etc.) defined recursively. Such recursive definitions have a number of base cases defining the simplest objects and a number of rules telling how a bigger object is build from smaller ones.

To prove that a statement S(x) is true for every object it suffices to prove:

Base Case: S(x) is true for the basic objects.

Induction Step: For every rule telling us how to build a bigger object x from smaller objects x_1, ... x_k, prove that S(x) is true, assuming as the IH that $S(x_1)$, ... , $S(x_k)$ are true.

Structural induction is induction on the size of the object.

Example: Balanced Parentheses

Parenthesis expressions (pexps) are defined recursively by the following rules:

[1.] The empty string ε is a pexp.
[2.] If w is a pexp, then (w) is a pexp.
[3.] If u and v are pexps, then uv is a pexp.

Note: pexps define a language over the alphabet $\Sigma = \{ \, (\, , \,) \, \}$.

Problem 1. Every pexp has equal number of left and right parentheses.

Pexp proof

Problem 1. Every pexp has equal number of left and right parentheses.

For a string w over the alphabet $\Sigma=\{(,)\}$, let E(w) denote the property "w has equal number of left and right parentheses".

Proof.

1. True for ε.
2. Assume w has the same number of left and right parentheses (E(w)). Then the same is true of (w) (E((w))).
3. Assume u and v both have equal number of left and right parentheses. Then the same holds for uv. (E(u) and E(v) $\Rightarrow$ E(uv))

qed

Problem 2

Problem 2. If w is a pexp, then every prefix of w has at least as many left as right parentheses.

Proof. Let S(w) stand for "every prefix of w has at least as many left as right parentheses".

1. S(ε) is true.
2. If S(w) is true, then S((w)) is true.
3. If S(u) and S(v) are true, then S(uv) is true.

qed

Problem 3

Problem. If a string w satisfies both S(w) and E(w) then w is a pexp.

Proof. Complete induction on |w|.

Base case. |w|=0 is OK because then we have w=ε, and ε is a pexp.

Induction step. Assume that w satisfies S(w) and E(w), that |w|>0, and (the IH) that all strings u shorter than w and satisfying S(u) and E(u) are pexps.

There are two possibilities for w:

(1) all its prefixes except ε and w itself have strictly greater number of ('s than)'s; (2) there exist a prefix u of w such that u$\neq \varepsilon$, u$\neq$ w, and u has equal number of ('s and)'s.

Case analysis

Case (1). w must be of the form w=(u) for some u. Clearly, E(u) is true. But S(u) must be true as well (why?). The IH implies that u is a pexp. Then, referring to the second rule for building pexps, we can conclude that w is a pexp.

Case (2). We can write w=uv. It follows that both u and v satisfy the properties E and S (why?). Since both u and v are shorter than w, the IH applies to them, so u and v are pexps. The third rule for building pexps implies finally that w is a pexp.

qed

Problem 4

There are two ways to form lists

[]	The empty list
(x : xs)	The list with at least 1 element x (called the head), and the rest of the list, xs, (called the tail).

In any implementation, the following “laws” must hold

head(x : xs) = x

tail(x : xs) = xs

Laws about append

Any implementation of the append function must also satisfy the following laws:

Law1: app([],ys) = ys

Law2: app((x : xs),ys) = (x : app(xs,ys))

Using these laws, and proof by structural induction (remember there are only 2 ways to form a list) prove:

app(x,app(y,z)) = app(app(x,y),z)

Structural Induction on lists

To prove P(x)

1) Base case: Prove P([])
2) Inductive step:
   Assume P(xs) then Prove P(x : xs)

For our example P(x) = app(x,app(y,z)) = app(app(x,y),z)
do induction on x (one might try y and z but it won't work out)

1) Base case: Prove
   app([],app(y,z)) = app(app([],y),z)

2) Induction step

Assume:
   app(xs,app(y,z)) = app(app(xs,y),z)

Prove:
   app((x : xs),app(y,z)) = app(app((x : xs),y),z)

Base Case

app([],app(y,z)) = app(app([],y),z)

app(y,z) = app(y,z)

By two applications of

Law1: app([],ys) = ys

Induction Step

Assume:

app(xs,app(y,z)) = app(app(xs,y),z)

Prove:

app((x : xs),app(y,z)) = app(app((x : xs),y),z)

By Law2:

(x : app(xs,app(y,z))) = app(app((x : xs),y),z)

By I.H.

(x : app(app(xs,y),z)) = app(app((x : xs),y),z)

By Law2 (applied right to left)

app((x : app(xs,y)),z) = app(app((x : xs),y),z)

By Law2 (applied right to left, again)

app(app((x : xs),y),z) = app(app((x : xs),y),z)

Law2: app((x : xs),ys) = (x : app(xs,ys))