Discrete Mathematics 1

Chapter 4: Induction and Recursions

Department of Mathematics
The FPT university

Chapter 4: Introduction

Topics covered:

Topics covered:

4.1 Mathematical Induction

Chapter 4: Introduction

Topics covered:

4.1 Mathematical Induction

4.2 Strong Induction and Well-Ordering

Topics covered:

4.1 Mathematical Induction

4.2 Strong Induction and Well-Ordering

4.3 Recursive Definitions and Structural Induction

Topics covered:

4.1 Mathematical Induction

4.2 Strong Induction and Well-Ordering

4.3 Recursive Definitions and Structural Induction

4.4 Recursive Algorithms

Topics covered:

4.1 Mathematical Induction

4.2 Strong Induction and Well-Ordering

4.3 Recursive Definitions and Structural Induction

4.4 Recursive Algorithms

4.5 Program Correctness

4.1 Mathematical Induction

Problem. Prove that the statement $P(n)$ is true for all $n = 1, 2, \ldots$

Problem. Prove that the statement $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Induction:

Problem. Prove that the statement $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Induction:

1: Prove that $P(1)$ is true.

Problem. Prove that the statement $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Induction:

1: Prove that $P(1)$ is true.

2: (Inductive hypothesis) Assume that $P(k)$ is true for some positive integer k.

Problem. Prove that the statement $P(n)$ is true for all $n = 1, 2, \dots$

Proof by Induction:

1: Prove that $P(1)$ is true.

2: (Inductive hypothesis) Assume that $P(k)$ is true for some positive integer k.

3: Prove that $P(k+1)$ is true.

Problem. Prove that the statement $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Induction:

1: Prove that $P(1)$ is true.
2: (Inductive hypothesis) Assume that $P(k)$ is true for some positive integer k.
3: Prove that $P(k+1)$ is true.
4: Conclusion: $P(n)$ is true for all positive integers n.

Problem. Prove that the statement $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Induction:

1: Prove that $P(1)$ is true.

2: (Inductive hypothesis) Assume that $P(k)$ is true for some positive integer k.

3: Prove that $P(k+1)$ is true.

4: Conclusion: $P(n)$ is true for all positive integers n.

Example 1. Prove that for all positive integers n we have
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Example 1. Prove that for all positive integers n we have

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Example 2. Prove that $n^3 - n$ is divisible by 6 for all integers $n \geq 1$.

Example 1. Prove that for all positive integers n we have
$1^2 + 2^2 + 3^2 + \cdots + n^2 = \dfrac{n(n+1)(2n+1)}{6}$

Example 2. Prove that $n^3 - n$ is divisible by 6 for all integers $n \geq 1$.

Example 3. Prove that $2^n > n^2$ for all integers $n > 4$.

Example 1. Prove that for all positive integers n we have
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Example 2. Prove that $n^3 - n$ is divisible by 6 for all integers $n \geq 1$.

Example 3. Prove that $2^n > n^2$ for all integers $n > 4$.

Example 4. Let n be a positive integer. Prove that every checkerboard of size $2^n \times 2^n$ with one square removed can be titled by triominoes.

4.2 Strong Induction and Well-Ordering

Problem. Prove that $P(n)$ is true for all $n = 1, 2, \ldots$

Problem. Prove that $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Strong Induction.

Problem. Prove that $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Strong Induction.

1: Prove that $P(1)$ is true.

Problem. Prove that $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Strong Induction.

1: Prove that $P(1)$ is true.

2: (Induction hypothesis) Assume that $P(1), P(2), \ldots, P(k)$ are all true for some $k \geq 1$.

4.2 Strong Induction and Well-Ordering

Problem. Prove that $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Strong Induction.

1: Prove that $P(1)$ is true.

2: (Induction hypothesis) Assume that $P(1), P(2), \ldots, P(k)$ are all true for some $k \geq 1$.

3: Prove that $P(k+1)$ is also true.

4.2 Strong Induction and Well-Ordering

Problem. Prove that $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Strong Induction.

1: Prove that $P(1)$ is true.

2: (Induction hypothesis) Assume that $P(1), P(2), \ldots, P(k)$ are all true for some $k \geq 1$.

3: Prove that $P(k+1)$ is also true.

4: Conclusion: $P(n)$ is true for all positive integers n.

4.2 Strong Induction and Well-Ordering

Problem. Prove that $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Strong Induction.

1: Prove that $P(1)$ is true.

2: (Induction hypothesis) Assume that $P(1), P(2), \ldots, P(k)$ are all true for some $k \geq 1$.

3: Prove that $P(k+1)$ is also true.

4: Conclusion: $P(n)$ is true for all positive integers n.

Example 1. Prove that every integer greater than 1 can be written as a product of primes.

4.2 Strong Induction and Well-Ordering

Problem. Prove that $P(n)$ is true for all $n = 1, 2, \ldots$

Proof by Strong Induction.

1: Prove that $P(1)$ is true.

2: (Induction hypothesis) Assume that $P(1), P(2), \ldots, P(k)$ are all true for some $k \geq 1$.

3: Prove that $P(k+1)$ is also true.

4: Conclusion: $P(n)$ is true for all positive integers n.

Example 1. Prove that every integer greater than 1 can be written as a product of primes.

Example 2. Prove that every postage of 12 cents or more can be formed using only 4-cent and 5-cent stamps.

Well-Ordering

Well-Ordering

The validity of the Principle of Mathematical Induction follows from the Well-Ordering property of the set of non-negative integers.

The validity of the Principle of Mathematical Induction follows from the Well-Ordering property of the set of non-negative integers.

Well-Ordering

The validity of the Principle of Mathematical Induction follows from the Well-Ordering property of the set of non-negative integers.

Well-Ordering

Any nonempty set of non-negative integers has a least element.

4.3 Recursive Definitions and Structural Induction

The Fibonacci $\{F_n\}, n = 1, 2, \ldots$ is defined as follows:

The Fibonacci $\{F_n\}, n = 1, 2, \ldots$ is defined as follows:

$F_0 = 0,\ F_1 = 1,$

The Fibonacci $\{F_n\}, n = 1, 2, \ldots$ is defined as follows:

$F_0 = 0,\ F_1 = 1$, and

The Fibonacci $\{F_n\}, n = 1, 2, \ldots$ is defined as follows:

$F_0 = 0,\ F_1 = 1$, and
$F_n = F_{n-1} + F_{n-2}$ for $n = 2, 3, \ldots$.

The Fibonacci $\{F_n\}, n = 1, 2, \ldots$ is defined as follows:

$F_0 = 0,\ F_1 = 1$, and
$F_n = F_{n-1} + F_{n-2}$ for $n = 2, 3, \ldots$.

This definition is called recursive definition.

The Fibonacci $\{F_n\}, n = 1, 2, \ldots$ is defined as follows:

$F_0 = 0,\ F_1 = 1$, and
$F_n = F_{n-1} + F_{n-2}$ for $n = 2, 3, \ldots$.

This definition is called recursive definition.

Example 1. Find the nth term of the sequence $\{x_n\}$ defined by recursive definition:

Example 1. Find the nth term of the sequence $\{x_n\}$ defined by recursive definition:

(a) $x_1 = 5$, $x_n = 3x_{n-1}$ for $n = 2, 3, \ldots$

Example 1. Find the nth term of the sequence $\{x_n\}$ defined by recursive definition:

(a) $x_1 = 5$, $x_n = 3x_{n-1}$ for $n = 2, 3, \ldots$

(b) $x_0 = 2$, $x_n = x_{n-1} + 1$ for $n = 1, 2, \ldots$

Example 1. Find the nth term of the sequence $\{x_n\}$ defined by recursive definition:

(a) $x_1 = 5$, $x_n = 3x_{n-1}$ for $n = 2, 3, \ldots$

(b) $x_0 = 2$, $x_n = x_{n-1} + 1$ for $n = 1, 2, \ldots$

Example 1. Find the nth term of the sequence $\{x_n\}$ defined by recursive definition:

(a) $x_1 = 5$, $x_n = 3x_{n-1}$ for $n = 2, 3, \ldots$

(b) $x_0 = 2$, $x_n = x_{n-1} + 1$ for $n = 1, 2, \ldots$

Example 2. Give a recursive definition for the sequence $\{x_n\}, n = 1, 2 \ldots$ whose nth term is:

Example 1. Find the nth term of the sequence $\{x_n\}$ defined by recursive definition:

(a) $x_1 = 5$, $x_n = 3x_{n-1}$ for $n = 2, 3, \ldots$

(b) $x_0 = 2$, $x_n = x_{n-1} + 1$ for $n = 1, 2, \ldots$

Example 2. Give a recursive definition for the sequence $\{x_n\}, n = 1, 2 \ldots$ whose nth term is:

(a) $x_n = 7 * 5^{n+1}$

Example 1. Find the nth term of the sequence $\{x_n\}$ defined by recursive definition:

(a) $x_1 = 5$, $x_n = 3x_{n-1}$ for $n = 2, 3, \ldots$

(b) $x_0 = 2$, $x_n = x_{n-1} + 1$ for $n = 1, 2, \ldots$

Example 2. Give a recursive definition for the sequence $\{x_n\}, n = 1, 2 \ldots$ whose nth term is:

(a) $x_n = 7 * 5^{n+1}$

(b) $x_n = n!$

Example 1. Find the nth term of the sequence $\{x_n\}$ defined by recursive definition:

(a) $x_1 = 5$, $x_n = 3x_{n-1}$ for $n = 2, 3, \ldots$

(b) $x_0 = 2$, $x_n = x_{n-1} + 1$ for $n = 1, 2, \ldots$

Example 2. Give a recursive definition for the sequence $\{x_n\}, n = 1, 2 \ldots$ whose nth term is:

(a) $x_n = 7 * 5^{n+1}$

(b) $x_n = n!$

(c) $x_n = (-1)^n$

Example 1. Find the nth term of the sequence $\{x_n\}$ defined by recursive definition:

(a) $x_1 = 5$, $x_n = 3x_{n-1}$ for $n = 2, 3, \ldots$

(b) $x_0 = 2$, $x_n = x_{n-1} + 1$ for $n = 1, 2, \ldots$

Example 2. Give a recursive definition for the sequence $\{x_n\}, n = 1, 2 \ldots$ whose nth term is:

(a) $x_n = 7 * 5^{n+1}$

(b) $x_n = n!$

(c) $x_n = (-1)^n$

(d) $x_n = 2n - 6$

Example 1. Find the nth term of the sequence $\{x_n\}$ defined by recursive definition:

(a) $x_1 = 5$, $x_n = 3x_{n-1}$ for $n = 2, 3, \ldots$
(b) $x_0 = 2$, $x_n = x_{n-1} + 1$ for $n = 1, 2, \ldots$

Example 2. Give a recursive definition for the sequence $\{x_n\}, n = 1, 2 \ldots$ whose nth term is:

(a) $x_n = 7 * 5^{n+1}$
(b) $x_n = n!$
(c) $x_n = (-1)^n$
(d) $x_n = 2n - 6$

Recursively Defined Sets and Structures

Example 1. Determine the set S defined by:

Example 1. Determine the set S defined by:

Basic step: $3 \in S$

Example 1. Determine the set S defined by:

Basic step: $3 \in S$

Recursive step: If $x, y \in S$ then $x + y \in S$

Example 1. Determine the set S defined by:

Basic step: $3 \in S$

Recursive step: If $x, y \in S$ then $x + y \in S$

Example 2.

Example 1. Determine the set S defined by:

Basic step: $3 \in S$

Recursive step: If $x, y \in S$ then $x + y \in S$

Example 2.

(a) Give a recursive definition for the set of positive integers that are not divisible by 3

Example 1. Determine the set S defined by:

Basic step: $3 \in S$

Recursive step: If $x, y \in S$ then $x + y \in S$

Example 2.

(a) Give a recursive definition for the set of positive integers that are not divisible by 3

(b) Give a recursive definition for the set of integers that are not divisible by 3

Example 1. Determine the set S defined by:

Basic step: $3 \in S$

Recursive step: If $x, y \in S$ then $x + y \in S$

Example 2.

(a) Give a recursive definition for the set of positive integers that are not divisible by 3

(b) Give a recursive definition for the set of integers that are not divisible by 3

Example 3. Recursive definition for the set of full binary trees.

Example 3. Recursive definition for the set of full binary trees.

Basic step: A single vertex is a full binary tree.

Example 3. Recursive definition for the set of full binary trees.

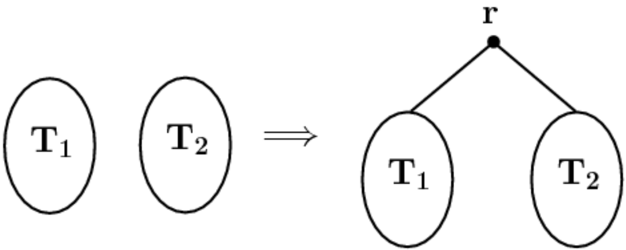
Basic step: A single vertex is a full binary tree.

Recursive step: If T_1 and T_2 are two full binary trees then there is a full binary tree, denoted by $T_1.T_2$, consisting of a root r together with edges connecting this root to the root of the left subtree T_1 and the root of the right subtree T_2.

Example 3. Recursive definition for the set of full binary trees.

Basic step: A single vertex is a full binary tree.

Recursive step: If T_1 and T_2 are two full binary trees then there is a full binary tree, denoted by $T_1.T_2$, consisting of a root r together with edges connecting this root to the root of the left subtree T_1 and the root of the right subtree T_2.



Example 4. Give a recursive definition for:

Example 4. Give a recursive definition for:

(a) Leaves of full binary trees.

Example 4. Give a recursive definition for:

(a) Leaves of full binary trees.
(b) Height of full binary trees.

Example 4. Give a recursive definition for:

(a) Leaves of full binary trees.

(b) Height of full binary trees.

Structural Induction

Structural Induction

Let S be a set defined recursively.

Structural Induction

Let S be a set defined recursively. To prove that a property P is true for all elements of S we can use structural induction.

Structural Induction

Let S be a set defined recursively. To prove that a property P is true for all elements of S we can use structural induction.

Basic step: Prove that P is true for elements of S defined in the basic step.

Structural Induction

Let S be a set defined recursively. To prove that a property P is true for all elements of S we can use structural induction.

Basic step: Prove that P is true for elements of S defined in the basic step.

Recursive step: Show that if the property P is true for the elements used to construct new elements in the recursive step of the definition of S, then the property P is also true for these new elements.

Example 1.

Example 1. Let T be a full binary tree with the number of vertices $n(T)$ and the number of leaves $l(T)$. Prove that $n(T) = 2l(T) - 1$.

Example 1. Let T be a full binary tree with the number of vertices $n(T)$ and the number of leaves $l(T)$. Prove that $n(T) = 2l(T) - 1$.

Example 2. Let T be a full binary tree with the number of vertices $n(T)$ and the height $h(T)$. Prove that $n(T) \leq 2^{h(T)+1} - 1$.

Example 1. Let T be a full binary tree with the number of vertices $n(T)$ and the number of leaves $l(T)$. Prove that $n(T) = 2l(T) - 1$.

Example 2. Let T be a full binary tree with the number of vertices $n(T)$ and the height $h(T)$. Prove that $n(T) \leq 2^{h(T)+1} - 1$.

Generalized Induction

Example.

Example.

Given the sequence $\{a_{m,n}\}$ defined recursively as follows:

Generalized Induction

Example.

Given the sequence $\{a_{m,n}\}$ defined recursively as follows:

$a_{0,0} = 0$, and

Example.

Given the sequence $\{a_{m,n}\}$ defined recursively as follows:

$a_{0,0} = 0$, and

$$a_{m,n} = \begin{cases} a_{m-1,n} + 1 & \text{if } n = 0 \vee m > 0, \\ a_{m,n-1} + n & \text{if } n > 0 \end{cases}$$

Example.

Given the sequence $\{a_{m,n}\}$ defined recursively as follows:

$a_{0,0} = 0$, and

$$a_{m,n} = \begin{cases} a_{m-1,n} + 1 & \text{if } n = 0 \text{ v } m > 0, \\ a_{m,n-1} + n & \text{if } n > 0 \end{cases}$$

Prove that $a_{m,n} = m + n(n+1)/2$ for all $m, n \geq 0$.

4.4 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

4.4 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 1. A recursive algorithm that computes 5^n for $n \geq 0$.

4.4 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 1. A recursive algorithm that computes 5^n for $n \geq 0$.

Procedure power (n: non-negative)
if $n = 0$ then $power(0) := 1$
else $power(n) := power(n-1) * 5$

Example 2. Write a recursive algorithm to compute $n!$.

Example 2. Write a recursive algorithm to compute $n!$.

Example 3. Write a recursive algorithm to compute the greatest common divisor of two non-negative integers.

Example 2. Write a recursive algorithm to compute $n!$.

Example 3. Write a recursive algorithm to compute the greatest common divisor of two non-negative integers.

Example 4. Express the linear search algorithm by a recursive procedure .

Example 2. Write a recursive algorithm to compute $n!$.

Example 3. Write a recursive algorithm to compute the greatest common divisor of two non-negative integers.

Example 4. Express the linear search algorithm by a recursive procedure .

Example 5. Express the binary search algorithm by a recursive procedure.

Example 2. Write a recursive algorithm to compute $n!$.

Example 3. Write a recursive algorithm to compute the greatest common divisor of two non-negative integers.

Example 4. Express the linear search algorithm by a recursive procedure .

Example 5. Express the binary search algorithm by a recursive procedure.

Recursion and Iteration

Problem. Write a recursive algorithm and an iteration algorithm to compute the nth Fibonacci number, and compare their complexity via the number of additions used.

Recursion and Iteration

Problem. Write a recursive algorithm and an iteration algorithm to compute the nth Fibonacci number, and compare their complexity via the number of additions used.

Problem. Write a recursive algorithm and an iteration algorithm to compute the nth Fibonacci number, and compare their complexity via the number of additions used.

Procedure Iterative Fib (n)
if $n = 0$ then $y := 0$
else
  x:=0
  y:=1
  for $i := 1$ to $n - 1$ do
    z:=x+y
    x:=y
    y:=z
Print(y)

Problem. Write a recursive algorithm and an iteration algorithm to compute the nth Fibonacci number, and compare their complexity via the number of additions used.

Procedure Iterative Fib (n)
if $n = 0$ then $y := 0$
else
  x:=0
  y:=1
  for $i := 1$ to $n - 1$ do
    z:=x+y
    x:=y
    y:=z
Print(y)

Procedure Fib (n)
if $n = 0$ then $Fib(0) := 0$
else if $n = 1$ then $Fib(1) := 1$
else
$Fib(n) := Fib(n-1) + Fib(n-2)$

Merge Sort Algorithm

Merge Sort Algorithm

Procedure mergesort $(L = a_1, a_2, \ldots, a_n)$
if $n > 1$ **then**

$m := \lfloor n/2 \rfloor$

$L_1 = a_1, a_2, \ldots, a_m$

$L_2 = a_{m+1}, a_{m+2}, \ldots, a_n$

$L := merge(mergesort(L_1), mergersort(L_2))$

Print (L)

Merge Sort Algorithm

Procedure mergesort $(L = a_1, a_2, \ldots, a_n)$
if $n > 1$ **then**
$m := \lfloor n/2 \rfloor$
$L_1 = a_1, a_2, \ldots, a_m$
$L_2 = a_{m+1}, a_{m+2}, \ldots, a_n$
$L := merge(mergesort(L_1), mergersort(L_2))$
Print (L)

Theorem

Merge Sort Algorithm

Procedure mergesort $(L = a_1, a_2, \ldots, a_n)$
if $n > 1$ **then**
$m := \lfloor n/2 \rfloor$
$L_1 = a_1, a_2, \ldots, a_m$
$L_2 = a_{m+1}, a_{m+2}, \ldots, a_n$
$L := merge(mergesort(L_1), mergersort(L_2))$
Print (L)

Theorem

The number of comparisons needed to merge sort a list of n elements is $O(n \log n)$.

4.5 Program Correctness

4.5 Program Correctness

A program is called correct if it produces correct output for every possible input.

4.5 Program Correctness

A program is called correct if it produces correct output for every possible input. A proof that a program is correct consists of two steps:

A program is called correct if it produces correct output for every possible input. A proof that a program is correct consists of two steps:

- Prove that if the program terminates then the answer is correct (this part is called partial correctness).

4.5 Program Correctness

A program is called correct if it produces correct output for every possible input. A proof that a program is correct consists of two steps:

- Prove that if the program terminates then the answer is correct (this part is called partial correctness).
- Show that the program always terminates.

A program segment S is called partially correct with respect to the initial assertion p and the final assertion q, if whenever p is true for the input values and after S terminates then q is true for the output values.

A program segment S is called partially correct with respect to the initial assertion p and the final assertion q, if whenever p is true for the input values and after S terminates then q is true for the output values.
The notation $p\{S\}q$ (Hoare triple) indicates that the program segment S is (partially) correct with respect to the initial assertion p and the final assertion q.

A program segment S is called partially correct with respect to the initial assertion p and the final assertion q, if whenever p is true for the input values and after S terminates then q is true for the output values. The notation $p\{S\}q$ (Hoare triple) indicates that the program segment S is (partially) correct with respect to the initial assertion p and the final assertion q.

Example. The program segment

y:=2
z:=x+y

is

A program segment S is called partially correct with respect to the initial assertion p and the final assertion q, if whenever p is true for the input values and after S terminates then q is true for the output values.
The notation $p\{S\}q$ (Hoare triple) indicates that the program segment S is (partially) correct with respect to the initial assertion p and the final assertion q.

Example. The program segment

```
y:=2
z:=x+y
```

is

- correct with respect to the initial assertion $p : x = 1$ and the final assertion $q : z = 3$

A program segment S is called partially correct with respect to the initial assertion p and the final assertion q, if whenever p is true for the input values and after S terminates then q is true for the output values.
The notation $p\{S\}q$ (Hoare triple) indicates that the program segment S is (partially) correct with respect to the initial assertion p and the final assertion q.

Example. The program segment

```
y:=2
z:=x+y
```

is

- correct with respect to the initial assertion $p : x = 1$ and the final assertion $q : z = 3$
- is not correct with respect to the initial assertion $p : x < 5$ and the final assertion $q : z > 10$

Rules of Inference for Condition Statements

Rules of Inference for Condition Statements

Consider a program segment

Rules of Inference for Condition Statements

Consider a program segment

Rules of Inference for Condition Statements

Consider a program segment

if condition **then**
    S

Rules of Inference for Condition Statements

Consider a program segment

if condition **then**
    S

To verify the correctness of this program with respect to the initial assertion p and the final assertion q we use the following rule of inference:

Rules of Inference for Condition Statements

Consider a program segment

if condition **then**
    S

To verify the correctness of this program with respect to the initial assertion p and the final assertion q we use the following rule of inference:

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

Consider a program segment

Consider a program segment

Consider a program segment

if condition **then**
    S_1
else
    S_2

Consider a program segment

if condition **then**

S_1

else

S_2

To verify the correctness of this program with respect to the initial assertion p and the final assertion q we use the following rule of inference:

Consider a program segment

if condition **then**
    S_1
else
    S_2

To verify the correctness of this program with respect to the initial assertion p and the final assertion q we use the following rule of inference:

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

Loop Invariants

Loop Invariants

Consider the program segment

Loop Invariants

Consider the program segment

```
while condition
    S
```

Consider the program segment

while condition
    S

To develope a rule of inference for this type of program, we must choose a statement that remains true each time S is executed.

Loop Invariants

Consider the program segment

```
while condition
      S
```

To develope a rule of inference for this type of program, we must choose a statement that remains true each time S is executed. Such a statement is called a loop invariant.

Consider the program segment

while condition
    S

To develope a rule of inference for this type of program, we must choose a statement that remains true each time S is executed. Such a statement is called a loop invariant.

If a loop invariant p is chosen then the rule of inference for this program segment is:

Loop Invariants

Consider the program segment

while condition
    S

To develope a rule of inference for this type of program, we must choose a statement that remains true each time S is executed. Such a statement is called a loop invariant.

If a loop invariant p is chosen then the rule of inference for this program segment is:

$$\frac{(p \wedge \text{ condition})\{S\}p}{\therefore p\{\textbf{while} \text{ condition } \textbf{then } S\}\ (\neg \text{ condition } \wedge p)}$$

Loop Invariants

Consider the program segment

while condition
    S

To develope a rule of inference for this type of program, we must choose a statement that remains true each time S is executed. Such a statement is called a loop invariant.

If a loop invariant p is chosen then the rule of inference for this program segment is:

$$\frac{(p \wedge \text{ condition})\{S\}p}{\therefore\ p\{\textbf{while}\ \text{ condition } \textbf{then}\ S\}\ (\neg \text{ condition } \wedge p)}$$

Example 1. Given the program segment:

Example 1. Given the program segment:

$i := 1$
factorial $:= 1$
while $i < n$
begin
    $i := i + 1$
    factorial $:=$ *factorial* $* i$
end

Example 1. Given the program segment:

$i := 1$
$factorial := 1$
while $i < n$
begin
  $i := i + 1$
  $factorial := factorial * i$
end

A loop invariant is $p : (factorial = i!) \wedge (i \leq n)$.

Example 1. Given the program segment:

$i := 1$
$factorial := 1$
while $i < n$
begin
  $i := i + 1$
  $factorial := factorial * i$
end

A loop invariant is $p : (factorial = i!) \wedge (i \leq n)$. Therefore when the program terminates we obtain

Example 1. Given the program segment:

$i := 1$
$factorial := 1$
while $i < n$
begin
  $i := i + 1$
  $factorial := factorial * i$
end

A loop invariant is $p : (factorial = i!) \wedge (i \leq n)$. Therefore when the program terminates we obtain

$p \wedge \neg$ condition $= (factorial = i!) \wedge (i = n)$,

Example 1. Given the program segment:

$i := 1$
$factorial := 1$
while $i < n$
begin
  $i := i + 1$
  $factorial := factorial * i$
end

A loop invariant is $p : (factorial = i!) \wedge (i \leq n)$. Therefore when the program terminates we obtain

$p \wedge \neg$ condition $= (factorial = i!) \wedge (i = n)$, then $factorial = n!$.

Example 2. Find a loop invariant for the program segment

Example 2. Find a loop invariant for the program segment

$power := 1$
$i := 1$
while $i \leq n$
begin
    $power := power * x$
    $i := i + 1$
end