Summary

Inhalt

[1. Introduction 2](#_Toc58838094)

[1.1 History 2](#_Toc58838095)

[1.1.1 David Hilbert 2](#_Toc58838096)

[1.1.2 Kurt Gödel 2](#_Toc58838097)

[1.1.3 Alan Turing 2](#_Toc58838098)

[1.2 Infinity times Infinity 2](#_Toc58838099)

[1.3 The Turing Machine 3](#_Toc58838100)

[1.3.1 Efficient Algorithms 3](#_Toc58838101)

# 1. Introduction

The central questions are:

* What cannot be done using a computer?
* What cannot be done efficiently using a computer?

## History

### 1.1.1 David Hilbert

Wanted to create a system of axioms in which every true statement can be proven, and that does not contain any paradoxes and inconsistencies.

### 1.1.2 Kurt Gödel

If a system is consistent (meaning there are no paradoxes) and powerful enough, then there are true statements in this system that cannot be proven (meaning it is incomplete).

The conclusion form this is that

* Mathematics is incomplete
* There are statements that we can neither prove not disprove
* This will stay that way

### 1.1.3 Alan Turing

Around the same time as Gödel, Turing asked

* What can be automated?
* Where lies the border to what can be automated?

He concluded that there are provable statements that we cannot prove automatically.

## 1.2 Infinity times Infinity







**Countable Infinity**: An infinite set A is called countable if it has the same size as N. This means that every element of A can be assigned to a natural number and can therefor count the elements of A.

**Uncountable Infinity**: Can be proven by contradiction.

## 1.3 The Turing Machine

The basic idea is

* There is a countable number of algorithms
* But an uncountable number of problems

**Alphabet**: finite set of symbols (letters)

**Word**: string of symbols

Example: DEC={1,2,3}, x=31232 is a word “over” DEC

* **Decision problem L**: Decide, for a given word x, whether it is in the set L
* **Algorithm**: Finite step-by-step method to solve every instance of a given problem in finite time

**Church Turing Thesis**: Turing machines that halt can compute exactly what an algorithm can compute

There are decision problems, which cannot be solved by any TM. They are called **undecidable**.

### 1.3.1 Efficient Algorithms

**Thesis of Cobham and Edmonds**: Efficient algorithms are those that run in polynomial time 

The term is independent of the concrete computing model. Therefore we call the class of polynomials robust

**Concrete computing model**: If a “real algorithm” can compute something in then there is a TM that can do it in .

The class  contains all decision problems that can be solved efficiently by TMs.

We are interested in the running time in the worst case