Mathematics for Computer Science

Prof. dr.hab. Viorel Bostan

Technical University of Moldova

viorel.bostan@adm.utm.md

Lecture 5

Picture of the day

Another picture of the day

Previous Lecture Outline

- Inference Rules (Deductions);
- Predicate Logic (1st order logic): Predicates, $\forall$ and $\exists$;
- Order of quantifiers;
- Negating a quantifier;
- Well Ordering Principle

Well Ordering Principle

Every **nonempty** set of **nonnegative integers** has a smallest element.

Induction Principle

Another important "proving" principle is the **Induction Principle**.

Induction Principle

Let $P(n)$ be a predicate. IF

- **Base case.** $P(0)$ is true,
- **Inductive case.** $P(n)$ IMPLIES $P(n+1)$ for all nonnegative integers n,

THEN
$P(m)$ is true for all nonnegative integers m.

Induction Rule

$$\frac{P(0), \quad \forall n \in \mathbb{N} \quad P(n) \to P(n+1)}{\forall m \in \mathbb{N} \quad P(m)}$$

Theorem

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}, \quad \forall n \in \mathbb{N}. \tag{1}$$

Proof.

Let predicate $P(n)$ be formula (1).

Base case. Need to show that $P(0)$ is true.
$P(0)$ means "sum of zero terms is $0 = \frac{0\cdot(0+1)}{2}$." This is true!

Inductive case. Suppose that $P(n)$ is true, i.e. (1) holds for an arbitrary n.
Need to prove that $P(n+1)$ is also true, i.e. need to prove:

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

Theorem

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}, \quad \forall n \in \mathbb{N}. \tag{1}$$

Proof contd.

From (1) it follows (by adding to both sides $n+1$) that

$$\begin{aligned} 1 + 2 + 3 + \cdots + n + (n+1) &= \tfrac{n(n+1)}{2} + (n+1) \\ &= \tfrac{n(n+1)+2(n+1)}{2} \\ &= \tfrac{n^2+3n+2}{2} \\ &= \tfrac{(n+1)(n+2)}{2}. \end{aligned}$$

Thus, we proved that $P(n+1)$ is true.
By Induction Principle, formula (1) holds for all $n \in \mathbb{N}$. □

Induction Principle: Template for proofs

1. State that the proof is by Induction Principle (or simply, by induction).
2. Define the predicate $P(n)$. This is called the induction hypothesis.
3. **Base Case.** Prove that $P(0)$ is true.
4. **Inductive Case.** Assume that $P(n)$ is true and then use this assumption to prove that $P(n+1)$ is also true. Once this is done we will have the following implications holding:
$$P(0) \to P(1),\ P(1) \to P(2),\ P(2) \to P(3),\ P(3) \to P(4), \ldots,$$
5. Invoke induction to conclude that predicate $P(n)$ is true $\forall n \in \mathbb{N}$.

Induction Principle

Also, the predicate might start not from 0, but say from 5:

$$P(n) \text{ is true for all } n \geqslant 5.$$

In that case, the base case will be not "$P(0)$ is true", but "$P(5)$ is true".

Theorem

$$1 + 3 + 5 + \cdots + (2n - 1) = n^2, \quad \forall n \in \mathbb{N}. \tag{1}$$

Proof.

Proof by induction. Let predicate $P(n)$ be (1).
Base case. Obviously $P(0)$ is true.

Inductive case. Suppose that $P(n)$ is true, i.e. (1) holds for n.
Need to prove that $P(n+1)$ is also true, i.e. need to prove:

$$1 + 3 + 5 + \cdots + (2n - 1) + (2n + 1) = (n + 1)^2.$$

Start from left side:

$$\underbrace{1 + 3 + 5 + \cdots + (2n - 1)}_{=n^2} + (2n + 1) = n^2 + 2n + 1 = (n + 1)^2.$$

Thus, if $P(n)$ is true, then $P(n+1)$ is also true. By Induction Principle QED □

Theorem

For any $n \in \mathbb{N}$, 3 divides $n^3 - n$ (written as $3|(n^3 - n)$).

Proof.

Proof by induction. Let predicate $P(n)$ be 3 divides $n^3 - n$.
Base case. $P(0)$ means: 3 divides $0 - 0$. True!

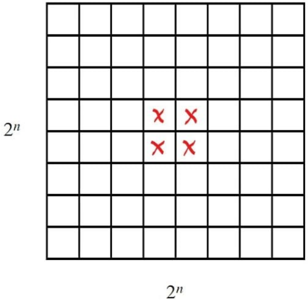
Inductive case. Suppose that $P(k)$ is true, i.e. 3 divides $k^3 - k$.
Need to prove that $P(k+1)$ is also true, i.e. need to prove that 3 divides $(k+1)^3 - (k+1)$.
Start from left side:

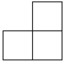
$$\begin{aligned}(k+1)^3 - (k+1) &= k^3 + 3k^2 + 3k + 1 - k - 1 = k^3 + 3k^2 + 2k = \ldots \\ &= \underbrace{k^3 - k}_{\text{multiple of 3 by } P(k)} + \underbrace{3k^2 + 3k}_{\text{multiple of 3}}\end{aligned}$$

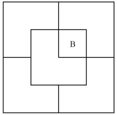
Thus, if $P(k)$ is true, then $P(k+1)$ is also true. $\square$

Courtyard Example

Let a big courtyard with dimensions $2^n \times 2^n$ and suppose that one of the central squares must be occupied by a statue:







Suppose that the tiles for the courtyard are of L shape. Want to tile the courtyard.

For $n = 2$ it is possible.

Problem

Is there a way to tile a $2^n \times 2^n$ courtyard with L-shaped tiles around a statue in the center?

Courtyard Example

Theorem

For all $n > 0$ there exists a tiling of a $2^n \times 2^n$ courtyard with a statue in a central square.

Proof.

Proof by induction.

Base case: $P(0)$ is true because statue fills the whole courtyard.
Inductive step: Assume that there is a tiling of a $2^n \times 2^n$ courtyard with statue in the center for some $n > 0$.
We must prove that there is a way to tile a $2^{n+1} \times 2^{n+1}$ courtyard with statue in the center
...
Trouble! It is not easy to go from $2^n \times 2^n$ to $2^{n+1} \times 2^{n+1}$.

Useful Rule

If you can't prove something, try to prove something grander!

Courtyard Example

Theorem

For all $n > 0$, there exists a tiling of a $2^n \times 2^n$ courtyard with a statue in any location.

Proof.

Proof by induction.

Base case: $P(0)$ is true because statue fills the whole courtyard.

Inductive step: Assume that there is a tiling of a $2^n \times 2^n$ courtyard with statue in any location for some $n > 0$.

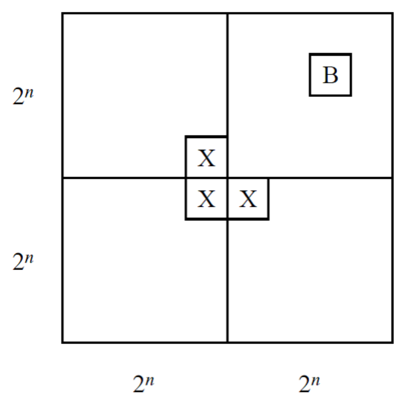
We must prove that there is a way to tile a $2^{n+1} \times 2^{n+1}$ courtyard with statue in any location.

Divide the $2^{n+1} \times 2^{n+1}$ courtyard into 4 quadrants, each of size $2^n \times 2^n$.

One quadrant will contain the statue in the desired location.

Courtyard Example

Proof contd.



Place a temporary statue in each of the three central squares lying outside this quadrant as shown on the left.

Each of the four quadrants $2^n \times 2^n$ can be tiled by the induction assumption.

Replacing the 3 temporary statues marked X with a single L-shaped tile completes the job.

This proves that $P(n) \to P(n+1)\ \forall n > 0$.

Thus, $P(n)$ is true $\forall n \in \mathbb{N}$, and the theorem follows. $\square$

Horse Example

Theorem (False Theorem)

All horses are the same color.

Theorem (False Theorem – reformulated)

In every set of $n \geqslant 1$ horses, all the horses are the same color.

Proof.

The proof is by induction on n. The induction hypothesis, $P(n)$, will be

$$P(n) : \text{In every set of } n \text{ horses, all are the same color.}$$

Base case: $P(1)$ is true, because in a set of horses of size 1, there's only one horse, and this horse is definitely the same color as itself.

Inductive Case. Assume that $P(k)$ is true for some $k \geq 1$.

That is, assume that in every set of k horses, all are the same color.

Proof contd.

Now consider a set of $k+1$ horses:

$$\underbrace{h_1, h_2, h_3, \ldots, h_{k-1}, h_k,}_{k \text{ horses same color}} h_{k+1}.$$

On the other hand,

$$h_1, \underbrace{h_2, h_3, \ldots, h_{k-1}, h_k, h_{k+1}}_{k \text{ horses same color}}.$$

So, h_1 is the same color as $h_2, \ldots, h_k$, and likewise h_{k+1} is the same color as $h_2, \ldots, h_k$.

Since h_1 and h_{k+1} are the same color as $h_2, \ldots, h_k$, horses h_1, h_2, h_3, $\ldots$, h_{k-1}, h_k, h_{k+1} must all be the same color, and so $P(k+1)$ is true.

Thus, $P(k)$ implies $P(k+1)$.

By induction principle, $P(n)$ holds for all $n \in \mathbb{N}$, in other words: all horses are having the same color. $\square$

What went wrong?

Actually, we have shown that

$$P(1), \quad P(2) \to P(3),\, P(3) \to P(4),\, P(4) \to P(5),\, \ldots$$

without showing $P(1) \to P(2)$.

And our proof cannot be applied to this particular case $P(1) \to P(2)$.

In fact, $P(1) \not\to P(2)$.

Strong Induction Principle

A useful variant of induction is called **Strong Induction**.

Strong induction is useful when a simple proof that the predicate holds for $n+1$ does not follow just from the fact that it holds for n,
but follows from the fact that it holds for other values $\leqslant n$.

Strong Induction Principle

Let $P(n)$ be a predicate. If

- **Base case.** $P(0)$ is true, and
- **Inductive case.** $P(0), P(1), P(2), \ldots, P(k)$ ALL IMPLY $P(k+1)$ for any $k \in \mathbb{N}$,
- THEN, $P(n)$ is true for all nonnegative integers n.

Theorem

Every integer greater than 1 *is a product of primes.*

Proof.

Proof by strong induction. Define predicate:

$$P(n) \ : \ n \text{ is a product of primes}, \quad \forall n \geqslant 2.$$

Base Case. $P(2)$ is true, since 2 is prime and it is a length 1 product.

Inductive Case. Suppose $k \geqslant 2$, and every integer from 2 to k is a product of primes. Need to show that $P(k+1)$ also holds, i.e. $k+1$ is also a product of primes.

Case 1: $k+1$ is a prime number. Thus, it is a product of length 1 of primes.
Case 2: $k+1$ is not a prime. Therefore, by definition

$$k+1 = q \cdot m, \quad \text{for some } q \text{ and } m,\ 2 \leqslant q, m \leqslant k$$

By strong induction hypotheses, q and m are products of primes.
Thus, $k+1 = q \cdot m$ is also a product of primes.
So, $P(k+1)$ is true, which completes the proof by strong induction. $\square$

Invariant Principle

Important use of induction in Computer Science involves proving that a program, algorithm or process preserves one or more desirable properties as it proceeds.

Definition

A property that is preserved through a series of operations or steps is called **invariant**.

Induction Principle is used to prove that a property is an invariant:

- Show that the property is true at the beginning (**base step**);
- Show that if it is true after n steps have been taken, it will also be true after step $n + 1$ (**inductive step**).

We can then use the induction principle to conclude that the property is indeed an invariant, namely, that it will always hold.

Invariants appear in systems that have a start state (starting configuration) and well-defined steps during which the system can change state.

Robot example

Example

Suppose there is a robot that can walk across diagonals on an infinite 2−dimensional grid.

The robot starts at position $(0, 0)$ and at each step it moves up or down by 1 unit vertically and left or right by 1 unit horizontally.

In this example, the state of the robot at any time can be specified by pair (x, y) that denotes the robot's position.

The start state is $(0, 0)$, since the robot starts at that position.

After the first step, the robot could be in states $(1, -1), (1, 1), (-1, 1), (-1, -1)$.

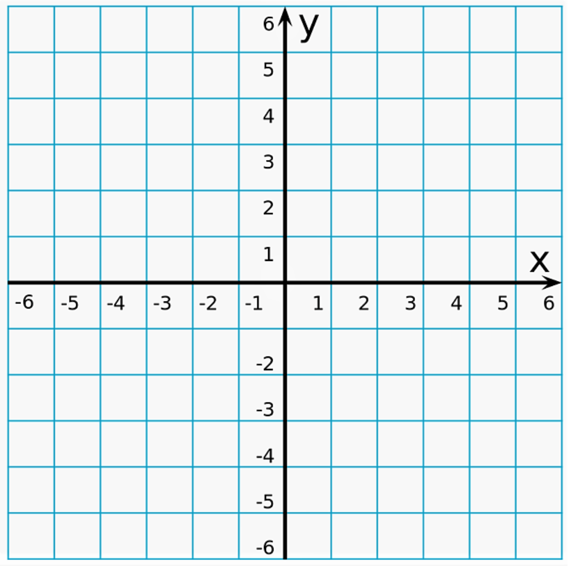
After the second step, the robot could be in states
$(0, 0), (2, 0), (-2, 0), (0, 2), (0, -2), (2, 2), (2, -2), (-2, 2), (-2, -2)$.

Question: After a finite number of moves, can robot ever reach position $(1, 0)$?

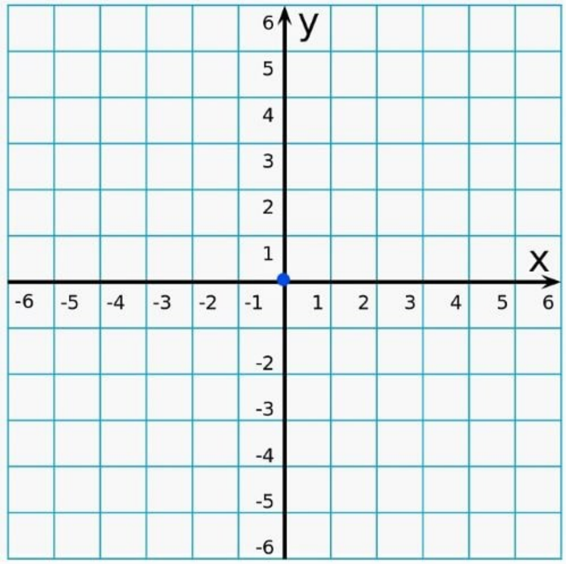
Answer: NO!

It is based on the observation that the sum of coordinates of positions that can be reached is always an even number.

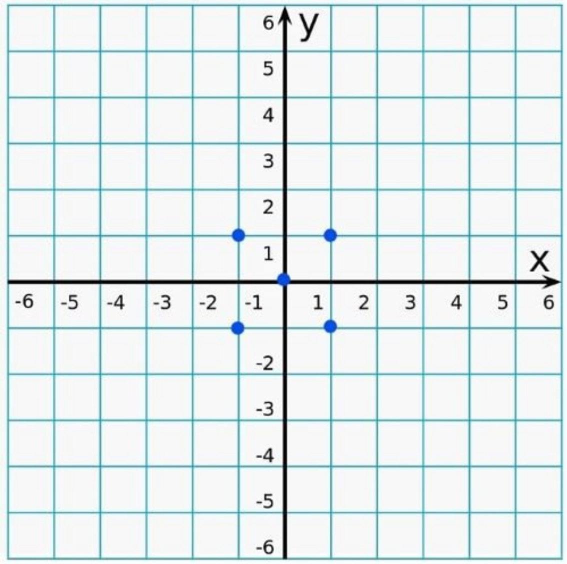
Robot example



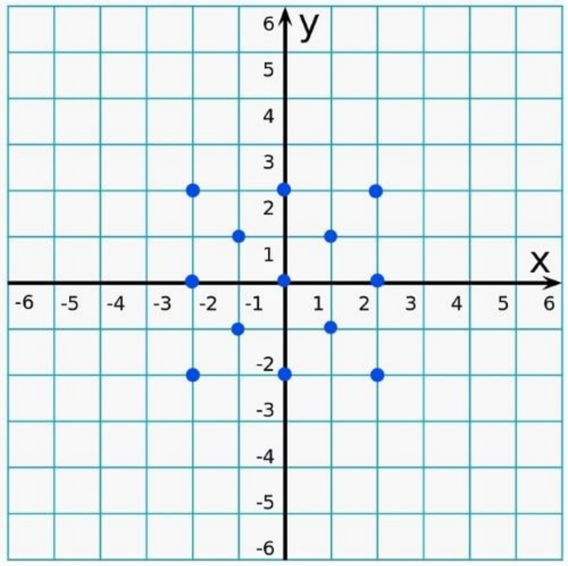
Robot example



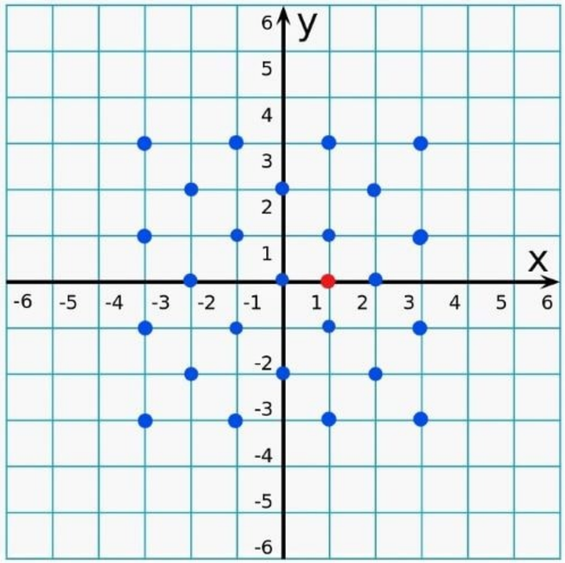
Robot example



Robot example



Robot example



Robot example

Theorem

The sum of robot's coordinates is always even.

Proof.

Define predicate $P(t)$: if the robot is at (x, y) after t steps, then $x + y$ is even.
We will prove that $P(t)$ is invariant (i.e. holds) using induction.
Base step. $P(0)$ is true, since robot starts at $0, 0$ and $0 + 0 = 0$ is even.
Inductive step. Assume that $P(t)$ is true. Let (x, y) be robot's position after t steps. Since $P(t)$ is assumed to be true, it means that $x + y$ is even.
There are four possible next positions for the robot:
$(x + 1, y + 1)$. Then, the sum of coordinates is $x + y + 2$, which is even.
$(x + 1, y - 1)$. The sum is $x + y$, which is even.
$(x - 1, y + 1)$. The sum is $x + y$, which is even.
$(x - 1, y - 1)$. The sum is $x + y - 2$, which is even.
In every case, $P(t + 1)$ is true and so we have proved that $P(t) \to P(t + 1)$.
By induction, $P(t)$ should be true for any t.

Suppose that you would like to show that some property NICE holds for every step of some process or program.

Then it might be helpful to consider the following method:

Invariant Method

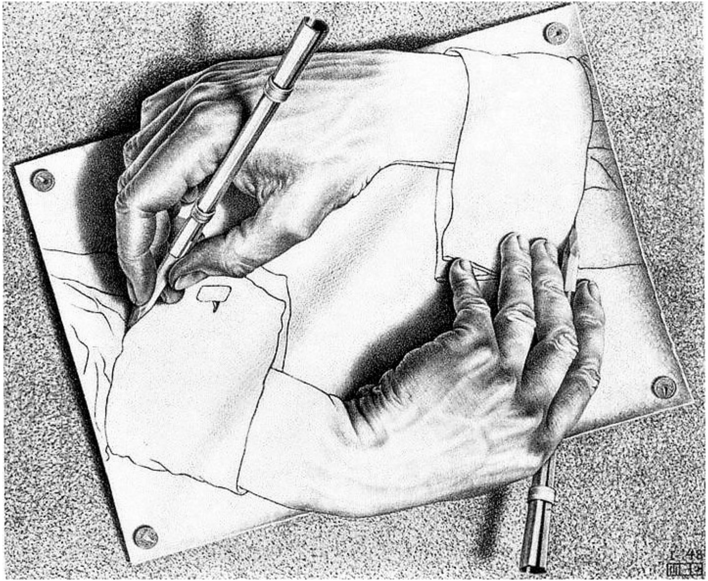
- Define $P(t)$ to be the predicate that NICE holds after step t.
- Show that $P(0)$ is true, i.e. show that NICE holds at the start state.
- Show that

$$\forall t \in \mathbb{N} \quad P(t) \rightarrow P(t+1),$$

  i.e. show that for any $t \geq 0$, if NICE holds after step t,
  then NICE must also hold after the following step $t+1$.

Third picture of the day

Forth picture of the day



Recursive Data Types

Recursive data types are used extensively in CS, and their mathematical foundation is induction.

Recursive data types are specified by **recursive definitions** — how to construct new data elements from previous ones.

Definition

The set of natural numbers $\mathbb{N}$ is a recursive data type defined:

Base case. $0 \in \mathbb{N}$.

Constructor case. If $n \in \mathbb{N}$, then the successor $S(n)$ of n is also in $\mathbb{N}$.

$$
\begin{aligned}
&S(0) \equiv 1 \in \mathbb{N}, \\
&S(S(0)) = S(1) \equiv 2 \in \mathbb{N}, \\
&S(S(S(0))) = S(S(1)) = S(2) \equiv 3 \in \mathbb{N}, \\
&S(S(S(S(0)))) = S(S(S(1))) = S(S(2)) = S(3) \equiv 4 \in \mathbb{N}.
\end{aligned}
$$

Example (Factorial function)

Base case. $\text{Fact}(0) = 1.$

Constructor case. $\text{Fact}(n+1) = (n+1) \cdot \text{Fact}(n), \quad \forall n \in \mathbb{N}.$

$$\begin{aligned}
Fact(7) &= 7 \cdot Fact(6) \\
&= 7 \cdot 6 \cdot Fact(5) \\
&= 7 \cdot 6 \cdot 5 \cdot Fact(4) \\
&= 7 \cdot 6 \cdot 5 \cdot 4 \cdot Fact(3) \\
&\quad \cdots\cdots\cdots\cdots\cdots \\
&= 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \\
&= 7!
\end{aligned}$$

Example (Summation notation)

Let $\mathrm{Sum}(n)$ be the abbreviation of $\sum_{i=1}^{n} f(i)$.

Base case. $\mathrm{Sum}(0) = 0$.

Constructor case. $\mathrm{Sum}(n) = f(n) + \mathrm{Sum}(n-1), \quad \forall n \geqslant 0$.

$$
\begin{aligned}
Sum(7) &= f(7) + Sum(6)\\
&= f(7) + (f(6) + Sum(5))\\
&= f(7) + f(6) + (f(5) + Sum(4))\\
&\quad \cdots\cdots\cdots\cdots\cdots\cdots\\
&= f(7) + f(6) + f(5) + (f(4) + f(3) + f(2) + f(1).
\end{aligned}
$$

Fibbonacci Numbers

Example (Fibonacci numbers)

$$\begin{aligned} \textbf{Base case.}\quad & \mathrm{Fib}(0) = 1. \\ & \mathrm{Fib}(1) = 1. \\ \textbf{Constructor case.}\quad & \mathrm{Fib}(n) = \mathrm{Fib}(n-1) + \mathrm{Fib}(n-2), \quad \forall n \geqslant 2. \end{aligned}$$

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 134, 223, 357, \ldots$$

Ill-defined recursive functions

Example

1. $f_1(n) = 2 + f_1(n-1)$

   This "definition" has NO base case.

2. $$f_2(n) = \begin{cases} 0, & \text{if } n = 0, \\ f_2(n+1), & \text{otherwise.} \end{cases}$$

   This "definition" has a base case, but it is not uniquely defined.

3. $$f_3(n) = \begin{cases} 0, & \text{if } n \text{ is divisible by 2,} \\ 1, & \text{if } n \text{ is divisible by 3,} \\ 2, & \text{otherwise.} \end{cases}$$

   This "definition" has base case, but it is inconsistent: $f_3(6) = 0$ and $f_3(6) = 1$.

Collatz function

Example (Collatz function)

$$f_4(n) = \begin{cases} 1, & \text{if } n \leqslant 1, \\ f_4(n/2), & \text{if } n > 1 \text{ is even}, \\ f_4(3n+1), & \text{if } n > 1 \text{ is odd}. \end{cases}$$

Clearly, $f_4(0) = f_4(1) = 1$, and, $f_4(2) = f_4(1) = 1$.

Compute the next value: $f_4(3) = f_4(10) = f_4(5) = f_4(16) = f_4(8) = f_4(4) = f_4(2) = 1$.

Also, $f_4(4) = 1$, $f_4(5) = 1$, $f_4(6) = f_4(3) = 1$.

Collatz Conjecture

$$f_4(n) \equiv 1, \quad \forall n \in \mathbb{N}.$$

Collatz conjecture was checked for all integers n up to over10^{18}.

Recursive definitions

Definitions of recursive data types have 2 parts, similar to induction principle or definition of recursive functions:

Base case. Specify that some initial elements (usually known) are in data type.

Constructor case. Specify how to construct new data elements from base elements or from previously constructed elements.

Definition

Let $\Sigma \neq \emptyset$ be a set of symbols, called **alphabet**, whose elements are usually called **letters** or **characters** or **symbols**.

The recursive data type Σ^* of **strings** or **words** over alphabet Σ is defined as follows:

Base case. The empty string $\lambda \in \Sigma^*$.

Constructor case. If $a \in \Sigma$ and $s \in \Sigma^*$, then pair $\langle a, s \rangle \in \Sigma^*$.

Definition

Given alphabet Σ, the recursive data type Σ^* over alphabet Σ is defined as follows:

Base case. The empty string $\lambda \in \Sigma^*$.

Constructor case. If $a \in \Sigma$ and $s \in \Sigma^*$, then pair $(a, s) \in \Sigma^*$.

Example

The set $\{0, 1\}^*$ is the set of binary strings.

The alphabet is $\Sigma = \{0, 1\}$.

Usually, we write elements of $\{0, 1\}^*$ as a consecutive sequence of 0 and 1.

For example, $1101001 \in \{0, 1\}^*$.

According to the recursive definition this string is:

$$1101001 = (1, (1, (0, (1, (0, (0, (1, \lambda))))))).$$

Recursive definition of Length

Definition

Let $s \in \Sigma^*$ be some string over alphabet Σ.
The **length** $|s|$ of string s is defined recursively based on definition of Σ^* as follows:

Base case. $|\lambda| = 0$.

Constructor case. $|(a, s)| = |s| + 1$.

Example

$$
\begin{aligned}
|\lambda| &= 0 \\
|1| = |(1, \lambda)| &= 0 + 1 = 1 \\
|01| = |(0, (1, \lambda))| &= 1 + 1 = 2 \\
|001| = |(0, (0, (1, \lambda)))| &= 2 + 1 = 3 \\
|1001| = |((1, 0, (0, (1, \lambda))))| &= 3 + 1 = 4 \\
&\dots \\
|1101001| = |(1, (1, (0, (1, (0, (0, (1, \lambda)))))))| &= 6 + 1 = 7.
\end{aligned}
$$

Recursive definition of Concatenation

Another operation defined on strings is concatenation.

Definition

Let $s, t \in \Sigma^*$ be 2 strings. The **concatenation** $s \cdot t$ is defined recursively as follows:

Base case. $\lambda \cdot t = t$.

Constructor case. $(a, s) \cdot t = (a, s \cdot t)$.

Example

Let $\Sigma = \{a, b\}$. What is $bb \cdot abab$?

$$\begin{aligned} bb \cdot abab = (b, (b, \lambda)) \cdot abab &= (b, (b, \lambda) \cdot abab) \\ &= (b, (b, \lambda \cdot abab)) \\ &= (b, (b, abab)) \\ &= (b, babab) \\ &= bbabab. \end{aligned}$$

Structural Induction

Structural induction is a method for proving that all the elements of a recursively defined data type have some property.

A structural induction proof has 2 parts:

1. Prove that base case elements have the property.
2. Prove that each constructor case elements have the property, when the constructor rules are applied to elements that have the property.

Theorem

For all $s, t \in \Sigma^*$,

$$|s \cdot t| = |s| + |t|.$$

Structural Induction Example

$$|s \cdot t| = |s| + |t| \quad \forall s, t \in \Sigma^*.$$

Proof.

Proof by structural induction.
Induction hypothesis is

$$P(s): \quad \forall t \in \Sigma^*, \quad |s \cdot t| = |s| + |t|.$$

Base Case. $(s = \lambda)$:

$$\begin{aligned}
|s \cdot t| &= |\lambda \cdot t| \\
&= |t| && \text{(def of } \cdot \text{, base case)} \\
&= 0 + |t| \\
&= |s| + |t| && \text{(def of length, base case)}
\end{aligned}$$

Structural Induction Example

Proof.

Inductive Case.
Suppose $s = (a, r)$ and assume the induction hypothesis $P(r)$ holds.
Need to show that $P(s)$ also holds.

$$
\begin{aligned}
|s \cdot t| &= |(a, r) \cdot t| \\
&= |(a, r \cdot t)| && \text{(concat def, constructor case)} \\
&= 1 + |r \cdot t| && \text{(length def, constructor case)} \\
&= 1 + (|r| + |t|) && \text{(since } P(r) \text{ holds)} \\
&= (1 + |r|) + |t| && \text{(associative law for } +\text{)} \\
&= |(a, r)| + |t| && \text{(length def, constructor case)} \\
&= |s| + |t|.
\end{aligned}
$$

This proves that $P(s)$ holds. By structural induction, the proof is completed. □

Arithmetic expressions

Expression evaluation is a key feature of programming languages.

Recognition of expressions as a recursive data type is a key to understanding how they can be processed.

To illustrate this approach let's take a look at a simple example:

Consider an arithmetic expressions involving only one variable x:

$$3x^2 + 2x + 1.$$

Arithmetic expressions involve:

- Numerals (symbols for numbers);
- Variable;
- Symbols for ariithmetic operations.

Such arithmetic expressions in one variable are collected in data type called *Aexp* defined recursively on the next page.

Definition

Base case. Variable $x \in Aexp$.

Arabic numeral $\texttt{k} \in Aexp, \quad \forall k \in \mathbb{N}$.

Constructor case. Let $\forall e, f \in Aexp$. Then,

- $[e + f] \in Aexp$. Expression $[e + f]$ is called a **sum**, and e, f are called **summands**.
- $[e * f] \in Aexp$. Expression $[e * f]$ is called a **product**, and e, f are called **multipliers** or **multiplicands**.
- $-[e] \in Aexp$. Expression $-[e]$ is **negative**.

Example

Expression $3x^2 + 2x + 1$ officially is written as:

$$[\,[\,3 * [\,x * x\,]\,] + [\,[\,2 * x\,] + 1\,]\,].$$

Evaluating an arithmetic expression for a given value is standard operation.

For example, if the value of x is 3, then the value of $3x^2 + 2x + 1$ is 34.

Generally, if $e \in Aexp$ and $n \in \mathbb{Z}$ is the value of x, we evaluate expression e to find its value $eval(e, n)$.

In other words,

$$eval\Big(\underbrace{[\,[\,3 * [\,x * x\,]\,] + [\,[\,2 * x\,] + 1\,]\,]}_{=e}, 3\Big) = 34.$$

Evaluation process is defined using a recursive definition

(see next page).

Evaluation of arithmetic expressions

Definition

The **evaluation function** $eval : Aexp \times \mathbb{Z} \to \mathbb{Z}$ is defined for $\forall e \in Aexp$ and $\forall n \in \mathbb{Z}$ as follows:

Base case.
- $eval(x, n) = n$, (value of variable x is n).
- $eval(\mathtt{k}, n) = k$, (value of numeral $\mathtt{k}$ is k).

Constructor case.
- $eval([e + f], n) = eval(e, n) + eval(f, n)$.
- $eval([e * f], n) = eval(e, n) \cdot eval(f, n)$.
- $eval(-e, n) = -eval(e, n)$.

Evaluation of arithmetic expressions

Base case.

- $eval(x, n) = n,$ (value of variable x is n). (1)
- $eval(\mathtt{k}, n) = k,$ (value of numeral $\mathtt{k}$ is k). (2)

Constructor case.

- $eval([e + f], n) = eval(e, n) + eval(f, n).$ (3)
- $eval([e * f], n) = eval(e, n) \cdot eval(f, n).$ (4)
- $eval(-e, n) = -eval(e, n).$ (5)

Example (Evaluation of $5 + x^2$ when $x = 3$)

$$
\begin{aligned}
eval([\,5 + [\,x * x\,]\,], 3) &= eval(5, 3) + eval([\,x * x\,], 3) && \text{Use (3)}\\
&= 5 + eval(\,[\,x * x\,]\,, 3) && \text{Use (2)}\\
&= 5 + (eval(x, 3) \cdot eval(x, 3)) && \text{Use (4)}\\
&= 5 + (3 \cdot 3) && \text{Use (1)}\\
&= 5 + 9 = 14.
\end{aligned}
$$

Lecture 5 Summary

- Induction Principle;
- Strong Induction Principle;
- Invariant Principle;
- Recursive Data Type;
- Recursive Functions;
- Recursive Definitions;
- Structural Induction.

Theorem

1 is the smallest positive number.

Proof.

Let the smallest positive number be called x.

Clearly, x^2 is also positive. Since x is the smallest positive number, then

$$x^2 \geqslant x.$$

Divide both sides by the positive number x to get

$$x \geqslant 1.$$

Since x is the smallest positive number, and 1 is positive, it follows that $x = 1$.
Thus 1 is the smallest positive number. $\square$