Languages and Computation (COMP2049/AE2LAC)

Introduction

Dr. Tianxiang Cui

tianxiang.cui@nottingham.edu.cn

Course Overview

- Instructor
  - Tianxiang Cui
  - Office: PMB(SEB) 323
  - Contact email: tianxiang.cui@nottingham.edu.cn

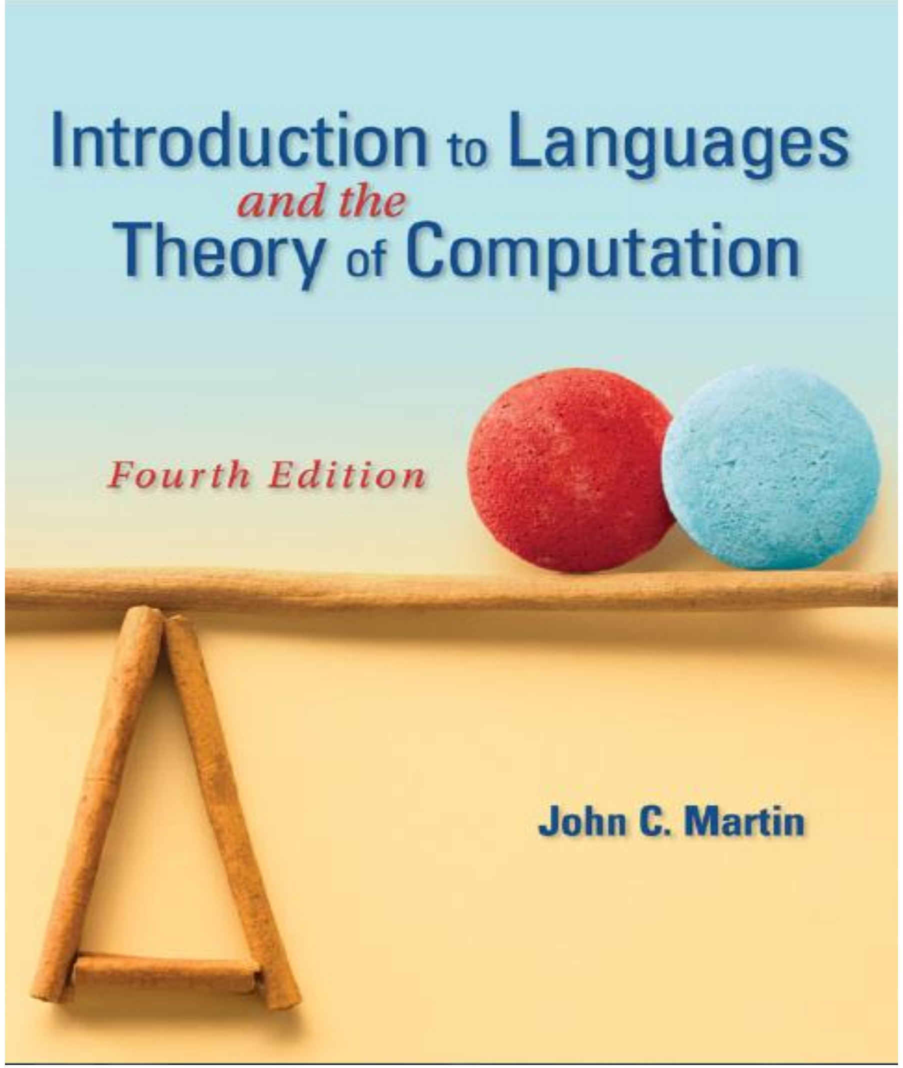
- Lectures
  - **Week 27-30 (26th Feb-23rd March), Week 35-38 (23rd April-18th May)**
  - Tuesday 9:00-10:00 DB-A04
  - Thursday 16:00-18:00 DB-A04

Assessment

- 1 Coursework
  - 25% of marking
  - Primarily theoretical problems
  - Details to follow on Moodle page
  - You will learn a lot from the coursework – learning by doing

- Exam
  - 75% of marking
  - 2-hour written exam

Course Textbook and References



Main textbook:

- Introduction to Languages and The Theory of Computation (4th Edition), John C. Martin, 2011

Other useful books:

- An introduction to formal languages and automata (Peter Linz)
- Introduction to the Theory of Computation (Michael Sipser)

Also, please remember that you should consult various other references

- Any books on Formal Languages and Automata Theory and related subjects in the library
- Any online resource on these subjects
- I will mention some references during the course, but you need to learn to do your own research as a university student and not limit yourself to the slides and the textbook

Attendance Policy

- It is not compulsory to attend the lectures
  - My primary role is to trick you into learning something, you can also do your own research
  - …but, those who attend the lectures may find it easier to follow the module without falling behind
  - Therefore, you are strongly encouraged to attend the lectures
- However, if you do attend, you must pay attention
  - Please do not disrupt the lectures by talking to others, using your mobile phone, laptop, etc ☺
  - As it is a theoretical course, it is easy to get lost…

Lectures, Slides and Textbook

- The lecture slides are mainly based on the textbook
- During each lecture, I may also present some materials that **may not appear** on the slides
- You will be tested on all of these
- The lecture slides are not a substitute for your textbook
  - After the lecture, please **read the textbook** and do more research on the topics that you find more difficult
  - You don't have to read every word of the book
  - Put your knowledge into practice by working out the exercises in the book
  - Or even better, come up with your own examples. That is more productive

Feedback

- It is your responsibility to ask questions
  - Please give me feedback on any issue: things you like, don’t like, find hard, find easy, don’t understand, etc.
- If you want to discuss something in details, please send me an email
  - I am happy to arrange a meeting

Aims of the Course

- To familiarize you with key Computer Science concepts in central areas like
  - Automata Theory
  - Formal Languages
  - Models of Computation
  - Complexity Theory
- To equip you with tools with wide applicability in the fields of CS and IT

Content

1. Mathematical models of computation, such as:
   - Finite automata
   - Pushdown automata
   - Turing machines
2. How to specify formal languages?
   - Regular expressions
   - Context free grammars
   - Context sensitive grammars
3. The relation between 1 and 2

Topics

- We will cover the following topics:
  - Finite State Machines and Regular Languages
    - Deterministic Finite Automata (DFA)
    - Nondeterministic Finite Automata (NFA)
    - Equivalence between NFA and DFA
    - Equivalence between Regular Languages and Finite Automata
  - Pushdown Automata (PDA) and Context Free Languages (CFL)
    - The Pumping Lemma for context-free languages
    - Equivalence between CFLs and PDAs
  - Turing Machines and Recursively Enumerable Languages
    - Decidability
    - Undecidable Problems

Why Study All This?

- Formal languages and automata have lots of applications in CS and IT. Some examples:
  - Specification of programming languages (grammar, syntax)
  - Implementation of programming language processors
  - Encoding documents
    - Extensible Markup Language (XML)
    - Document Type Definition (DTD)
  - Finding words and patterns in large bodies of text, e.g. in web pages
  - Verification of systems with finite number of states, e.g. communication protocols

Why Study All This?

- Automata are essential for the study of the limits of computation. Deep theoretical questions with big practical implications. Two key issues:
- What can a computer do at all ?
  - **Decidability**
- What can a computer do efficiently ?
  - **Time and space Complexity**
  - Time Complexity classes
    - **P**olynomial time (P)
    - **N**ondeterministic, **P**olynomial time (NP)

Example

- Imagine you're the lead developer for a new web browser. It obviously needs the capability to run JavaScript
- To make your product stand out from the competition, your boss proposes you implement a termination check:
  - Any non-terminating JavaScript programs can then be rejected, without being run
- If you succeed, your salary will be doubled. But if you fail, you'd have to look for a new job
- **Should you accept?**

Example

- Consider the following program. Does it terminate for all values of n $\geq$ 1?

```
while (n > 1) {
    if even(n) {
        n = n / 2;
    } else {
        n = n * 3 + 1;
    }
}
```

Example

- Not as easy to answer as it might first seem
- Say we start with n = 7, for example:
  - 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
- In fact, for all numbers that have been tried (**a lot!**), it does terminate . . .
- But, no one has ever been able to **prove** that it always terminates!
- This is known as the **Halting Problem**

Halting Problem

- The following important decidability result should then perhaps not come as a total surprise:

  It is impossible to write a program that decides if another, arbitrary, program terminates (halts) or not

- What might be surprising is that it is possible to prove such a result. This was first done by the British mathematician Alan Turing using Turing Machines

- The Halting problem is **undecidable** over Turing Machines
  - One of the first problems to be proved undecidable

Alan Turing

- Alan Turing (1912-1954) – English computer scientist, mathematician
- Introduced an abstract model of computation, **Turing Machine**, to give a precise definition of what problems that can be solved by a computer
- Instrumental in the success of British code breaking efforts during WWII
  - Help to crack Enigma machine code used by German naval

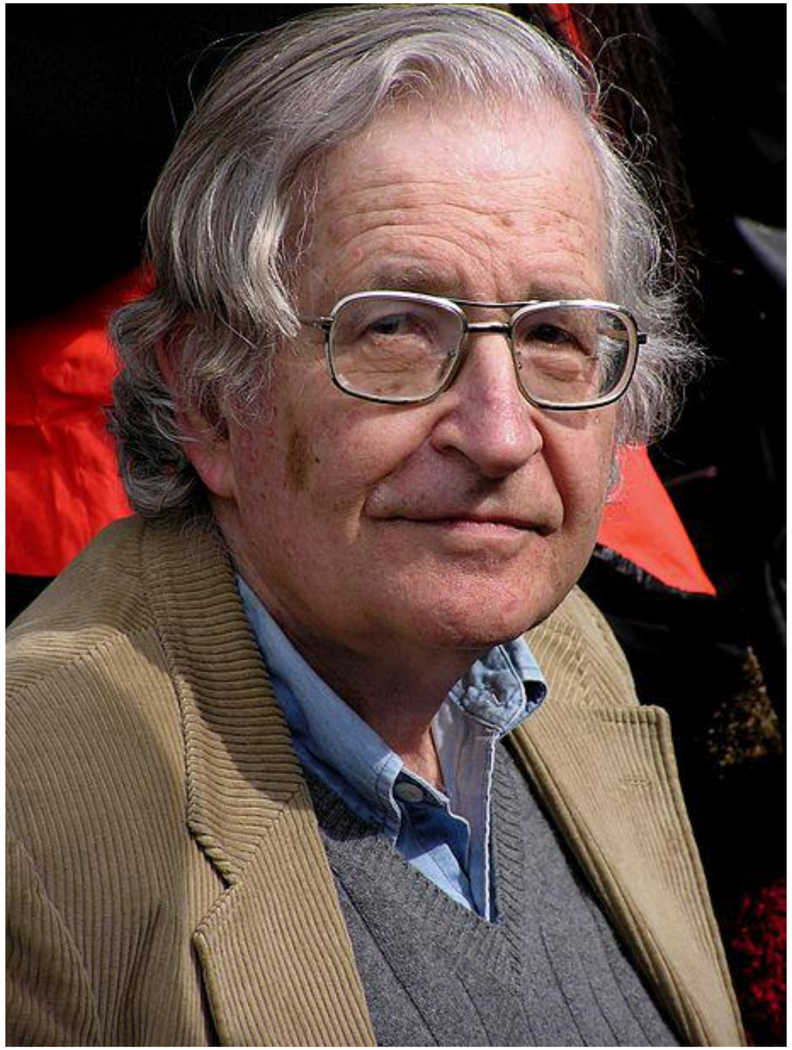
Alan Turing


"BENEDICT CUMBERBATCH IS OUTSTANDING"
"THE BEST BRITISH FILM OF THE YEAR"
★★★★★
"AN INSTANT CLASSIC"
★★★★★
"A SUPERB THRILLER"
★★★★
★★★★ ★★★★
THE
BENEDICT CUMBERBATCH
KEIRA KNIGHTLEY
IMITATION GAME
BASED ON THE INCREDIBLE TRUE STORY
IN CINEMAS NOVEMBER 14

Noam Chomsky

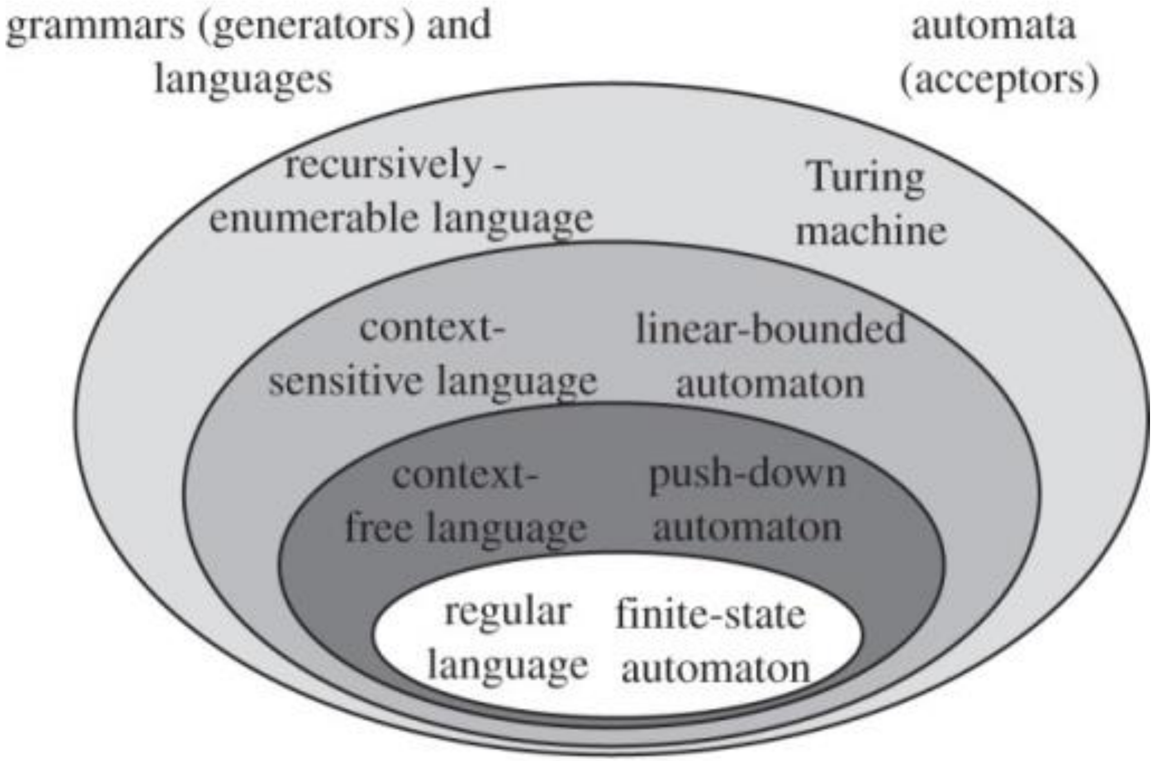
- Noam Chomsky (born in 1928, age 90)
- American linguist who introduced **Context Free Grammars** in an attempt to describe natural languages formally
- Also introduced the **Chomsky Hierarchy** which classifies grammars and languages and their descriptive power

Noam Chomsky



The Chomsky Hierarchy

- Chomsky introduced the hierarchy of grammars in his study of natural languages
  - Type 0 – recursively enumerable languages
  - Type 1 – context sensitive languages
  - Type 2 – context free languages
  - Type 3 – regular languages



the traditional Chomsky hierarchy

Languages

- The terms **language** and **word** are used in a strict technical sense in this course
  - A **language** is a (possibly infinite) set of words
  - A **word** is a finite sequence (or string) of symbols
- ε denotes the empty word, the sequence of zero symbols
- The term **string** is often used interchangeably with the term **word**

Symbols and Alphabets

- What is a symbol, then?
- Anything, but it has to come from an alphabet Σ which is a **finite** set
- Examples:
  - The binary alphabet $\Sigma = \{0, 1\}$
  - The set of all lower-case letters $\Sigma = \{a, b, \ldots, z\}$
  - The set of all ASCII characters
- ε, the empty word, is **never** a symbol of an alphabet

Languages: Example

- Alphabet
- $\Sigma = \{a, b\}$

- Words
- ε, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, . . .

- Languages
- $\emptyset$, $\{\varepsilon\}$, $\{a\}$, $\{b\}$, $\{a, aa\}$, $\{\varepsilon, a, aa, aaa\}$,
- $\{a^n \mid n \geq 0\}$,
- $\{a^n b^n \mid n \geq 0, n \text{ even}\}$

Note the distinction between ε, $\emptyset$, and $\{\varepsilon\}$!

Exercises

- Is the set of natural numbers, N, a possible alphabet? Why/Why not?
- What about the set of all natural numbers smaller than some given number n?
- Suggest an alphabet of the set {0010, 00000000, 01110011}.
- List some words over your alphabet?

Powers of An Alphabet

- If Σ is an alphabet, we can express the set of all strings of a **certain length** from that alphabet by using the **exponential notation**:
  - Σ^k: the set of strings of length k, each of whose is in Σ
- Example:
  - Σ^0 : $\{\varepsilon\}$, regardless of what alphabet Σ is, as ε is the only string of length 0
  - If $\Sigma = \{0, 1\}$
    - $\Sigma^1 = \{0, 1\}$
    - $\Sigma^2 = \{00, 01, 10, 11\}$
    - $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
  - Note: confusion between Σ and Σ^1
    - Σ is an alphabet; its members 0 and 1 are symbols
    - Σ^1 is a set of strings; its members are strings (each one of length 1)

Kleene Star

- Given an alphabet Σ we define the set Σ^* as set of words (or strings) over Σ
- The symbol $*$ is called **Kleene star** and is named after the mathematician and logician Stephen Cole Kleene
- Inductive definition (define the elements in a set in terms of other elements in the set)
  - The empty word $\varepsilon \in \Sigma^*$
  - Given a symbol $x \in \Sigma$ and a word $w \in \Sigma^*$, $xw \in \Sigma^*$
  - These are all elements in Σ^*
  - Alternatively, $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
  - E.g. if $\Sigma = \{a\}$, $\Sigma^* = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$
- Is Σ^* always infinite? Always non-empty?

Example

- If $\Sigma = \{0,1\}$, some elements of Σ^* are:
  - ε (the empty word, Σ^0)
  - Σ^1?
  - Σ^2?
  - Σ^3?
  - …
- We are just applying the inductive definition
- Note: although there are infinitely many words in Σ^* (when $\Sigma \neq \emptyset$), each word has a **finite** length

Concatenation of Words

- An important operation on Σ^* is **concatenation**
  - Let $w = w_1 \ldots w_k$ and $y = y_1 \ldots y_k$ be two strings over some alphabet Σ. Then the concatenation of w and y (in symbols $w \cdot y$, or just wy) is the string $w_1 \ldots w_k y_1 \ldots y_k$
  - i.e. the concatenated string is formed by making a copy of w and following it by a copy of y

- Example:
  - w = 01101 and y = 110, then wy = 01101110 and yw = 11001101
  - w = ab and y = ba, then wy = abba and yw = baab

- For any string w, the equations $\varepsilon w = w\varepsilon = w$ hold

- Concatenation is associative
  - i.e. $w(yv) = (wy)v$, so we will simply write wyv

- By w^k we denote w concatenated with itself k times

Language Revisited

- The notion of a language L of a set of words over an alphabet Σ can now be made precise:
  - A language L over an alphabet Σ is a subset of Σ^*, i.e. $L \subseteq \Sigma^*$
  - Or equivalently, $L \in \boldsymbol{P}(\Sigma^*)$
    - $\boldsymbol{P}$ is the power set, i.e. the set of all subsets of Σ^*

Languages: Example

- The set {0010, 00000000, ε} is a language over Σ = {0,1} (Finite or infinite?)
- The language of all strings consisting of n 0s followed by n 1s over Σ = {0,1} ($n \geq 0$) (Finite or infinite?)

  {ε, 01, 0011, 000111, . . .}

- The set of strings of 0s and 1s with an equal number of each over Σ = {0,1} (Finite or infinite?)

  {ε, 01, 10, 0011, 0101, 1001, . . .}

Languages: Example

- $\emptyset$, the empty language, is a language over any alphabet
- {ε}, the language consisting of only the empty string, is also a language over any alphabet
- English language
  - The collection of legal English words is a set of strings over the alphabet that consists of all the letters
- C programming language
  - The alphabet is a subset of the ASCII characters and programs are subset of strings that can be formed from this alphabet
- Note: a language over Σ need not include strings with all symbols of Σ. Thus, a language over Σ is also a language over any alphabet that is a superset of Σ

Languages Union

- The union of two languages L and M, denoted $L \cup M$, is the set of strings that are in either L, or M, or both
- Example:

$$L = \{001, 10, 111\}$$

$$M = \{\varepsilon, 001\}$$

$$L \cup M = \{\varepsilon, 001, 10, 111\}$$

Languages Concatenation

- Concatenation of words can be extended to languages. The concatenation of languages L and M, denoted $L.M$ or just LM , is the set of strings that can be formed by taking any string in L and concatenating it with any string in M. Formally, we have:

$$LM = \{uv \mid u \in L \wedge v \in M\}$$

- Example:

$$L = \{\varepsilon, \text{a}, \text{aa}\}$$

$$M = \{\text{b}, \text{c}\}$$

$$LM = \{uv \mid u \in \{\varepsilon, \text{a}, \text{aa}\} \wedge v \in \{\text{b}, \text{c}\}\}$$

$$= \{\varepsilon\text{b}, \varepsilon\text{c}, \text{ab}, \text{ac}, \text{aab}, \text{aac}\}$$

$$= \{\text{b}, \text{c}, \text{ab}, \text{ac}, \text{aab}, \text{aac}\}$$

Languages Concatenation

- Concatenation of languages is also associative:

$$L(MN) = (LM)N$$

- For any language L, the equations $L\{\varepsilon\} = L = \{\varepsilon\}L$ hold
- How about concatenating the **empty language (∅)** to any language?

It yields the **empty language (∅)**

$$L\emptyset = \emptyset = \emptyset L$$

Languages Concatenation

- Concatenation distributes through set union:

$$L(M \cup N) = LM \cup LN$$

$$(L \cup M)N = LN \cup MN$$

- But **not** through intersection

$$L(M \cap N) \neq LM \cap LN$$

- **Counterexample**:

$$L = \{\varepsilon, \text{a}\},\ M = \{\varepsilon\},\ N = \{\text{a}\}$$

$$L(M \cap N) = L\emptyset = \emptyset$$

$$LM \cap LN = \{\varepsilon, \text{a}\} \cap \{\text{a}, \text{aa}\} = \{\text{a}\}$$

Languages Closure

- Exponent notation is used to denote iterated concatenation:

$$L^1 = L$$

$$L^2 = LL$$

$$L^3 = LLL$$

- By definition: $L^0 = \{\varepsilon\}$ (for any language, **incl. ∅**)
- The **closure** of a language L is denoted L^* and represents the set of those strings that can be formed by taking **any number** of strings from L, possibly with repetitions (i.e., the same string may be selected more than once) and concatenating all of them

$$L^* = \bigcup_{n=0}^{\infty} L^n$$

Languages Closure: Example

- If $L = \{0, 1\}$ then L^* is all strings of 0 and 1
- If $L = \{0, 11\}$ then L^* consists of strings of 0 and 1 such that the 1 come in pairs, e.g., 011, 11110, 11011, etc. But **not 01011 or 101**.

Languages Membership

- Fundamental question for a language L: $w \in L$?
- L finite: Easy! (Enumerate L and check)
- L infinite: ? We need:
  - A **finite** (and preferably concise) formal **description** of L
  - An algorithmic **method to decide** if $w \in L$ given a suitable description
- Various approaches to achieve this
  - Will be key a theme throughout the module

Concluding Remarks

- The Chomsky Hierarchy
- Symbol
- Alphabet
- Word
- Language
- Operators on languages
  - Union
  - Concatenation
  - Closure