Why?
To solve several problems using a general solution.

Computer Science
- Solve problems.

Not to make solutions so general because it may not be solvable, but we will be wasting our time.

We need to check if a program (formulate in some generality) is solvable.

First question: Is the program solvable?

- Yes
  - On what machine?
  - How feasible is the solution?

- No
  - What do we do?
  - Solve problem with high probability
  - May solve approximately

This is theory of computation!

In the exams we will have 'why' questions.

To prove that something is not computable we need a precise definition of "computable"

Church thesis
Church-Turing thesis

\[ A \leq_1 \] Alonso
New Section

- Computer Architecture
- Turing machine
- Operation by operation

- Computable if it is computable in a Turing machine
- Is paper published in 1932 1934

German international language of science - Equivalent definitions

Church-Turing Thesis: Anything computable on any physical device can be computed on a Turing machine by a Java program.

To prove that something is not computable we need to prove that it cannot be computed by a Java program.

Problem: Java has many options, many features, analyzing all of them is too complicated.

Some features of Java add convenience, efficiency, etc., but do not change what is computable.

Transforming: Objects and operations on objects

Computers are 0’s and 1’s

Objects: 0, 1, 2, 3, ..., n
Operations: +, -, *, 1, %, +1, -1

1 11
\[ k \cdot k = 0 \Rightarrow \sigma_{\text{small}} = \text{next} \]
\[ \quad = \quad = \text{prev} \]

Combination: \( \text{for-loops, if-then-else, while-loops} \)

Projection is moving points to \( x \)-axis.

Definition: A function is called "simple" if it can be obtained from \( 1, 2, \ldots, n, \ldots, n^k, \sigma^2, t, \ldots, x_1, \ldots, t^k \) by using
\[ a \text{ for loop} \]

Example power in Java Program
\[ a^n \]
\[ \text{in Java} \]
\[ \text{int pow} = 1 \]
\[ \text{for(}i=1; i<=n; i++) \]
\[ \quad \text{pow} = \text{pow} \times a; \]

\[ \text{Primitive recursion} \]
\[ \text{Recurso = going back} \]

\[ \text{pow}(a, 0) = 1 \]
\[ \text{pow}(a, 1 + n = \text{pow}(\text{a}^n)) \times a \]

\[ \text{mathemathical proof} \]
What is computable? ...

For loops ⇒ how to formalize them

Code:

\[
\begin{align*}
\text{pow} &= 1; \\
\text{for } (i = 1; i \leq n; i++) & \quad \text{pow} = \text{pow} \times a; \\
\end{align*}
\]

in general terms.
- initial conditions
- how the value at next iterations depends on the previous

\[ f(l_{n_1}, \ldots, n_k, n) \rightarrow \text{what we are defining} \]

\[ f(l_{n_1}, \ldots, n_k, 0) = g(l_{n_1}, \ldots, n_k) \rightarrow \text{initialization} \]

\[ f(l_{n_1}, \ldots, n_k, m+1) = h(n_1, \ldots, n_k, n, f(l_{n_1}, \ldots, n_k, n)) \rightarrow \text{what happens inside the loop} \]

primitive recursion \( f = PTZ(g, h) \)

\[ f(n_1, 0) = 1 \quad f = \text{pow} = PTZ(1, \Pi_1^{\ast}, n_1) \]

\[ f(n_1, m+1) = f(n_1, m) \times n_1 \quad g(n_1) = ( \]

\[ h(n_1, m, f(n_1, m)) = n_1 \times f(n_1, m) \]

\[ h(a_1b_1, c) = a \times c = \Pi_1^{\ast}, \Pi_3^{\ast} \]

\[ h(a_1b_1, c) = a \times c = \Pi_1^{\ast}, \Pi_3^{\ast} \]
```
int f = 01(n_1, ..., n_k);
for (int i = 1; i <= n_1; i++)
    f = h(n_1, ..., n_k, n_1, f);
```

\( \alpha \) is \( \alpha + 1 \) so:

\[
0, 1, 2 \\
1 = \sigma(0) \\
2 = \sigma(1)
\]

\[
a + b = a + \underbrace{b + 1 + 1 + 1 + \cdots}_{b \text{ times}}
\]

```
int sum = a;
for (int i = 1; i <= b; i++)
    sum = sum + 1;
```

---

**Primitive Recursive**

A function is called primitive recursive if it can be obtain from \( 0, \sigma \) and \( \Pi \) by using composition and primitive recursion.

\[
f = PR(g, h); \\
f(n_1, ..., n_k, n) = g(la_1, ..., a_k) \\
f(n_1, ..., n_k, m+1) = h(la_1, ..., n_k, m, Ha_1, ..., n_k, m))
\]

**Theorem:** Addition is primitive recursive

\[
a + b = a + \underbrace{b + 1 + 1 + \cdots}_{b \text{ times}}
\]
\text{Let } \text{sum} = a_1; \\
\text{for } (i+1, j+1) \leq b_{i,j} + k; \\
\text{sum} = \text{sum} + 1; \\
\text{sum}(a, b) = a; \\
\text{sum}(a, b + 1) = \text{sum}(a, b) + 1.

To know how many variables we count,
\( \text{\( n \text{ for loop} \) addition using for-loop} \)

**Theorem:** Multiplication is PR

\[ a \times b = a + a + \ldots + a \]

\[ \text{for times} \]

\[ \text{int prod } = \emptyset \]

\[ \text{for(int i = 1; i <= b; i++)} \]

\[ \text{prod } = \text{prod } + a; \]

**Mathematical notation**

\[ \text{prod}(a, \emptyset) = \emptyset \]

\[ \text{prod}(a, i+1) = \text{prod}(a, i) + a \]

\[ \text{f}(n, \emptyset) = g(n) \]

\[ \text{f}(n, m+1) = \text{f}(n, m, \text{f}(n+1, m)) \]

\[ \text{f}(n, \emptyset) = \emptyset \]

\[ \text{f}(n, m+1) = \text{f}(n, m) + n \]

\[ g(n) = \emptyset \]

\[ g = \emptyset \]

\[ h = \bar{n}^3 + \bar{n}^3 \implies h = \text{add}(\bar{n}^3, \bar{n}^3) \]
\( \text{Pr}(0, \text{add}(\pi_2^3, \pi_1^3)) \) — this is enough
\[ \text{add} = \text{Pr}(\pi_1^1, \text{add}(\pi_3^3)) \]
\[ \text{prod}(\emptyset, \text{add}(\pi_2^3, \text{add}(\pi_1^3, \text{add}(\pi_3^3, \pi_1^3)))) \]

\( a^5 \) is PR
\[ a^5 = a \times a \times \ldots \times a \]
5 times

\( \text{int} = \text{fact} = 1; \)
\[ \text{for}(\text{int} = 1; i < n; i++) \]
\[ \text{fact} = \text{fact} \times i; \]
\[ \text{fact}(0) = 1 \]
\[ \text{fact}(m+1) = \text{fact}(m) \times (m+1) \Rightarrow \frac{\text{fact}(0)}{\text{fact}(m+1)} = \frac{1}{\text{fact}(m) \times (m+1)} \]

\[ K = \emptyset \]
\[ \text{fact}(0) = g(\emptyset) \]
\[ \text{fact}(m+1) = n(m) \times \text{fact}(m) \]

\[ \Delta \]
\[ g = \sigma(\emptyset) = \sigma \times \emptyset \]
\[ h = \pi_2^2 \times (\pi_1^2 + 1) \]
\[ \sigma \times \pi_1^2 \]
\[ h = \text{prod}(\pi_2^2, \text{add}(\pi_1^2, \sigma \pi_1^2)) \]
\[ \text{fact} = \text{Pr}(\emptyset, \emptyset, \text{prod}(\pi_2^2, \sigma \pi_1^2)) \]