Mathematical Induction

If we have a propositional function P(n), and we want to prove that P(n) is true for any natural number n, we do the following:

- Show that P(0) is true. (basis step)
  We could also start at any other integer m, in which case we prove it for all $n \geq m$.
- Show that if P(n) then P(n+1) for any $n \in N$.
  (inductive step)
- Then P(n) must be true for any $n \in N$.
  (conclusion)

Induction

Example:

Show that $n < 2^n$ for all positive integers n.

Let P(n) be the proposition "$n < 2^n$."

1. Show that P(1) is true.
(basis step)

P(1) is true, because $1 < 2^1 = 2$.

Induction

2. Show that if P(n) is true, then P(n + 1) is true.
(inductive step)
3. Assume that $n < 2^n$ is true.

We need to show that P(n + 1) is true, i.e.

$n + 1 < 2^{n+1}$

We start from $n < 2^n$:

$n + 1 < 2^n + 1 \leq 2^n + 2^n = 2^{n+1}$

Therefore, if $n < 2^n$ then $n + 1 < 2^{n+1}$

Induction

3. Then P(n) must be true for any positive integer.
(conclusion)

$n < 2^n$ is true for any positive integer.

End of proof.

Induction

Another Example ("Gauss"):

$1 + 2 + \ldots + n = n (n + 1)/2$

1. Show that P(1) is true.
(basis step)
For n = 1 we get 1 = 1. True.

Induction

2. Show that if P(n) then P(n + 1) for any $n \geq 1$. (inductive step)

$1 + 2 + \ldots + n = n (n + 1)/2$

$1 + 2 + \ldots + n + (n + 1) = (n + 1) + n (n + 1)/2$

$= (n + 1)(1 + n/2)$

$= (n + 1)((2 + n)/2)$

$= (n + 1) ((n + 1) + 1)/2$

Induction

3. Then P(n) must be true for any $n \geq 1$. (conclusion)

   $1 + 2 + \ldots + n = n(n + 1)/2$ is true for all $n \geq 1$

   End of proof.

   Note that we've already seen a proof for this not using induction. Often more than one proof is possible.

Induction

A variation on induction is **the second principle of mathematical induction** or strong induction.

It is also used to prove that a propositional function P(n) is true for any natural number n.

It's easy to check that strong induction follows from regular mathematical induction.

Induction

The second principle of mathematical induction:

- Show that P(0) is true.
  (basis step)
  Show that if P(0) and P(1) and P(2)… and P(n),
  then P(n + 1) for any $n \in N$.
  (inductive step)
- Then P(n) must be true for any $n \in N$.
  (conclusion)

Induction

Example:

Show that every integer greater than 1 can be written as the product of primes.

- Show that P(2) is true.
  (basis step)

2 is the product of one prime: itself.

Induction

- Show that if P(2) and P(3) and … and P(n), then P(n + 1) for any $n \in N$. (inductive step)

Two possible cases:
If (n + 1) is **prime**, then obviously P(n + 1) is true.
If (n + 1) is **composite**, it can be written as the product of two integers a and b such that $2 \leq a \leq b < n + 1$.
By the **induction hypothesis**, both a and b can be written as the product of primes.
Therefore, $n + 1 = a \cdot b$ can be written as the product of primes.

Induction

- Then P(n) must be true for any $n \in N$.
  (conclusion)

End of proof.

We have shown that **every integer greater than 1** can be written as the product of primes.

This proves the Fundamental Theorem of Arithmetic (p. 258) (p. 211 6th ed), except for the uniqueness part.

Well Ordering

The Well Ordering Property of the Natural Numbers is:

Every non-empty set of natural numbers has a least element.

This is an axiom of the natural numbers, and is equivalent to mathematical induction.

Well Ordering

Theorem: the following are equivalent

1. Mathematical Induction is valid.
2. Strong induction is valid
3. Every non empty set of natural numbers has a least element

Proof: 1 → 2. Suppose regular induction is valid, and suppose

(i) P(0) is true and

(ii) $P(0) \wedge P(1) \wedge \ldots \wedge P(n-1) \rightarrow P(n)$ for $n \geq 1$.

We need to prove P(n) for all n.

Let Q(n) be "P(0), P(1), …, P(n−1) are true".

Then Q(0) is true and by (ii), $Q(n-1) \rightarrow Q(n)$.

Thus by regular induction Q(n), and hence P(n), is true for all n.

2 → 3.

Suppose strong induction is valid.

Let A be a set of natural numbers without a least element.

Let P(j) be "j is not in A".

Then P(0) is true, and if P(0), P(1),…, P(n−1) are true then also P(n) is true, or n would be the least element of A. But then by strong induction P(n) is true for all n in A, and hence A is empty.

Thus any non empty subset of N has a least element.

3 → 1.

Suppose well ordering is valid and P(j) is a property such that

(i) P(0) is true

(ii) $P(n-1) \rightarrow P(n)$ for all $n > 0$.

Let $A = \{x \in N \mid P(x) \text{ is false}\}$

We need to show A is empty.

If A is not empty it has a least element u.

But u is not 0 by (i).

And if u > 0 then P(u−1) is true and P(u) is false, contradicting (ii).

Hence P(n) is true for all $n \in N$.

Recursive Definitions

Recursion is a principle closely related to mathematical induction.

In a **recursive definition**, an object is defined in terms of itself.

We can recursively define **sequences**, **functions** and **sets**.

Recursively Defined Sequences

Example:

The sequence $\{a_n\}$ of powers of 2 is given by $a_n = 2^n$ for n = 0, 1, 2, … .

The same sequence can also be defined **recursively**:

$a_0 = 1$

$a_{n+1} = 2a_n$ for n = 0, 1, 2, …

Obviously, induction and recursion are similar principles.

Recursively Defined Functions

We can use the following method to define a function with the **natural numbers** as its domain:

Specify the value of the function at zero.

Give a rule for finding its value at any integer from its values at smaller integers.

Such a definition is called **recursive** or **inductive definition.**

Recursively Defined Functions

Example:

f(0) = 3

f(n + 1) = 2f(n) + 3

f(0) = 3

f(1) = 2f(0) + 3 = 2·3 + 3 = 9

f(2) = 2f(1) + 3 = 2·9 + 3 = 21

f(3) = 2f(2) + 3 = 2·21 + 3 = 45

f(4) = 2f(3) + 3 = 2·45 + 3 = 93

Recursively Defined Functions

How can we recursively define the factorial function f(n) = n! ?

f(0) = 1

f(n + 1) = (n + 1)f(n)

f(0) = 1

f(1) = 1f(0) = 1·1 = 1

f(2) = 2f(1) = 2·1 = 2

f(3) = 3f(2) = 3·2 = 6

f(4) = 4f(3) = 4·6 = 24

Recursively Defined Functions

A famous example: The Fibonacci numbers

f(0) = 0, f(1) = 1

f(n) = f(n – 1) + f(n − 2)

f(0) = 0

f(1) = 1

f(2) = f(1) + f(0) = 1 + 0 = 1

f(3) = f(2) + f(1) = 1 + 1 = 2

f(4) = f(3) + f(2) = 2 + 1 = 3

f(5) = f(4) + f(3) = 3 + 2 = 5

f(6) = f(5) + f(4) = 5 + 3 = 8

Recursively Defined Sets

If we want to recursively define a set A, we need to provide two things:

an **initial set** of elements,

- **rules** for the construction of **additional** elements from elements in the set. Example: $x_1, x_2, \ldots x_n \in A \rightarrow R(x_1, \ldots, x_n) \in A$

When we want to prove P(x) is true for all x in a recursively defined set A we must prove

- P(x) is true for each element of the initial set of A.

For each rule generating new elements, if $P(x_1)$, $P(x_2,)$,…, $P(x_n\ )$ are true then $P(R(x_1,\ldots,x_n))$ is true

Recursively Defined Sets

Example: Let S be recursively defined by:
$3 \in S$
$(x + y) \in S$ if $(x \in S)$ and $(y \in S)$
S is the set of positive integers divisible by 3.

Proof:
Let A be the set of all positive integers divisible by 3.

To show that A = S, we must show that
$A \subseteq S$ and $S \subseteq A$.

Part I: To prove that $A \subseteq S$, we must show that every positive integer divisible by 3 is in S.

We will use mathematical induction to show this.

Recursively Defined Sets

Let P(n) be the statement “3n belongs to S”.
Basis step: P(1) is true, because 3 is in S.

Inductive step: To show:
If P(n) is true, then P(n + 1) is true.

Assume 3n is in S. Since 3n is in S and 3 is in S, it follows from the recursive definition of S that $3n + 3 = 3(n + 1)$ is also in S.

Conclusion of Part I: $A \subseteq S$.

Recursively Defined Sets

Part II: To show: $S \subseteq A$.

Basis step: To show:
All initial elements of S are in A. 3 is in A. True.

Inductive step: To show:
(x + y) is in A whenever x and y are in S.
If x and y are both in A, it follows that $3 \mid x$ and $3 \mid y$. As we already know,
it follows that $3 \mid (x + y)$.

Conclusion of Part II: $S \subseteq A$.
Overall conclusion: A = S.

Recursively Defined Sets

Another example:
The well-formed formulas of variables, numerals and operators from {+, −, *, /, ^} are defined by

x is a well-formed formula if x is a numeral or variable.

(f + g), (f – g), (f * g), (f / g), (f ^ g) are well-formed formulas if f and g are.

Recursively Defined Sets

With this definition, we can construct formulas such as:
(x – y)
((z / 3) – y)
((z / 3) – (6 + 5))
((z / (2 * 4)) – (6 + 5))

Recursive Algorithms

An algorithm is called **recursive** if it solves a problem by reducing it to an instance of the same problem with smaller input.

Example I: Recursive Euclidean Algorithm

procedure gcd(a, b: nonnegative integers with a < b)
if a = 0 **then** gcd(a, b) := b
else gcd(a, b) := gcd(b **mod** a, a)

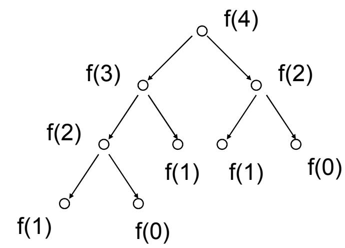
Recursive Algorithms

Example II: Recursive Fibonacci Algorithm

procedure fibo(n: nonnegative integer)
if n = 0 **then** fibo(0) := 0
else if n = 1 **then** fibo(1) := 1
else fibo(n) := fibo(n – 1) + fibo(n – 2)

Recursive Algorithms

Recursive Fibonacci Evaluation:



Recursive Algorithms

```
procedure iterative_fibo(n: nonnegative integer)
if n = 0 then y := 0
else
begin
        x := 0
        y := 1
        for i := 1 to n-1
        begin
                z := x + y
                x : = y
                y := z
        end
end  {y is the n-th Fibonacci number}
```


Recursive Algorithms

For every recursive algorithm, there is an **equivalent** iterative algorithm.

Recursive algorithms are often **shorter**, **more elegant**, and **easier to understand** than their iterative counterparts.

However, iterative algorithms are usually **more efficient** in their use of space and time.

Program Verification

Proof that a program works correctly is difficult. One approach is to attach statements about the state of the program (values of the variables) and prove thereby that sequences of statements will do what you expect. See section 5.5, p 372

(4.5, p 322 in 6th edition)

Partial Correctness

Def. A program segment S is partially correct with respect to initial assertion p and final assertion q, if whenever p is true and S is executed and terminates then q will be true.

In this case we write: p{S}q

Composition Rule

$$\frac{\begin{array}{l} p\{S_1\}q \\ q\{S_2\}r \end{array}}{p\{S_1;\ S_2\}r}$$

This means that we can combine the assertions about S_1 and S_2 to get an assertion about what happens when we execute first S_1 and then S_2.

Example

Suppose p is T, q is "$x > 0$", r is "$y > 0$"

Then $p\{x := 4\}q$, and

$q\{y := 2*x\}r$ are correct,

and thus so is

$p\{x := 4;\ y := 2*x\}r$.

Conditionals 1

$$\frac{\begin{array}{l} (p \wedge \mathit{condition})\ \{S\}\ q \\ (p \wedge \neg\mathit{condition}) \rightarrow q \end{array}}{p\ \{\ \text{if } \mathit{condition} \text{ then } S\}\ q}$$

Here we get a correctness condition on execution of S giving us a correctness condition for the conditional.

Conditionals 2

$$\frac{\begin{array}{l} (p \wedge \mathit{condition})\ \{S_1\}\ q \\ (p \wedge \neg\mathit{condition})\ \{S_2\}\ q \end{array}}{p\ \{\ \text{if } \mathit{condition} \text{ then } S_1 \text{ else } S_2\}\ q}$$

Similar thing, for if-then-else.

Loop Invariants

$$\frac{(p \wedge \mathit{condition})\ \{S\}\ p}{p\ \{\text{while } \mathit{condition}\ S\}\ (\neg\mathit{condition} \wedge p)}$$

Here p is called a loop invariant because it remains true on each pass through the loop.

We usually pick a loop invariant carefully to establish some fact we want.

Example of loop invariants

We can use loop invariants to prove that binary search is correct.
(searching for x in an ordered sequence $a_1, \cdots, a_n$)

```
i := 1; k:= n;
while (i < k) {
   m := ⌊(i+k)/2⌋;
   if (x > a_m) then i := m+1;  // i_post = m+1, k_post = k_pre

                  else k := m;     // i_post = i_pre, k_post = m
}
if (x = a_i) then location := i;
              else location := 0;
    // A loop invariant that works:
    // p: (x = a_j for some j)→(a_i ≤ x ≤ a_k)∧(i ≤ k)
```

To see this loop invariant works:

1. Check that $i \le k$ is invariant.
2. $m = \lfloor (i+k)/2 \rfloor = \lfloor k - (k-i)/2 \rfloor = k - \lceil (k-i)/2 \rceil$, so $m < k$ (if $i < k$).
3. $m = \lfloor (i+k)/2 \rfloor = \lfloor i+(k-i)/2 \rfloor = i + \lfloor (k-i)/2 \rfloor$, so $m \ge i$.
4. Thus $i \le m < k$ on each pass through the loop and $i < m < k$ unless $i+1=k$. (i, k are i_{pre}, k_{pre})

Now check that $a_i \le x \le a_k$ is invariant on each pass through the loop.

if ($x > a_m$)
then $i_{post} := m+1$;
// so $a_{m+1} \le x \le a_k$, if x is one of the a_j
else $k_{post} := m$;
// so $a_i \le x \le a_m$

Thus in each case we also have $a_i \le x \le a_k$ after each pass through the loop.