Reading and Writing Mathematical Proofs

Slides by Arthur van Goetham

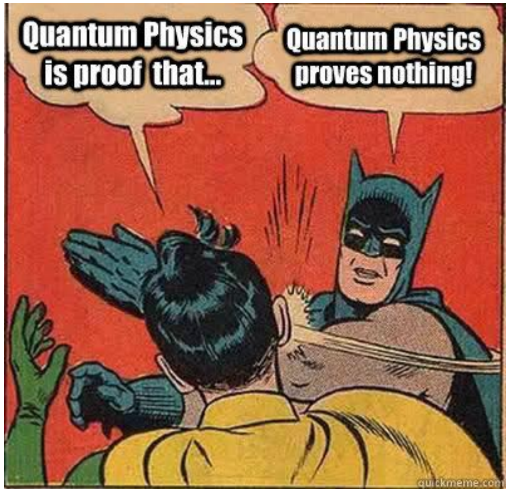
What is a proof?

Why explanations are not proofs…

What is a proof?

A method for establishing truth

What establishes truth depends on context



Physics	Sufficient experimental evidence
Courtroom	Admissible evidence and witness testimony
Mathematical proof	Not a doubt possible!

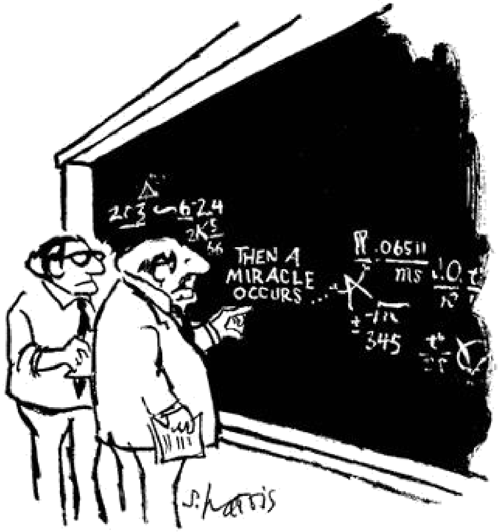
What is a proof?

A form of communication

Proof must convince reader (*not the writer!*) of correctness

Proofs must be:

- Clearly written
  - Should be easy to follow
  - Very different from “proving process”
- Very precise
  - No ambiguities!
- Leaving no doubts



“I think you should be more explicit here in step two.”

Definition

Mathematical proof

A convincing argument for the reader to establish the correctness of a mathematical statement without any doubt.

Definition

Mathematical proof

A convincing argument for the reader to establish the correctness of a $\boxed{\text{mathematical statement}}$ without any doubt.

Statement must be true or false

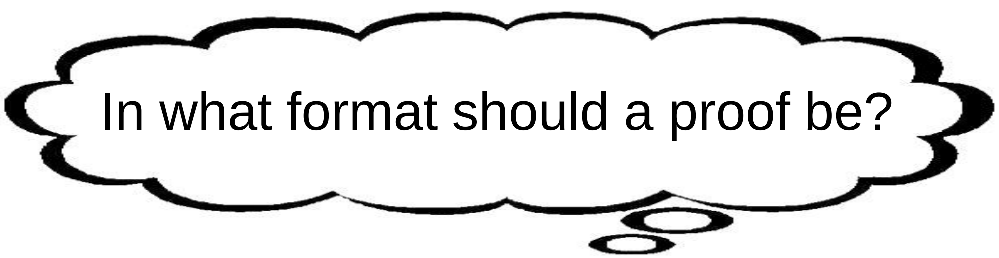
$\cancel{3 + 6}$

$3 + 6 = 9$ 

Definition

Mathematical proof

A convincing argument for the reader to establish the correctness of a mathematical statement without any doubt.



Logical derivation

{ Assume: }

(1) $\boxed{\textbf{var } x; x \in \mathbb{Z}}$

{ Assume: }

(2) $\boxed{\exists_k[x = 2k+1]}$

{ $\exists^*$-elim on (2): }

(3) $x = 2k+1$

{ Mathematics: }

(4) $x^2 = (2k+1)^2$

$= 4k^2 + 4k + 1$

$= 2(2k^2 + 2k) + 1$

{ $\exists^*$-intro on (4) with $m = 2k^2 + 2k$: }

(5) $\exists_m[x^2 = 2m+1]$

{ $\Rightarrow$-intro on (2) and (5): }

(6) $\exists_k[x = 2k+1] \Rightarrow \exists_m[x^2 = 2m+1]$

{ $\forall$-intro on (1) and (6) }

(7) $\forall_x[\exists_k[x = 2k+1] \Rightarrow \exists_m[x^2 = 2m+1]]$

Good

- Very systematic
- Hard to make mistakes

Bad

- Not convenient for statements not stated in logical formulas
- Emphasis on logical reasoning $\rightarrow$ detract from crux argument
- Hard to read
- Cumbersome

Common English

Theorem

If x is odd, then x^2 is odd

Proof

Since x is odd, there exists a $k \in \mathbb{Z}$ such that $x = 2k + 1$. Then,
$x^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1 = 2m+1$.

As there exists a $m \in \mathbb{Z}$ such that $x^2 = 2m + 1$, x^2 is odd. □

This is the kind of proof we expect in Data Structures!

Good

- Short and to the point
- Easy to read

Bad

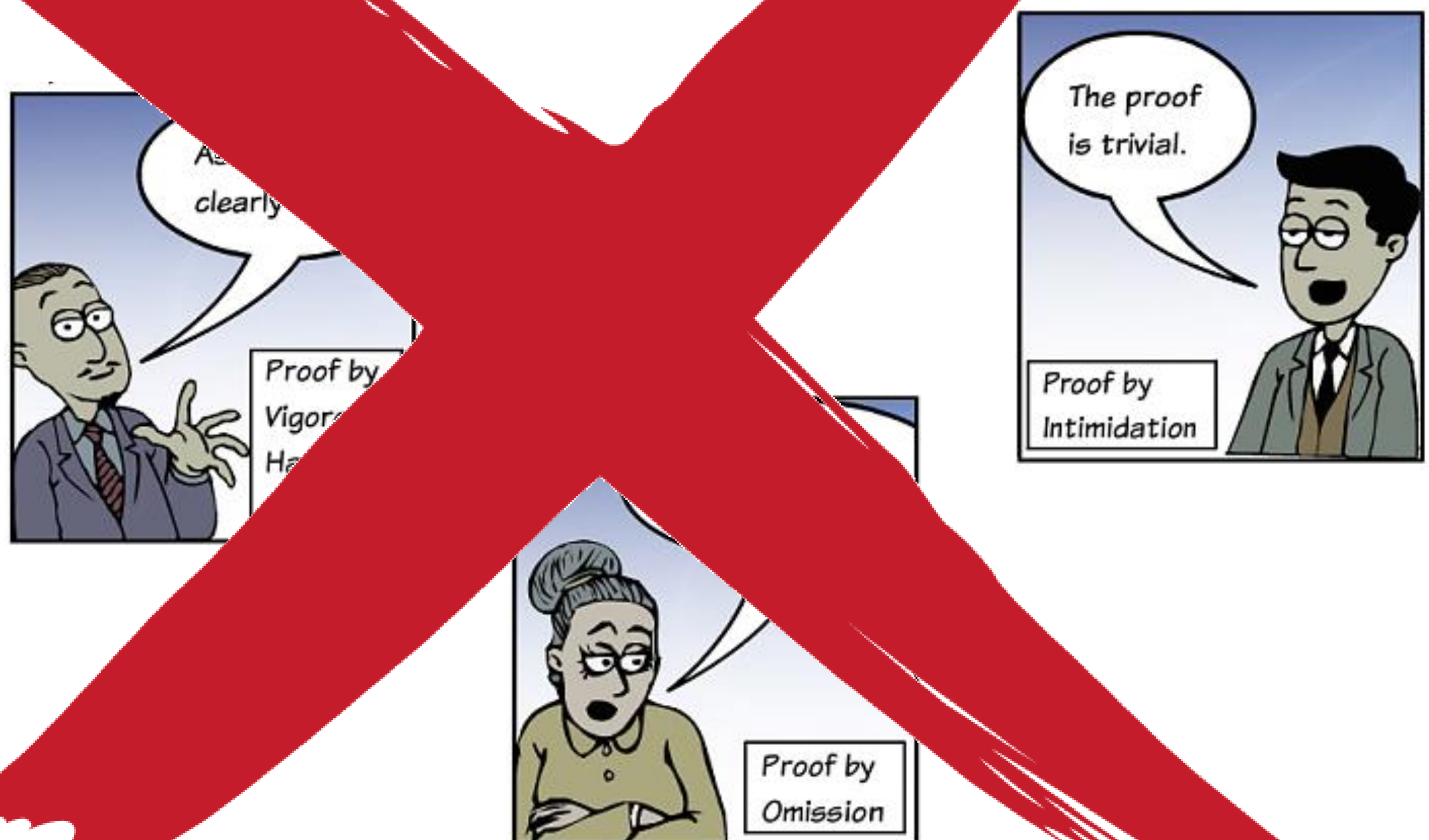
- Logical reasoning somewhat hidden
- Natural language can be ambiguous

Basic Proving Techniques

Proving 101…

Overview

Basi niques



Overview

Basic Proving Techniques

1. Forward-backward method
2. Mathematical induction
3. Case analysis
4. Proof by contradiction

Forward-Backward Method

How to get from A to B and B to A…

Forward-Backward Method

The most basic approach

Logically combine axioms, definitions, and earlier theorems (forward)

Simplify the goal (backward)

FACTS $\longrightarrow$ $\longleftarrow$ GOAL

This should always be your **default approach**

Usage

When to use?

Generally used for statements of the form: If P then Q

Premise — P; Goal — Q

Reason forward from the premise

Reason backward from the goal

Note

Thinking: Reasoning backward and forwards

Writing: Reason forward to keep the flow

Basic Example

Theorem

If my hamsters do excessive exercise, I will be tired in the morning.

Proof

My hamsters do excessive exercise

They are running in their exercise wheel

The exercise wheel is making noise

Something is keeping me awake

I do not get a good night's sleep

I am tired in the morning

Example

Theorem

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Proof

What does $f(n) = O(g(n))$ mean again?

There exist positive constants c and n_0 such that $f(n) \leq c\, g(n)$ for all $n \geq n_0$.

Example

Theorem

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Proof

By definition there exist positive constants c' and n_0' such that $f_1(n) \leq c' \, g_1(n)$ for all $n \geq n_0'$. Similarly, there exist positive constants c'' and n_0'' such that $f_2(n) \leq c'' \, g_2(n)$ for all $n \geq n_0''$. We need to show that there exist positive constants c and n_0 such that $f_1(n)f_2(n) \leq c \, g_1(n)g_2(n)$ for all $n \geq n_0$.

We must be very careful with variables! Use different names!

Example

Theorem

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Proof

By definition there exist positive constants c' and n_0' such that $f_1(n) \leq c' g_1(n)$ for all $n \geq n_0'$. Similarly, there exist positive constants c'' and n_0'' such that $f_2(n) \leq c'' g_2(n)$ for all $n \geq n_0''$. We need to show that there exist positive constants c and n_0 such that $f_1(n)f_2(n) \leq c\, g_1(n)g_2(n)$ for all $n \geq n_0$.

Example

Theorem

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Proof

By definition there exist positive constants c' and n_0' such that $f_1(n) \leq c' g_1(n)$ for all $n \geq n_0$'. Similarly, there exist positive constants c'' and n_0'' such that $f_2(n) \leq c'' g_2(n)$ for all $n \geq n_0$''. We need to show that there exist positive constants c and n_0 such that $f_1(n)f_2(n) \leq c\, g_1(n)g_2(n)$ for all $n \geq n_0$.

To establish this, we need to find suitable values for c and n_0.

Example

Theorem

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Proof

By definition there exist positive constants c' and n_0' such that $f_1(n) \le c' g_1(n)$ for all $n \ge n_0'$. Similarly, there exist positive constants c'' and n_0'' such that $f_2(n) \le c'' g_2(n)$ for all $n \ge n_0''$. We need to show that there exist positive constants c and n_0 such that $f_1(n)f_2(n) \le c\, g_1(n)g_2(n)$ for all $n \ge n_0$.

To establish this, we need to find suitable values for c and n_0.
We already are given constants c', c'', n_0', and n_0''.

Example

Theorem

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Proof

By definition there exist positive constants c' and n_0' such that $f_1(n) \le c' g_1(n)$ for all $n \ge n_0'$. Similarly, there exist positive constants c'' and n_0'' such that $f_2(n) \le c'' g_2(n)$ for all $n \ge n_0''$. We need to show that there exist positive constants c and n_0 such that $f_1(n)f_2(n) \le c\, g_1(n)g_2(n)$ for all $n \ge n_0$.

To establish this, we need to find suitable values for c and n_0.
We already are given constants c', c'', n_0', and n_0''.
Note that $f_1(n)f_2(n) \le c'g_1(n)\, c''g_2(n) = c'\, c''\, g_1(n)g_2(n)$,

Example

Theorem

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Proof

By definition there exist positive constants c' and n_0' such that $f_1(n) \leq c'\, g_1(n)$ for all $n \geq n_0'$. Similarly, there exist positive constants c'' and n_0'' such that $f_2(n) \leq c''\, g_2(n)$ for all $n \geq n_0''$. We need to show that there exist positive constants c and n_0 such that $f_1(n)f_2(n) \leq c\, g_1(n)g_2(n)$ for all $n \geq n_0$.

To establish this, we need to find suitable values for c and n_0.
We already are given constants c', c'', n_0', and n_0''.
Note that $f_1(n)f_2(n) \leq c'g_1(n)\, c''g_2(n) = c'\, c''\, g_1(n)g_2(n)$,
but only if $n \geq n_0'$ and $n \geq n_0''$.
So $n \geq n_0$ should imply $n \geq n_0'$ and $n \geq n_0''$.

Example

Theorem

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Proof

By definition there exist positive constants c' and n_0' such that $f_1(n) \le c' g_1(n)$ for all $n \ge n_0'$. Similarly, there exist positive constants c'' and n_0'' such that $f_2(n) \le c'' g_2(n)$ for all $n \ge n_0''$. We need to show that there exist positive constants c and n_0 such that

$f_1(n)f_2(n) \le c\, g_1(n)g_2(n)$ for all $n \ge n_0$.

Let $n_0 = \max(n_0', n_0'')$ and $c = c' c''$. Then, for all $n \ge n_0$ (which implies $n \ge n_0'$ and $n \ge n_0''$), $f_1(n)f_2(n) \le c' g_1(n)\, c'' g_2(n) = c' c'' g_1(n)\, g_2(n) = c\, g_1(n)\, g_2(n)$.

We have $f_1(n)f_2(n) \le c\, g_1(n)\, g_2(n)$ for all $n \ge n_0$.

Thus, $f_1(n)f_2(n) = O(g_1(n)g_2(n))$. □

Example

Theorem

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Proof

By definition there exist positive constants c' and n_0' such that $f_1(n) \leq c' \, g_1(n)$ for all $n \geq n_0'$. Similarly, there exist positive constants c'' and n_0'' such that $f_2(n) \leq c'' \, g_2(n)$ for all $n \geq n_0''$. We need to show that there exist positive constants c and n_0 such that

$f_1(n)f_2(n) \leq c \, g_1(n)g_2(n)$ for all $n \geq n_0$.

Let $n_0 = \max(n_0', n_0'')$ and $c = c' c''$. Then, for all $n \geq n_0$ (which implies $n \geq n_0'$ and $n \geq n_0''$), $f_1(n)f_2(n) \leq c' \, g_1(n) \, c'' \, g_2(n) = c' \, c'' \, g_1(n) \, g_2(n) = c \, g_1(n) \, g_2(n)$.

We have $f_1(n)f_2(n) \leq c \, g_1(n) \, g_2(n)$ for all $n \geq n_0$.

Thus, $f_1(n)f_2(n) = O(g_1(n)g_2(n))$. □

Overview

Basic Proving Techniques

1. Forward-backward method
2. Mathematical induction
3. Case analysis
4. Proof by contradiction

Mathematical Induction

An introduction to induction…

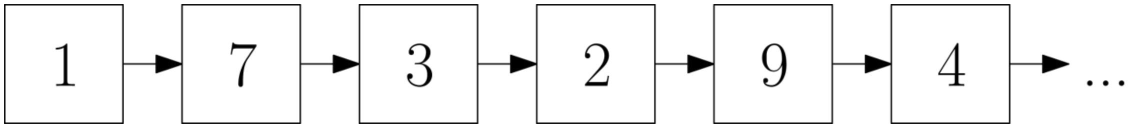
Recurring structures

Many algorithms and (data)structures have recurrences

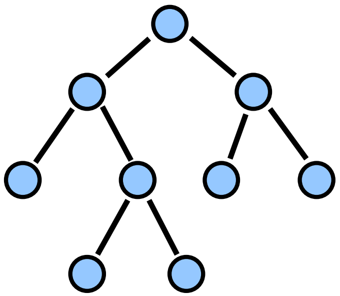
- Loop

```
sum = 0
for i = 1 to A.length
    do  sum = sum + A[i]
```

- List



- Tree



The Idea

The Base Idea of Induction

$$P(1) \to P(2) \to P(3) \to \dots \to P(n)$$

Base Case

One (or more) very simple cases that we can trivially proof.

Induction Hypothesis

The statement that we want to prove (for any n).

Induction Step

Prove that if the statement holds for a small instance, it must also hold for a larger instance.

Usage

When to use?

- Whenever you need to prove something is true for all values of n.
  - (or all values $\geq\ x$)
  - Infinite possibilities!

- When there is a clear structure in the problem (e.g., trees)
  - We will talk more about this later

Basic Example

Theorem

If n dominos are placed in a row and I push the first; they all fall.

Proof:

We use induction on n.

Base Case $(n = 1)$:

If there is only 1 domino, it must also be the first.

I will push the first over, so trivially they all fall.

Basic Example

Proof:

We use induction on n.

Base Case ($n = 1$):

If there is only 1 domino, it must also be the first.

I will push the first over, so trivially they all fall and the IH holds.

Induction Hypothesis:

If n dominos are placed in a row and I push the first; they all fall.

Induction Step:

Assume the IH holds for n dominos.

If there were $n + 1$ dominos in a row, the first n form a row of length n.

By IH the first n dominos will all fall. As all n dominos fall, so must the n^{th} domino. If the n^{th} domino falls, then it will tip over the $n + 1^{\text{th}}$. The first n dominos fall over and the $n + 1^{\text{th}}$ domino also falls over.

So all $n + 1$ dominos fall over. Thus, the IH holds.

Basic Example

Proof:

We use induction on n.

Base Case ($n = 1$):

If there is only 1 domino, it must also be the first.

I will push the first over, so trivially they all fall and the IH holds.

Induction Hypothesis:

If n dominos are placed in a row and I push the first; they all fall.

Induction Step:

Assume the IH holds for n dominos.

If there were $n+1$ dominos in a row, the first n form a row of length n.

By IH the first n dominos will all fall. As all n dominos fall, so must the n^{th} domino. If the n^{th} domino falls, then it will tip over the $n+1^{\text{th}}$. The first n dominos fall over and the $n+1^{\text{th}}$ domino also falls over.

So all $n+1$ dominos fall over. Thus, the IH holds.

Example

Theorem

For all positive integers n, $3^n - 1$ is even.

Proof:

We use induction on n.

Base Case (n = 1): $3^1 - 1 = 2$, which is indeed even.

IH: $3^n - 1$ is even.

Induction Step (n >= 1):

Assume that $3^n - 1$ is even. (IH)

We need to show that $3^{n+1} - 1$ is even.

We have: $3^{n+1} - 1 = 3 * 3^n - 1 = (2 * 3^n) + (3^n - 1)$.

A multiplication with an even number is always even $(2 * 3^n)$.

By IH, $(3^n - 1)$ is also even. The sum of two even numbers is also even.

Thus, $3^{n+1} - 1$ must be even. The IH holds. □

Practice 1

Theorem

For all positive integers n, $\sum_{k=1}^{n} k = n(n+1)/2$

Practice 1

Proof:

We use induction on n.

Base case ($n = 1$):

$$\sum_{k=1}^{n} k = \sum_{k=1}^{1} k = 1 = \frac{1(1+1)}{2} = \frac{n(n+1)}{2}.$$

As both values equate to the same the IH holds.

IH: $\sum_{k=1}^{n} k = n(n+1)/2$

Induction Step ($n \geq 1$):

Suppose that $\sum_{k=1}^{n} k = n(n+1)/2$. (IH)

We need to show that $\sum_{k=1}^{n+1} k = (n+1)(n+2)/2$.

We have: $\sum_{k=1}^{n+1} k = \sum_{k=1}^{n} k + (n+1) = \frac{n(n+1)}{2} + (n+1)$ (by IH)

$$= \frac{n(n+1)+2(n+1)}{2} = \frac{(n+1)(n+2)}{2}.$$

Thus the IH holds and it follows by induction that $\sum_{k=1}^{n} k = n(n+1)/2$ for all positive integers n. □

TU/e technische universiteit eindhoven

Practice 2

Theorem

For every integer $n \geq 5$, $2^n > n^2$

Practice 2

Proof:

We use induction on n.

Base case ($n = 5$): $2^n = 2^5 = 32 > 25 = 5^2 = n^2$

IH: $2^n > n^2$

Induction Step ($n \geq 5$):

Suppose that $2^n > n^2$ (IH).

We need to show that $2^{n+1} > (n+1)^2$.

We have:

$$2^{n+1} = 2 * 2^n > 2 * n^2 \text{ (by IH)}$$

So it is sufficient to show that $2 * n^2 \geq (n+1)^2 = n^2 + 2n + 1$ for $n \geq 5$. This can be simplified to $n^2 - 2n - 1 \geq 0$ or $(n-1)^2 \geq 2$.

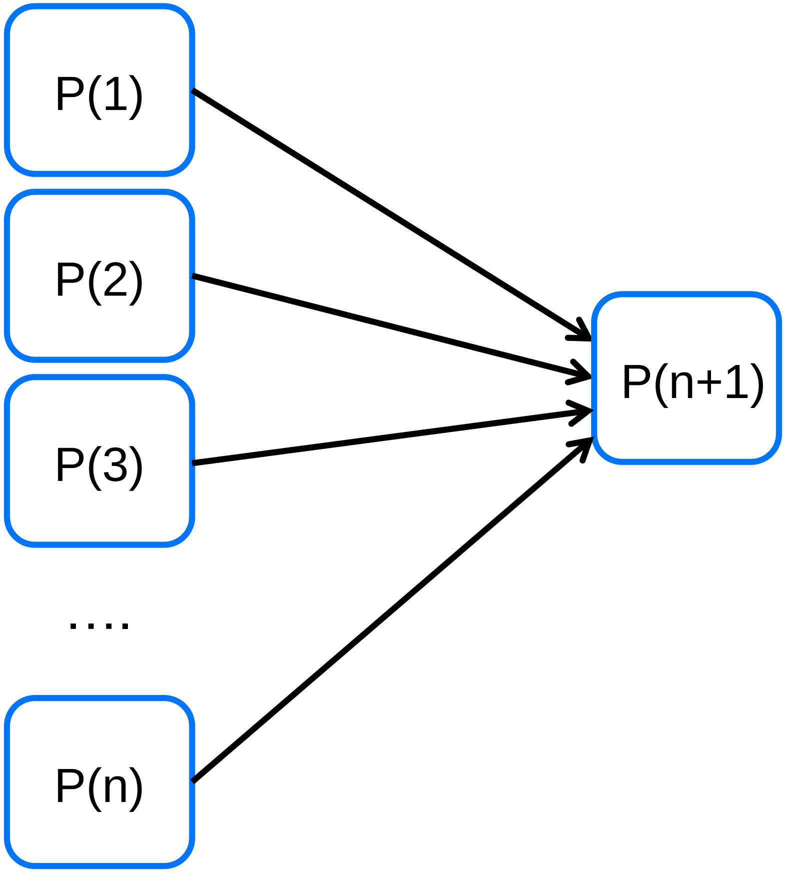
This is clearly true for $n \geq 5$.

So it follows by induction that $2^n > n^2$ for $n \geq 5$. □

Strong Induction

$P(1)$

$P(1) \wedge \ldots \wedge P(n) \Rightarrow P(n+1)$



Strong Induction

Theorem (Nim)

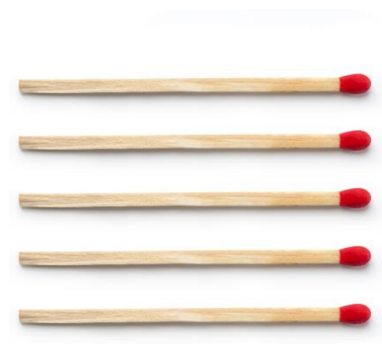
If the two piles contain the same number of matches at the start of the game, then the second player can always win.

Strong Induction

Theorem (Nim)

If the two piles contain the same number of matches at the start of the game, then the second player can always win.

Player 1



Player 2

Strong Induction

Theorem (Nim)

If the two piles contain the same number of matches at the start of the game, then the second player can always win.

Proof

We use strong induction on n.

IH: "If the two piles both contain n matches at the … always win"

Base Case ($n = 1$):

The first player only has one option, emptying one of the piles. The second player can empty the second pile and, thus, wins.

Induction Step ($n \geq 1$):

Assume the second player can always win if there are two piles with k matchsticks each, for $1 \leq k \leq n$. (IH)

We prove the IH for two piles with $n + 1$ matchsticks each. Assume w.l.o.g. that player 1 takes $m \geq 1$ matchsticks from the first pile.

The second player can then always take m matchsticks from the other pile. We are now left with two piles with both $n + 1 - m \leq n$ matchsticks. By IH, player two can always win from this setting. □

Strong Induction

Theorem (Nim)

If the two piles contain the same number of matches at the start of the game, then the second player can always win.

Proof

We use strong induction on n.

IH: **"**If the two piles both contain n matches at the … always win"

Base Case ($n = 1$):

The first player only has one option, emptying one of the piles. The second player can empty the second pile and, thus, wins.

Induction Step ($n \geq 1$):

Assume the second player can always win if there are two piles with k matchsticks each, for $1 \leq k \leq n$. (IH)

We prove the IH for two piles with $n + 1$ matchsticks each. Assume w.l.o.g. that player 1 takes $m \geq 1$ matchsticks from the first pile.

The second player can then always take m matchsticks from the other pile. We are now left with two piles with both $n + 1 - m \leq n$ matchsticks. By IH, player two can always win from this setting. □

Practice 1

Theorem

It takes $n - 1$ breaks to break a chocolate bar with $n \geq 1$ squares into individual squares

Proof:

We use strong induction on n.

Base case $(n = 1)$:

It's just 1 square, so $1 - 1 = 0$ breaks is trivially correct.

IH: …

Induction Step $(n \geq 2)$:

Consider a chocolate bar with n squares.

Suppose the chocolate bar is broken into 2 pieces of a and b squares, where $1 \leq a, b < n$ and $a + b = n$.

By the IH we need $a - 1$ breaks for the first part and $b - 1$ for the second.

Thus, we need $1 + (a - 1) + (b - 1) = a + b - 1 = n - 1$ breaks. □

Loop Invariant

An introduction to proving loops…

Loop Invariant

```
sum = 0
for i = 1 to A.length
    do  sum = sum + A[i]
```

What do we want?

How do we prove something is true at the end?

What do we really know?

Loop Invariant

$i = 1$ →

$i = 1$ →

```
sum = 0
for i = 1 to A.length
    do  sum = sum + A[i]
```

What do we really know?

At the start:

Sum $=$ 0

$=$ Sum first 0 elements

Loop Invariant

$i = 2$ →
$i = 2$ →

sum = 0
for i = 1 **to** A.length
    do sum = sum + A[i]

What do we really know?

At the start:
Sum = 0
= Sum first 0 elements

After first iteration:
Sum = $0 + A[1]$
= Sum first 1 element

Loop Invariant

$i = 3$ →

```
sum = 0
for i = 1 to A.length
    do  sum = sum + A[i]
```

What do we really know?

At the start:

Sum = 0

= Sum first 0 elements

After first iteration:

Sum = $0 + A[1]$

= Sum first 1 element

After second iteration:

Sum = $0 + A[1] + A[2]$

= Sum first 2 elements

Loop Invariant

Loop Invariants replicate the chain of logical derivations

At the start:
Sum $= 0$
$=$ Sum first 0 elements

After first iteration:
Sum $= 0 + A[1]$
$=$ Sum first 1 element

After second iteration:
Sum $= 0 + A[1] + A[2]$
$=$ Sum first 2 elements

To prove a claim is true at the end, we show…

- …it is true at the start
- …if it is true at the start of a *random* iteration i,
  it is still true at the start of the next iteration $i + 1$
- …the claim is true at the end of the loop.

Loop Invariant

Loop Invariants replicate the chain of logical derivations

At the start:

Sum $= 0$

$=$ Sum first 0 elements

After first iteration:

Sum $= 0 + A[1]$

$=$ Sum first 1 element

After second iteration:

Sum $= 0 + A[1] + A[2]$

$=$ Sum first 2 elements

To prove a statement is true at the end, we need..

- Invariant (What remains true)
- Initialization (Starting conditions)
- Maintenance (Making sure it remains true)
- Termination (Ending conditions)

Basic Example

```
sum = 0
for i = 1 to A.length
    do  sum = sum + A[i]
```

Invariant

At the start of iteration i, sum contains the sum of $A[1..i-1]$.

Initialization

At the start of the loop $i = 1$ and $sum = 0$.

For the loop invariant to hold, sum must contains the sum of $A[1..0] = \emptyset$.

The sum of no elements is trivially 0.

So sum is correctly set to 0.

Basic Example

```
sum = 0
for i = 1 to A.length
    do  sum = sum + A[i]
```

Invariant

At the start of iteration i, sum contains the sum of $A[1..i-1]$.

Maintenance

At the start of iteration i, by the loop invariant sum contains the sum of $A[1..i-1]$. In iteration i, sum is increased by $A[i]$. So sum is the sum of elements $A[1..i-1] + A[i]$ = the sum of elements $A[1..i] = A[1..i+1-1]$. Thus the invariant will be maintained.

Basic Example

```
sum = 0
for i = 1 to A.length
    do  sum = sum + A[i]
```

Invariant

At the start of iteration i, sum contains the sum of $A[1..i-1]$.

Termination

The loop terminates when $i > A.length$, so $i = A.length + 1$.
By the loop invariant we know that sum contains the sum
of $A[1..A.length + 1 - 1] = A[1..A.length]$.
This is exactly what we wanted to compute.

Tips and Tricks

Finding a Loop Invariant

What do you want to know at the end?

Loop Invariant (generally) proves something that is growing

I.e., $A[1..i-1]$

Think about a specific iteration.

What do you **know**.

Which indices do you need at the start and end.

Tips and Tricks

```
sum = 0
for i = 1 to A.length
    do  sum = sum + A[i]
```

What do you want to know at the end?

I want to show that sum contains the sum of all elements in A.
So I need to know something about sum and A.

I now know the loop invariant should contain A and sum.

Tips and Tricks

```
sum = 0
for i = 1 to A.length
    do  sum = sum + A[i]
```

Loop Invariant (generally) proves something that is growing

What does my loop do?

In each iteration I know something more about the array A.

I'm going through the loop starting at the beginning of A.

So **perhaps I can do** something like $A[1..i-1]$.

Tips and Tricks

```
sum = 0
for i = 1 to A.length
    do  sum = sum + A[i]
```

Think about a specific iteration

Let's think about a random iteration 5. What do I know?

I will have seen items 1, 2, 3 and 4.

And assuming my program works, sum should be their sum.

The rest of the array I do not yet know.

It seems that:

At the start of iteration 5, sum contains the sum of A[1..4]

Tips and Tricks

Finding a Loop Invariant

What do you want to know at the end?

Loop Invariant (generally) proves something that is growing

I.e., $A[1..i-1]$

Think about a specific iteration.

What do you **know**.

Which indices do you need at the start and end.

Loop Invariant

At the start of iteration i, sum contains the sum of $A[1..i-1]$.

Notes

Invariant

Be careful with i or $i-1$.

Maintenance

Use loop invariant at start of loop to prove loop invariant at start of next loop.

Termination

Requires loop invariant

At which value does the loop terminate?

1. **for** i = n **downto** 1
2. **do** *stuff*

$i = 0$

1. **while** $x^2 < n$
2. **do** x = x + 1

$x = \lceil \sqrt{n} \rceil$

1. **while** $x \leq n$
2. **do** x = x + 2

$x = n+1$ or $x = n+2$

Termination values?

Practice

Prove **using loop invariant** that… $y = c$ after the loop.

```
x = c
y = 0
while x > 0
    do  x--;
        y++;
```

Practice

Loop Invariant:

At the start of iteration, $x + y = c$.

Initialization:

At the start, $\mathrm{x} = \mathrm{c}$ and $y = 0$,

so $x + y = c$ which is correct.

```
x = c
y = 0
while x > 0
    do  x--;
        y++;
```

Practice

```
x = c
y = 0
while x > 0
    do x--;
       y++;
```

Maintenance:

Assume that the loop invariance holds at the start of loop i. Then $x + y = c$. Let x' and y' be the Values of x and y at the end of the loop.

We know $x' = x - 1$ and $y' = y + 1$.

But then at the end of the loop it holds that

$x' + y' = x - 1 + y + 1 = x + y = c$.

Thus the loop invariant is maintained.

Termination:

At termination, $\mathrm{x} = 0$. By the loop invariant we know $x + y = c$.

Combining both statements gives $y = c$.

Loop Invariant or Induction??

Differences and similarities…

(Dis)similarities

Loop Invariant

- … is a special kind of induction
- … used to prove loops (…obviously…)
- … has a termination condition

Induction

- Induction can have multiple base cases
- There are other forms of induction (e.g, structural induction)

Always

- Induction Hypothesis is like Loop Invariant
- Maintenance (/Step), assumes LI (/IH) and proves it for next.

Overview

Basic Proving Techniques

1. Forward-backward method

2. Mathematical induction

3. Case analysis

4. Proof by contradiction

Case Analysis

A) Suitcase   B) Bookcase   C) In case …

Case Analysis

Case analysis

Prove the theorem by considering a small number of cases

Let's prove $P \Rightarrow Q$

$P_1 \Rightarrow Q$

$P_2 \Rightarrow Q$

$P_3 \Rightarrow Q$

$P \Rightarrow P_1 \lor P_2 \lor P_3$

P_1, P_2, and P_3 describe the different cases

Don't forget to prove: P_1 or P_2 or P_3 (one of the cases must hold)!

Usage

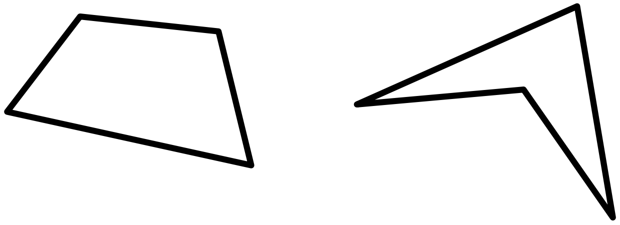
When to use?

Generally useful for a “for all”-quantifier

- Can be broken down into a small number of configurations

Examples

- An integer is odd or even
- An integer is positive, negative, or zero
- $x \leq y$ or $y < x$
- A quadrilateral is convex or not



Basic Example

Theorem

I do not like any teletubby.

Let’s prove $P \Rightarrow Q$

Basic Example

Theorem

For any teletubby, I do not like it.

Let’s prove $P \Rightarrow Q$

Case 1 (Tinky-Winky):

Is purple.

I don’t like purple.

Thus, I do not like Tinky-Winky.

$P_1 \Rightarrow Q$

Basic Example

Theorem

For any teletubby, I do not like it.

Let's prove $P \Rightarrow Q$

Case 2 (Po):

$P_2 \Rightarrow Q$

Has a circle on his head.

I don't like circles.

Thus, I do not like Po.

Basic Example

Theorem

For any teletubby, I do not like it.

Let's prove $P \Rightarrow Q$

Case 1 (Tinky-Winky):

$P_1 \Rightarrow Q$

…

…

Case 4 (Dispy):

$P_4 \Rightarrow Q$

…

As any teletubby must fall into these categories (by definition), I do not like any teletubby.

$P \Rightarrow P_1 \vee P_2 \vee P_3 \vee P_4$

Example

Theorem

For any integer x, $x(x+1)$ is even

Proof

Right now we know nothing about x, which makes it hard to prove that $x(x+1)$ is even (we have nothing to work with).

What happens if x is odd?

In that case $(x+1)$ is even, and hence the multiplication must be even.

What happens if x is even?

Doesn’t really matter, the multiplication will be even.

Example

Theorem

For any integer x, $x(x+1)$ is even

Proof

We consider two cases:

Case (1): x is odd

Then there exists an integer k such that $x = 2k+1$. Hence,

$x(x+1) = (2k+1)(2k+2) = 2(2k+1)(k+1)$.

Thus, $x(x+1)$ is even.

Case (2): x is even

Then there exists an integer k such that $x = 2k$. Hence,

$x(x+1) = 2k(2k+1) = 2(2k^2+k)$.

Thus, $x(x+1)$ is even.

Since an integer is either odd or even, this concludes the proof. □

Often (incorrectly) omitted

Practice 1

Algorithm LargeEven(A)

large = -∞

for i = 1 **to** n

  if A[i] > large **and** A[i] is even

    then large = A[i]

Loop Invariant

At the start of iteration i, *large* is the biggest even value in A[1..i-1] (or -∞ if there are no even numbers in A[1..i-1]).

Maintenance

We assume the loop invariant (LI) holds at the start of iteration i.

Then *large* is the biggest even value in A[1..i-1].

…..

So we have proven the LI is also true at the start of iteration i+1.

Practice 1

Algorithm LargeEven(*A*)

large = -∞

for i = 1 **to** n

    if A[i] > large **and** A[i] is even

        then large = A[i]

Assumption

At the start of iteration i, *large* is the biggest even value in A[1..i-1] (or -∞ if there are no even numbers in A[1..i-1])..

Claim

At the start of iteration i+1, *large* is the biggest even value in A[1..i] (or -∞ if there are no even numbers in A[1..i])...

Practice 1

Assump: At the start of iteration i, *large* is the biggest even value in A[1..i-1].

Claim: At the start of iteration i+1, *large* is the biggest even value in A[1..i].

Proof:

At the start of iteration i+1, *large* is the biggest even number in A[1..i].

There are three cases.

Case 1) A[i] > *large* and A[i] is odd

As A[i] is odd is can not change the value of the biggest even number. *Large* was the biggest even number in A[1..i-1], so *large* is also the biggest even number in A[1..i-1] ∪ A[i] = A[1..i].

Case 2) A[i] ≤ large

The biggest even number in A[1..i-1] is *large*. As A[i] ≤ *large,* the biggest even number in A[1..i-1] ∪ A[i] = A[1..i] is still *large*.

Practice 1

Assump: At the start of iteration i, *large* is the biggest even value in A[1..i-1].

Claim: At the start of iteration i+1, *large* is the biggest even value in A[1..i].

Proof:

At the start of iteration i+1, *large* is the biggest even number in A[1..i].

There are three cases.

Case 3) A[i] > *large* and A[i] is even

large is the biggest even number in A[1..i-1], and A[i] is even bigger than *large*. Then A[i] is bigger than any number in A[1..i-1]. So A[i] is the biggest number in A[1..i].

large is changed to A[i], so *large* now holds the biggest number in A[1..i].

As it must either hold that A[i] > *large* or A[i] ≤ *large*

and also either that A[i] is even or odd,

these cases cover all possibilities.

Practice 2

Theorem

Among any 6 people there are 3 mutual friends or 3 mutual strangers.

Overview

Basic Proving Techniques

1. Forward-backward method

2. Mathematical induction

3. Case analysis

4. Proof by contradiction

Proof by Contradiction

It's elementary…

Contradiction

"When you have eliminated the impossible, whatever remains, however improbable, must be the truth"

Proof by Contradiction

- Assume the negation and show that "it is impossible"
- To prove Q:
  - Assume ¬Q and derive contradiction (false) by forward reasoning
- To prove ¬Q:
  - Assume Q and derive contradiction…
- Very powerful technique!

Usage

When to use?

- Useful when the negation of the statement is easier to work with
- Useful when the negation as a premise gives more information
  - E.g. when the negation has a “there exists”-quantifier
- Always try this method if you’re stuck!

Basic Example

Theorem

I never leave my house without my Ferrari

Proof

For sake of contradiction,

assume I did leave my house without my Ferrari.

But then I would not look cool (by Lemma X).

I *am* very cool 😎 (by Axiom Y).

Contradiction, thus the assumption must be false.

Hence, I never leave my house without my Ferrari.

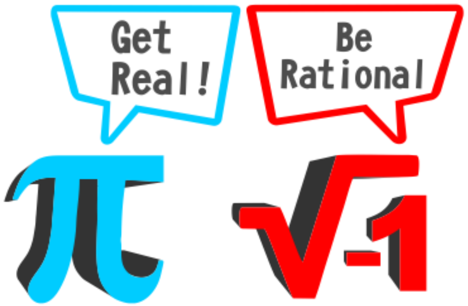
Rational Numbers

Definition

A number x is rational if there exists integers a and b such that $x = a / b$

Examples

- 6, $\frac{1}{3}$, and $-\frac{5}{8}$ are rational
- π and e are irrational (not rational)



Example

Theorem

$\sqrt{2}$ is irrational

Proof

We should prove there exist no integers $a,\ b$ *such that* $\sqrt{2} = a/b$*.*

What can we do with that? Not sure…

How about a proof by contradiction?

That means we assume that such a and b ***do*** *exist.*

What is wrong with that?

Example

Theorem

$\sqrt{2}$ is irrational

Proof

For the sake of contradiction, assume there exist integers a and b such that $\sqrt{2} = a/b$. Without loss of generality we assume $b > 0$ (why?).

Square both sides and rewrite to obtain $2b^2 = a^2$.

This means that a^2 is even and thus a is even.

Hence there exists a k such that $a = 2k$.

But then $2b^2 = a^2 = (2k)^2$ or $b^2 = 2k^2$, and thus b is also even.

So both a and b are even.

If $b = 2m$, then $a/b = 2k/2m = k/m = \sqrt{2}$.

*And k and m are **smaller** integers. The same argument for k and m gives even smaller integers. This cannot go on forever!*

Example

Theorem

$\sqrt{2}$ is irrational

Proof

For the sake of contradiction, let a and b be the smallest positive integers such that $\sqrt{2} = a/b$. Square both sides and rewrite to obtain $2b^2 = a^2$. This means that a^2 is even and thus a is even.

Hence there exists a k such that $a = 2k$.

But then $2b^2 = a^2 = 4k^2$ or $b^2 = 2k^2$, thus there exists an integer m such that $b = 2m$. We get that $\mathrm{a/b} = 2k/2m = k/m = \sqrt{2}$.

But k and m are smaller than a and b, which contradicts the assumption that a and b are smallest positive integers such that $\sqrt{2} = a/b$. Thus, we find a contradiction and our assumption must be false. Thus, there exists no a and b such that $\sqrt{2} = a/b$ and it must be that $\sqrt{2}$ is irrational. □

Practice

Theorem: $n^2 \log n \neq O(n^2)$

Proof

For the sake of contradiction, assume that there exist positive constants c and n_0 such that $n^2 \log n \leq c\, n^2$ for all $n \geq n_0$.
By dividing both sides by n^2, we obtain that $\log n \leq c$ for all $n \geq n_0$.

This is false for $n = \max(2^{c+1}, n_0)$, since then
$\log n \geq \log 2^{c+1} = c + 1 > c$.
This contradicts that $\log n \leq c$ for all $n \geq n_0$.
Thus, $n^2 \log n \neq O(n^2)$. □

Usually it is sufficient to say that the function $f(n)$ *(in this case:* $\log n$*) is unbounded. This automatically implies that, for any constant* c*, there exists an* n *large enough such that* $f(n) > c$*.*

Practice

Theorem

$n^2 \log n \neq O(n^2)$

Proof

For the sake of contradiction, assume that there exist positive constants c and n_0 such that $n^2 \log n \leq c\, n^2$ for all $n \geq n_0$.

By dividing both sides by n^2, we obtain that $\log n \leq c$ for all $n \geq n_0$.

As $\lim_{n \to \infty} \log n = \infty$, there can not exist a constant c that is always larger. Thus, the assumption must be false.□

Common Errors

As can be seen in Figure 4 this is true…

Find the correct proofs!

Correct or not? - Test

Theorem

In every set of $n \geq 1$ horses, all horses have the same color

Proof

We use induction on n.

Base case $(n = 1)$:

There is only one horse, so it must be true. The IH holds.

IH: In every set of $n \geq 1$ horses, all horses have the same color.

Step $(n \geq 1)$:

Suppose that in every set of n horses, all horses have the same color (IH).

We need to show that any set of $n + 1$ horses share the same color.

By the IH, the first n horses have the same color. Similarly, by the IH, the last n horses have the same color. Thus all horses have the same color. □

Base Case not reached (NOT correct)

Theorem

In every set of $n \geq 1$ horses, all horses have the same color

Proof

We use induction on n.

Base case ($n = 1$):

There is only one horse, so it must be true. The IH holds.

IH: In every set of $n \geq 1$ horses, all horses have the same color.

Step ($n \geq 1$):

Suppose that in every set of n horses, all horses have the same color (IH).

We need to show that any set of $n + 1$ horses share the same color.

By the IH, the first n horses have the same color. Similarly, by the IH, the last n horses have the same color. Thus all horses have the same color. $\square$

Correct or not?

Theorem

In a sorted list duplicates are always next to each other.

Proof

If we look at a sorted list, for example [1,3,3,4,7] both values of 3 are next to each other in the list. Clearly, this list is sorted and both values are next to each other.

Thus, duplicates must be next to each other in a sorted list.

Proof by Example (NOT correct)

Theorem

In a sorted list duplicates are always next to each other.

Proof

If we look at a sorted list, for example [1,3,3,4,7] both values of 3 are next to each other in the list. Clearly, this list is sorted and both values are next to each other.

Thus, duplicates must be next to each other in a sorted list.

Correct or not?

Theorem

For any integer x, $x(x+1)$ is even

Proof

We consider two cases:

Case (1): x is odd

Then there exists an integer k such that $x = 2k + 1$. Hence,

$x(x+1) = (2k+1)(2k+2) = 2(2k+1)(k+1)$.

Thus, $x(x+1)$ is even.

Case (2): x is even

Then there exists an integer k such that $x = 2k$. Hence,

$x(x+1) = 2k(2k+1) = 2(2k^2 + k)$.

Thus, $x(x+1)$ is even.

Finishing proofs (NOT correct)

Theorem

For any integer x, $x(x+1)$ is even

Proof

We consider two cases:

Case (1): x is odd

Then there exists an integer k such that $x = 2k+1$. Hence,

$x(x+1) = (2k+1)(2k+2) = 2(2k+1)(k+1)$.

Thus, $x(x+1)$ is even.

Case (2): x is even

Then there exists an integer k such that $x = 2k$. Hence,

$x(x+1) = 2k(2k+1) = 2(2k^2+k)$.

Thus, $x(x+1)$ is even.

Proof Techniques Summary

Basic Proving Techniques

1. Forward-backward method

2. Mathematical induction
3. Case analysis

4. Proof by contradiction