INT201 Decision, Computation and Language

Lecture 1 – Overview

Dr Yushi Li

Xi’an Jiaotong-Liverpool University
西交利物浦大学

Lecturer-Dr Yushi Li

- Graduated from The Hong Kong Polytechnic University
- Office hour: 10:00 – 12:00 Tuesday (Teaching days)
- Contact: Yushi.Li@xjtlu.edu.cn

Lecturer-Dr Yushi Li

- Research in computer vision and graph learning

✓ Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do.

✓ Generally speaking, graph learning refers to machine learning on graphs. Graph learning methods map the features of a graph to feature vectors with the same dimensions in the embedding space.

Lecturer-Dr Chunchuan Lyu (Module leader)

- Graduated from The University of Edinburgh
- Office hour: 15:00-17:00 Wednesday (Teaching days)
- Contact: chunchuan.lyu@xjtlu.edu.cn

Timetable Information

- Lecture (onsite)

  Tuesday: 14:00 – 15:50, Week 1-13;

- Tutorial: (onsite)

  Thursday: 11:00 – 12:50, Week 1-13.

Assessments

- 2 assignments (10% + 10%)
- Final exam (80%)
- Resit exam (100%)

Decision, Computation and Language

- This module is about the theory of computation and answering the questions:

1. What are the mathematical properties of computer hardware and software?

2. What is a computation and what is an algorithm? Can we give rigorous mathematical definitions of these notions?

3. What are the limitations of computers? Can “everything” be computed? (As we will see, the answer to this question is “no”.)

Decision, Computation and Language

Purpose of the theory of Computation:

Develop formal mathematical models of computation that reflect real-world computers.

Brief history

Already in 1930's Alan Turing studied an abstract machine, Turing machine!

In 1940's and 1950's simpler kinds of machines, finite automata, were studied. In the late 1950's Noam Chomsky began his study of formal grammar.

In 1969 Stephen Cook separated problems that are "intractable" (also called "NP-hard")

Decision, Computation and Language

Importance of this course

- This course is about the fundamental capabilities and limitations of computers. These topics form the core of computer science.
- It is about mathematical properties of computer hardware and software.
- This theory is very much relevant to practice, for example, in the design of new programming languages, compilers, string searching, pattern matching, computer security, artificial intelligence, etc.
- This course helps you to learn problem solving skills. Theory teaches you how to think, prove, argue, solve problems, express, and abstract.
- This theory simplifies the complex computers to an abstract and simple mathematical model, and helps you to understand them better.
- This course is about rigorously analyzing capabilities and limitations of systems.

What will we discuss in this module?

Notations for representing formal languages that give us:

- ✓ ways to define them precisely;
- ✓ ways to build compilers that recognize the languages;
- ✓ ways to check whether
    - a string of symbols belongs to a language
    - two alternative descriptions of languages are actually the same language

What will we discuss in this module?

Tools to analyze languages

These tools originate in analysis of natural languages as well as programming languages

NL (natural languages): want to recognize valid sentence

PL (programming languages): want to recognize valid program

What will we discuss in this module?

Some notations/methods for describing a language (other than explicitly listing it) include:

- ✓ Finite state automaton
- ✓ Regular expression
- ✓ Context-free grammar
- ✓ Pushdown Automaton
- ✓ Turing machine

We will see that some of the above can describe more languages than others (more powerful than others).

Basic concepts

- Languages
- Grammars
- Automata

Languages

Informal definition: a system suitable for the expression of certain ideas, facts, or concepts, including a set of symbols and rules for manipulation. This is not sufficient, we need a precise definition for **languages** in the theory of computation (**Formal languages**).

Spoken languages: English, French, etc, are not formal languages, although we can still try to write down rules that work most of the time.

Programming languages: Python, C++, C#, Java ...

Formal languages include programming languages, database query languages, various file formats.

Languages

Informal descriptions:

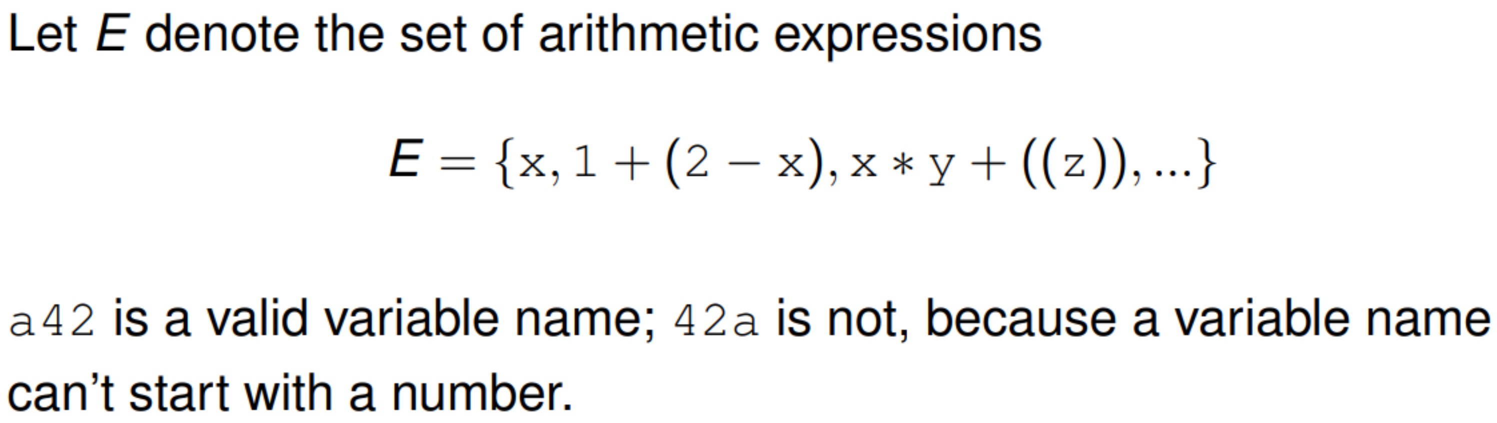
“An arithmetic expression is constructed from variables and numbers, and infix operators +, -, *, /, and subexpressions may possibly be enclosed in parentheses, in which case every (must have a corresponding) ...”

“A comment begins with /* and ends with */”

“Your password should have 4-8 characters and contain a non-alphabetic character.”

Languages

Informal descriptions:



Let E denote the set of arithmetic expressions

$$E = \{\mathrm{x}, 1 + (2 - \mathrm{x}), \mathrm{x} * \mathrm{y} + ((\mathrm{z})), \ldots\}$$

`a42` is a valid variable name; `42a` is not, because a variable name can't start with a number.

The variable-name description is a combination of English and examples. We need some notation to express these descriptions more precisely!

Languages (Formal)

Before giving the precise definition of languages, we need to briefly introduce several symbols and their definitions.

Alphabet – a finite, nonempty set $\sum$ of symbols.

String (word) – a finite sequence of symbols from the alphabet

Example

$\sum = \{a, b\}$, then *abab, aaaabbba* are strings on $\sum$

$w = abaaa$ *indicates* the string named w has the specific value *abaaa*

Languages

Empty string (word) – the string with no symbols, denoted as ϵ.

$$|\epsilon| = 0$$

$$\epsilon w = w\epsilon = w$$

Length of a string – the number of symbols in the string, denoted as $|w|$.

$$|w_1 w_2| = |w_1| + |w_2|$$

Reverse of a string – obtained by writing the symbols in reverse order, denoted as w^R

$$w = a_1\ a_2 \ldots\ a_n$$

$$w^R = a_n\ \ldots\ a_2\ a_1$$

A **palindrome** is a string w satisfying $w = w^R$.

Languages

The **concatenation** of two strings is obtained by appending the symbols to the right end (concatenation is associative).

$$w = a_1\ a_2 \dots\ a_n$$

$$v = b_1\ b_2 \dots\ b_n$$

$$wv = a_1\ a_2 \dots\ a_n\ b_1\ b_2 \dots\ b_n$$

If u, v, w are strings and $w{=}uv$ then u is a **prefix** of w and v is a **suffix** of w. A proper prefix of w is a prefix that is not equal to ϵ of w. (similarly for proper suffix)

w^n denotes the concatenation of n copies of w

$\sum^*$ (**star closure** of $\sum$) denotes the set of all strings (words) over $\sum$.

$\sum^+$ (**positive closure** of $\sum$) denotes the set of all non-empty strings (words) over $\sum$.

Both of $\sum^*$ and $\sum^+$ are infinite set.

Languages

A **formal language** is a set of strings (sequences of symbols) constructed from a finite alphabet, according to a specific set of precise, mathematical rules. The key idea is that it is defined by its form, not its meaning .

A **formal language** $\boldsymbol{L}$ over alphabet $\sum$ is a subset of $\sum^*$.

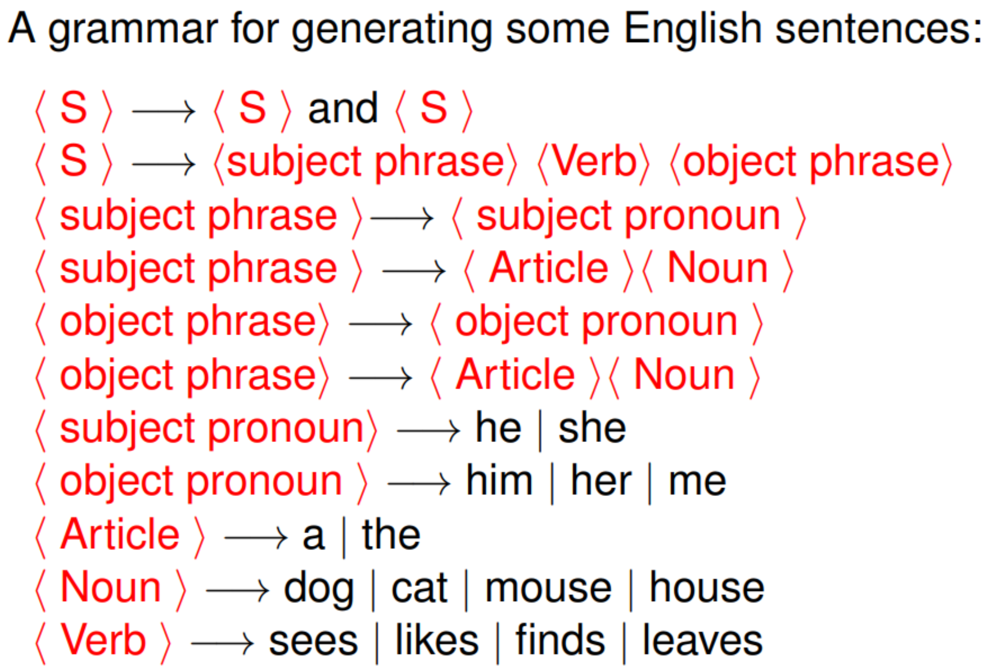
We can express new languages in terms of other languages using concatenation and closure:

$$\boldsymbol{L}_1\boldsymbol{L}_2 = \{w_1w_2 : w_1 \in \boldsymbol{L}_1 \; and \; w_2 \in \boldsymbol{L}_2\}$$

$$\boldsymbol{L}^* = \{w_1w_2 \ldots w_{\mathrm{n}} : \; n \geq 0 \; and \; w_1, w_2, \ldots w_{\mathrm{n}} \in \boldsymbol{L}\}$$

Grammars

Set of rules for generating syntactically correct programs/sentences.



A grammar for generating some English sentences:

$\langle \text{S} \rangle \longrightarrow \langle \text{S} \rangle$ and $\langle \text{S} \rangle$

$\langle \text{S} \rangle \longrightarrow \langle \text{subject phrase} \rangle \langle \text{Verb} \rangle \langle \text{object phrase} \rangle$

$\langle \text{subject phrase} \rangle \longrightarrow \langle \text{subject pronoun} \rangle$

$\langle \text{subject phrase} \rangle \longrightarrow \langle \text{Article} \rangle \langle \text{Noun} \rangle$

$\langle \text{object phrase} \rangle \longrightarrow \langle \text{object pronoun} \rangle$

$\langle \text{object phrase} \rangle \longrightarrow \langle \text{Article} \rangle \langle \text{Noun} \rangle$

$\langle \text{subject pronoun} \rangle \longrightarrow$ he | she

$\langle \text{object pronoun} \rangle \longrightarrow$ him | her | me

$\langle \text{Article} \rangle \longrightarrow$ a | the

$\langle \text{Noun} \rangle \longrightarrow$ dog | cat | mouse | house

$\langle \text{Verb} \rangle \longrightarrow$ sees | likes | finds | leaves

Grammars

- The grammar can generate sentences like “the dog sees the cat and the mouse leaves the house”, which may be nonsense e.g. “the house sees me”.
- Arbitrarily long sentences can be generated (which you can’t do by enumeration!).
- Semantics can be given with reference to grammar, e.g. logical conjunction of subsentences formed by word “and” .
- You can’t define natural language sentences completely this way, but you can for programming languages.

Grammars

Applications in programming languages (stages of compilation):

1. **Lexical analysis**

2. **Parsing**

3. **Code generation**

4. **Code optimization**

Grammars

Lexical Analysis: divide sequence of characters into tokens, such as variable names, operators, labels. In a natural language tokens are strings of consecutive letters (easy to recognize!)

```
pay=salary+(overtimerate*overtime);
```

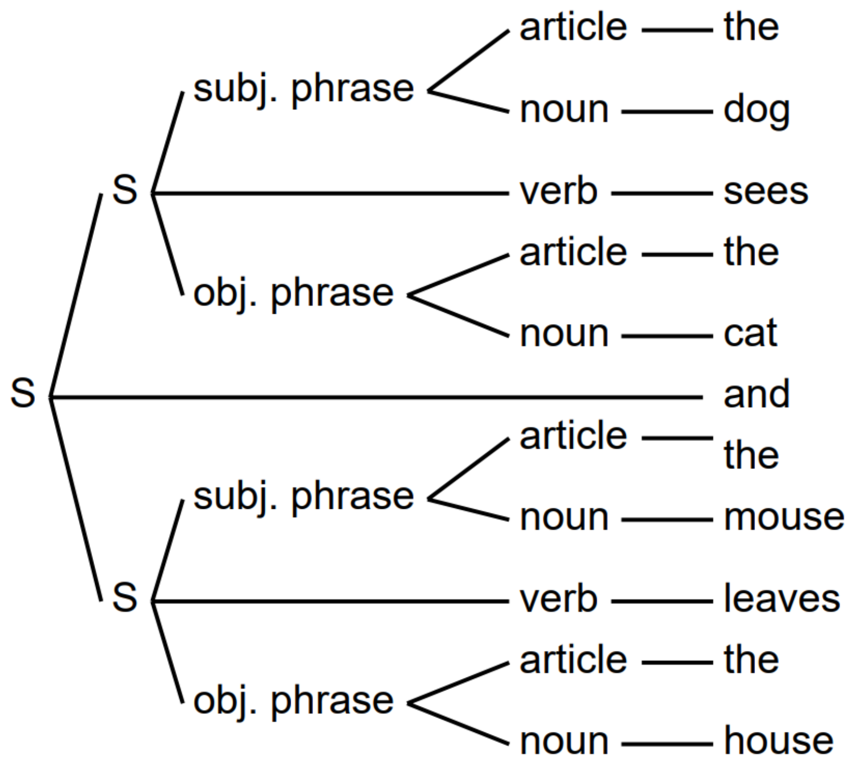
Break into tokens as follows:

```
pay
=
salary
+
(
overtimerate
*
overtime
)
;
```

Grammars

Parse Tree: identify relationships between tokens

parse tree or parsing tree or derivation tree or concrete syntax tree is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar.



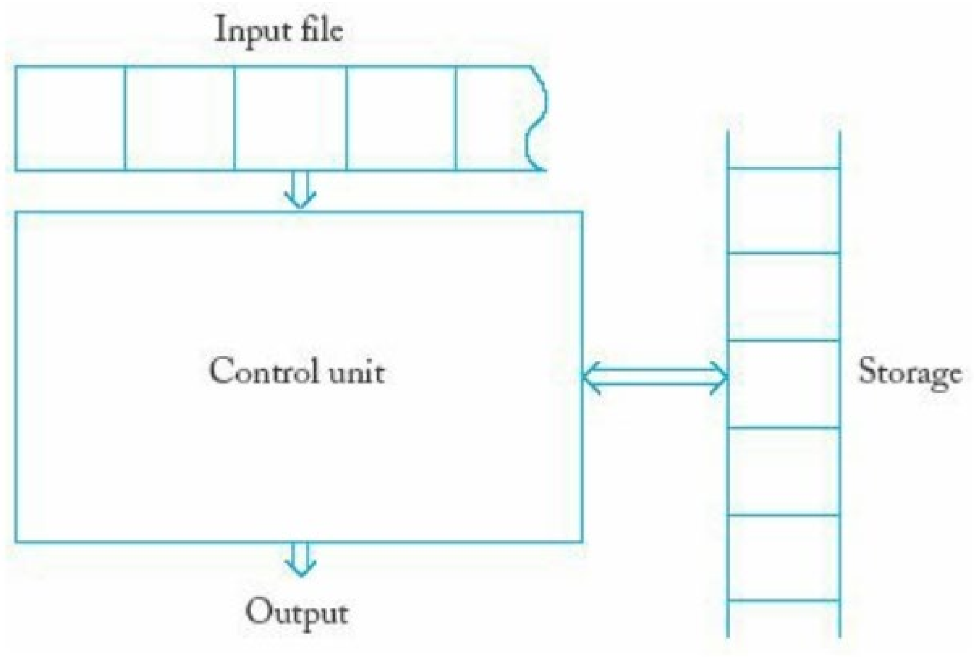
Grammars

Code generation: is part of the process chain of a compiler and converts intermediate representation of source code into a form (e.g., machine code) that can be readily executed by the target system.

Code optimization: is the process of modifying a software system to make some aspect of it work more efficiently or use fewer resources. In general, a computer program may be optimized so that it executes more rapidly, or to make it capable of operating with less memory storage or other resources, or draw less power.

Automata

An automaton is an abstract model of a digital computer



Mathematical preliminaries

Complexity theory

The main question asked in this area is “What makes some problems computationally hard and other problems easy?”

Informally, a problem is called “easy”, if it is efficiently solvable.

Example

- sorting a sequence of, say, 1,000,000 numbers,
- searching for a name in a telephone directory,
- computing the fastest way to drive from Suzhou to Shanghai.

On the other hand, a problem is called “hard”, if it cannot be solved efficiently, or if we don’t know whether it can be solved efficiently.

Example

- time table scheduling for all courses at XJTLU
- factoring a 300-digit integer into its prime factors

Computation theory

Central question in Computability Theory: classify problems as being solvable or unsolvable.

An example of such a problem is “Is an arbitrary mathematical statement true or false?”

To attack such a problem, we need formal definitions of the notions of:

- Computer
- Algorithm
- Computation

The theoretical models that were proposed in order to understand solvable and unsolvable problems led to the development of real computers.

Automata theory

Central question in Automata Theory: Do these models have the same power, or can one model solve more problems than the other?

Automata Theory deals with definitions and properties of different types of "computation models". Examples of such models are:

- Finite Automata. These are used in text processing, compilers, and hardware design.
- Context-Free Grammars. These are used to define programming languages and in Artificial Intelligence.
- Turing Machines. These form a simple abstract model of a "real" computer, such as your PC at home.

Mathematical preliminaries

- Set
- Relation and function
- Graph
- Proof techniques

Mathematical preliminaries

Set

A **set** is a collection of well-defined objects.

Examples

- the set of all Olympic Gold Medallists,
- the set of all pubs in Hong Kong,
- the set of all even natural numbers.

Mathematical preliminaries

Set

If A and B are sets, then A is a **subset** of B, written as $A \subseteq B$, if every element of A is also an element of B.

For example, the set of even natural numbers is a subset of the set of all natural numbers. Every set A is a subset of itself, i.e., $A \subseteq A$. The empty set is a subset of every set A, i.e., $\emptyset \subseteq A$.

If B is a set, then the **power set** P(B) of B is defined to be the set of all subsets of B: $P(B) = \{A : A \subseteq B\}$. Observe that $\emptyset \in P(B)$ and $B \in P(B)$.

Mathematical preliminaries

Set

If A and B are two sets, then

- their **union** is defined as $A \cup B = \{x : x \in A \text{ or } x \in B\}$,
- their **intersection** is defined as $A \cap B = \{x : x \in A \text{ and } x \in B\}$,
- their **difference** is defined as $A \setminus B = \{x : x \in A \text{ and } x \notin B\}$,
- the **Cartesian product** of A and B is defined as $A \times B = \{(x, y) : x \in A \text{ and } y \in B\}$,
- the **complement** of A is defined as $A = \{x : x \notin A\}$.

Mathematical preliminaries

Relation and function

A **binary relation** on two sets A and B is a subset of A $\times$ B (Cartesian product of A and B).

A **function** f from A to B, denoted by f : A → B, is a binary relation R, having the property that for each element a ∈ A, there is exactly one ordered pair in R, whose first component is a. We will also say that f(a) = b, or f maps a to b, or the image of a under f is b. The set A is called the domain of f, and the set

$$\{b \in B : \text{there is an } a \in A \text{ with } f(a) = b\}$$

is called the range of f.

Mathematical preliminaries

Relation and function

A function $f : A \rightarrow B$ is **one-to-one** (or **injective**), if for any two distinct elements a and a' in A, we have $f(a) \neq f(a')$. The function f is **onto** (or **surjective**), if for each element $b \in B$, there exists an element $a \in A$, such that $f(a) = b$; in other words, the range of f is equal to the set B. A function f is a **bijection**, if f is both injective and surjective.

Mathematical preliminaries

Relation and function

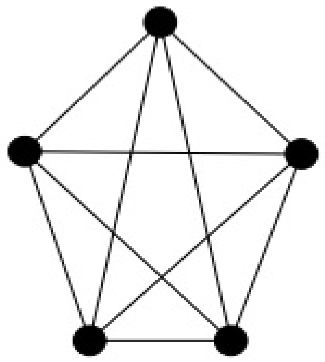
A binary relation $R \subseteq A \times A$ is an **equivalence relation**, if it satisfies the following three conditions:

- R is **reflexive**: For every element in $a \in A$, we have $(a, a) \in R$.
- R is **symmetric**: For all a and b in A, if $(a, b) \in R$, then also $(b, a) \in R$.
- R is **transitive**: For all a, b, and c in A, if $(a, b) \in R$ and $(b, c) \in R$, then also $(a, c) \in R$.

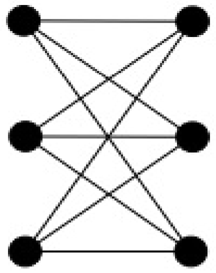
Mathematical preliminaries

Graph

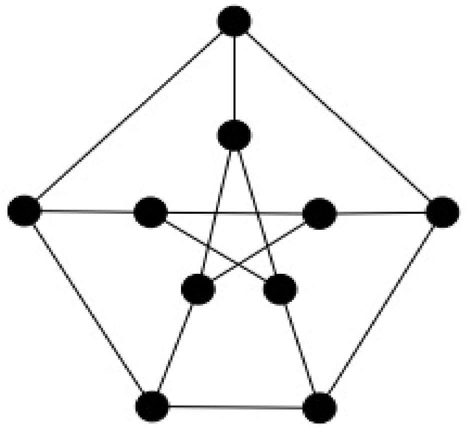
A **graph** G = (V, E) is a pair consisting of a set V , whose elements are called **vertices**, and a set E, where each element of E is a pair of distinct vertices. The elements of E are called **edges**.



K_5



$K_{3,3}$



Peterson graph

Mathematical preliminaries

Graph

The **degree** of a vertex v, denoted by deg(v), is defined to be the number of edges that are incident on v.

A **path** in a graph is a sequence of vertices that are connected by edges. A path is a **cycle**, if it starts and ends at the same vertex. A **simple path** is a path without any repeated vertices. A graph is **connected**, if there is a path between every pair of vertices.

Mathematical preliminaries

Proof techniques

Axiom: an unprovable rule or first principle accepted as true because it is self-evident or particularly useful.

Statement (proposition): a declarative sentence that is either true or false but not both. In proof, it often includes axioms, hypotheses of the theorem to be proved, and previously proved theorems.

Theorem: a statement that is true.

Proof: a sequence of mathematical statements that form an argument to show that a theorem is true.

Mathematical preliminaries

Proof techniques

How do we go about proving theorems?

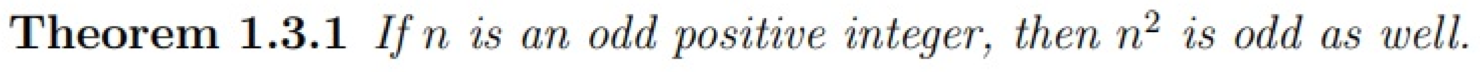
There is no specified way of coming up with a proof, but there are some generic strategies that could be of help:

- Direct proof
- Constructive proof
- Non-constructive proof
- Contradiction proof
- Induction proof

Direct proof

Approach the theorem directly.

Example



Theorem 1.3.1 *If n is an odd positive integer, then n^2 is odd as well.*

Constructive proof

A method of proof that demonstrates the existence of a mathematical object (anything that has been formally defined) by creating or providing a method for creating the object.

Example

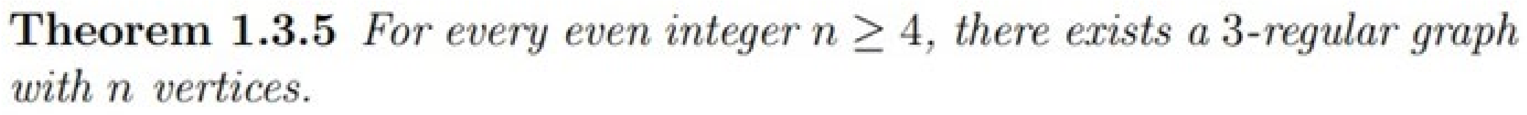
There exists an object with property $\mathcal{P}$.

Proof. Here is the object: [. . .]

And here is the proof that the object satisfies property $\mathcal{P}$: [. . .]

Constructive proof

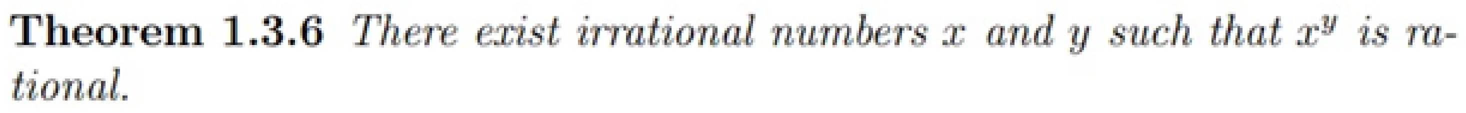
Example


Theorem 1.3.5 *For every even integer $n \geq 4$, there exists a 3-regular graph with n vertices.*

Nonconstructive proof

In a nonconstructive proof, we show that a certain object exists, without actually creating it.

Example

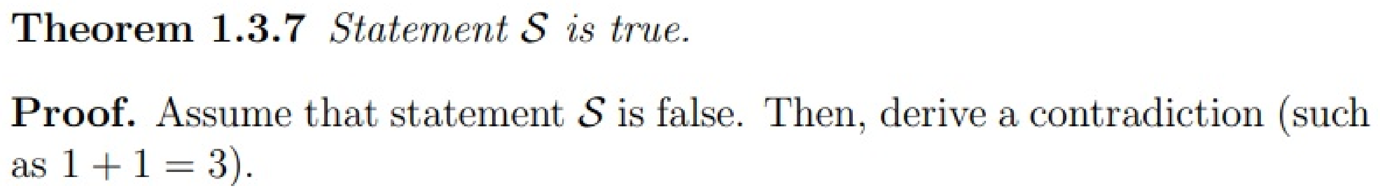


Theorem 1.3.6 *There exist irrational numbers x and y such that x^y is rational.*

Proof by Contradiction

A form of proof that establishes the truth or the validity of a proposition, by showing that assuming the proposition to be false leads to a contradiction

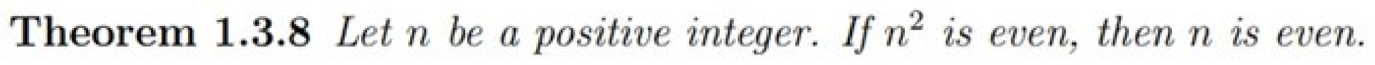
Example



Theorem 1.3.7 *Statement $\mathcal{S}$ is true.*

Proof. Assume that statement $\mathcal{S}$ is false. Then, derive a contradiction (such as $1 + 1 = 3$).

Example



Theorem 1.3.8 *Let n be a positive integer. If n^2 is even, then n is even.*

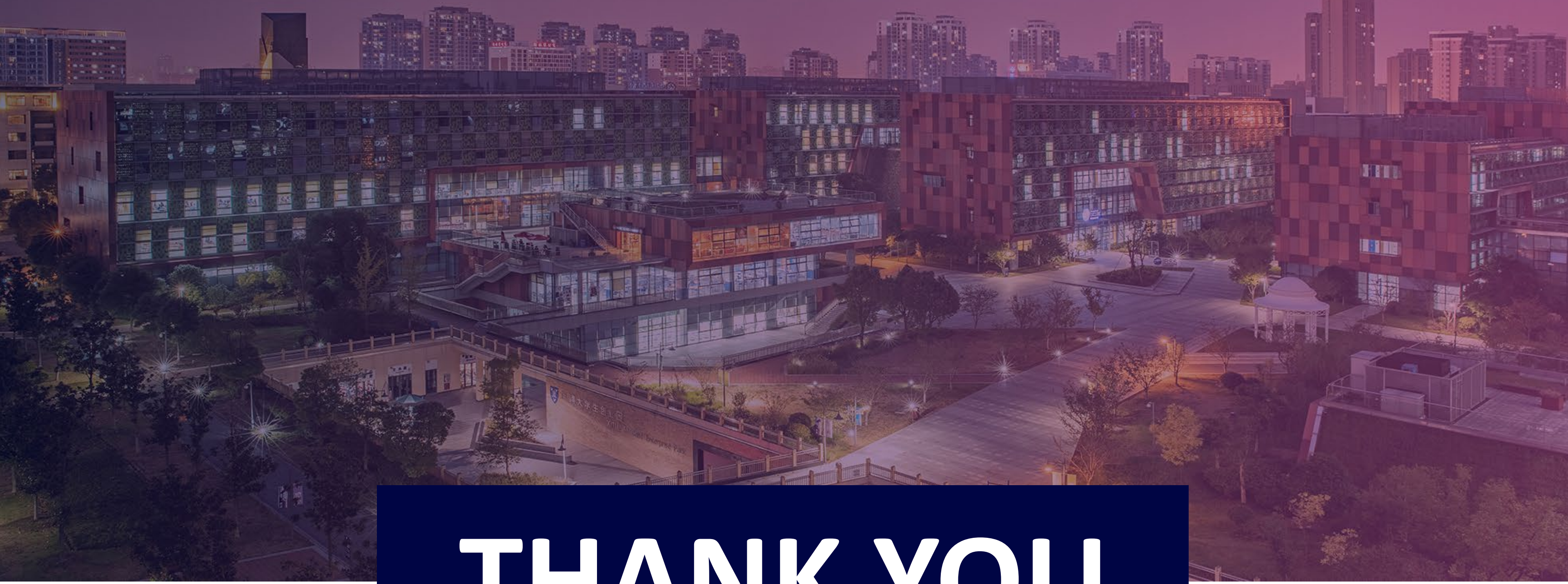
Proof by Induction

A mathematical proof technique. It is essentially used to prove that a statement P(n) holds for every natural number n = 0, 1, 2, 3, ... ;

Example

For all positive integers n, we have

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



THANK YOU

Xi'an Jiaotong-Liverpool University
西交利物浦大学

XJTLU | SCHOOL OF FILM AND TV ARTS