COMP2012/G52LAC
Languages and Computation
Lecture 1

Administrative Details and Introduction

Venanzio Capretta and Henrik Nilsson

University of Nottingham

Finding People and Information

- Venanzio Capretta
  Room C05
- Henrik Nilsson
  Room A08
- Moodle
- Main module web page:
  `www.cs.nott.ac.uk/~nhn/COMP2012`
- Moodle forum!

Aims of the Course

Aims of the Course

- To familiarize you with key Computer Science ***concepts*** in central areas:
  - Automata Theory
  - Formal Languages
  - Models of Computation
  - Complexity Theory

Aims of the Course

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

Aims of the Course

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

Draws from: COMP1001/G51MCS
Feeds into: COMP3012/G53CMP, COMP3001/G53COM, COMP4001/G54FOP

Organization (1)

- ***Lectures:***
  - Two 1 h lectures per week (back to back).
  - Detailed but provisional schedule available on the module web page.

Organization (1)

- ***Lectures:***
  - Two 1 h lectures per week (back to back).
  - Detailed but provisional schedule available on the module web page.
- ***Coursework:***
  - 3 problem sets.
  - Made available via the module web page.
  - Best 2 counts.
  - Deadlines: 27/2, 20/3, 10/4.
  - Released a week prior to submission deadline.

Organization (2)

- ***Assessment:***
  - Coursework, 25 %
  - 2 hour written examination, 75 %

Organization (2)

- ***Assessment:***
  - Coursework, 25 %
  - 2 hour written examination, 75 %
- However, ***resits*** are by 100 % written examination (standard School policy)

Literature (1)

- Main reference: John E. Hopcroft, Rajeev Motwani, & Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 3rd edition*, Pearson, 2007.

Literature (1)

- Main reference: John E. Hopcroft, Rajeev Motwani, & Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 3rd edition*, Pearson, 2007.
- Alternative/complement: Linz. *An Introduction to Formal Languages and Automata, 6th edition*, Jones & Bartlett Publishers, 2017.

Literature (1)

- Main reference: John E. Hopcroft, Rajeev Motwani, & Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 3rd edition*, Pearson, 2007.
- Alternative/complement: Linz. *An Introduction to Formal Languages and Automata, 6th edition*, Jones & Bartlett Publishers, 2017.
- The lecture notes by Altenkirch, Capretta, Nilsson (January 2019).
  Available via the module web page.

Literature (2)

- Supplementary material; e.g., slides, sample program code.
  (Available via the module web page.)

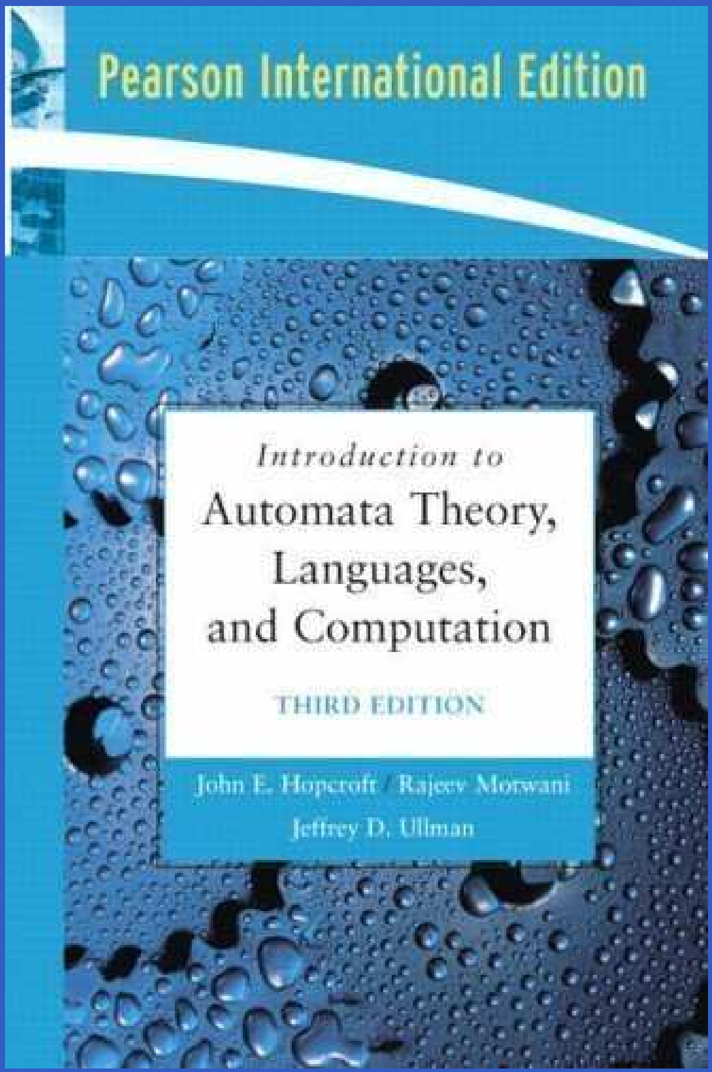
Literature (2)

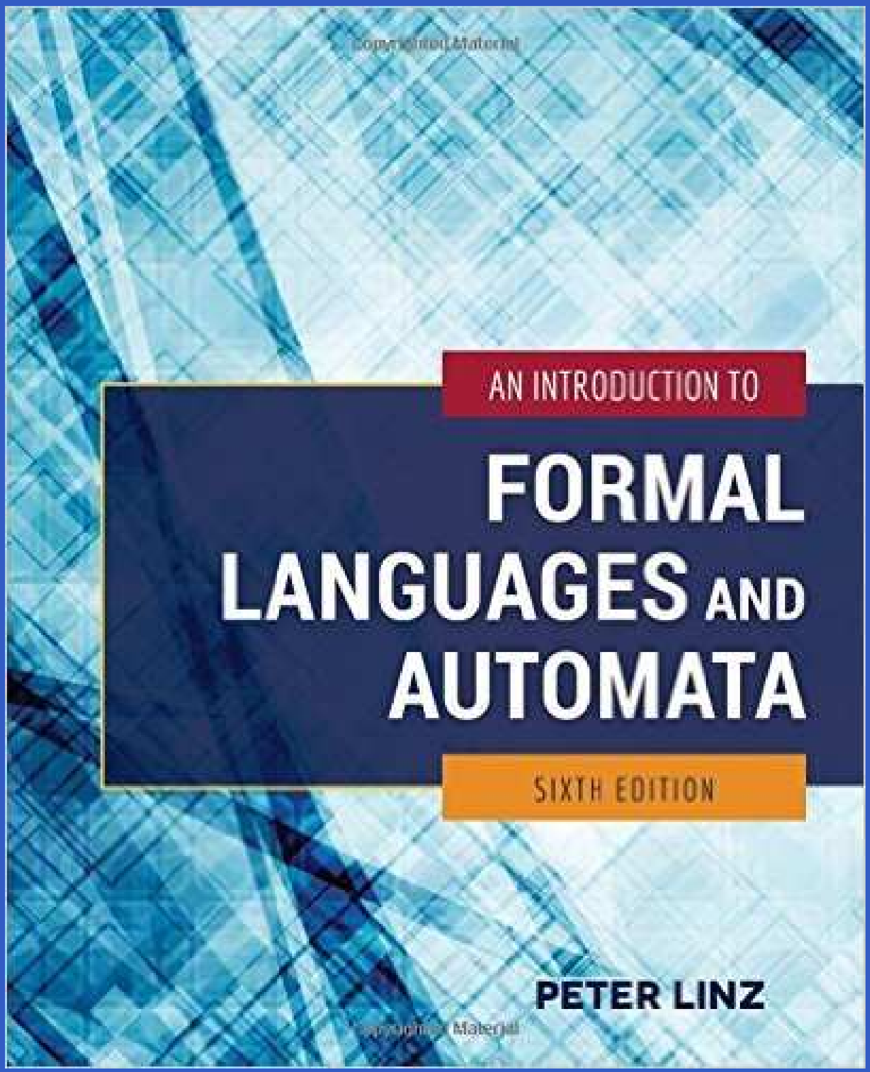
- Supplementary material; e.g., slides, sample program code.
  (Available via the module web page.)
- Your own notes from the lectures!

Literature (2)

- Supplementary material; e.g., slides, sample program code.
  (Available via the module web page.)
- Your own notes from the lectures!
- The lecture schedule contains detailed lecture-by-lecture references to the literature.

Literature (3)





The Lecture Notes

The Lecture Notes

- Comprehensive, typeset lecture notes.
  (At present around 160 pages.)

The Lecture Notes

- Comprehensive, typeset lecture notes. (At present around 160 pages.)
- Carefully aligned with the lectures.

The Lecture Notes

- Comprehensive, typeset lecture notes. (At present around 160 pages.)
- Carefully aligned with the lectures.
- Covers everything said in the lectures (and more).

The Lecture Notes

- Comprehensive, typeset lecture notes. (At present around 160 pages.)
- Carefully aligned with the lectures.
- Covers everything said in the lectures (and more).
- Exercises with detailed model solutions (in response to student feedback).

The Lecture Notes

- Comprehensive, typeset lecture notes. (At present around 160 pages.)
- Carefully aligned with the lectures.
- Covers everything said in the lectures (and more).
- Exercises with detailed model solutions (in response to student feedback).
- The exercises are quite similar to typical coursework problems.

Your Own Notes

You are strongly encourage to take your own notes as well during lectures because:

Your Own Notes

You are strongly encourage to take your own notes as well during lectures because:

- Lectures may provide an alternative perspective, use different examples, etc.

Your Own Notes

You are strongly encourage to take your own notes as well during lectures because:

- Lectures may provide an alternative perspective, use different examples, etc.
- Research shows that note taking significantly aids learning.

Your Own Notes

You are strongly encourage to take your own notes as well during lectures because:

- Lectures may provide an alternative perspective, use different examples, etc.
- Research shows that note taking significantly aids learning.

Taking relevant notes is a lot easier if you familiarise yourself with the relevant parts of the typeset lecture notes prior to each lecture!

Content (1)

Content (1)

- The notion of a formal language

Content (1)

- The notion of a formal language
- Description of different classes of languages:
  - Regular expressions
  - Grammars

Content (1)

- The notion of a formal language
- Description of different classes of languages:
  - Regular expressions
  - Grammars
- Recognition of different classes of languages:
  - Finite Automata
  - Push Down Automata

Content (1)

- The notion of a formal language
- Description of different classes of languages:
  - Regular expressions
  - Grammars
- Recognition of different classes of languages:
  - Finite Automata
  - Push Down Automata
- Applications: Scanning and Parsing

Content (2)

Leading to:

Content (2)

Leading to:

- General notions of computation:
  - Turing machines
  - Lambda calculus

Content (2)

Leading to:

- General notions of computation:
  - Turing machines
  - Lambda calculus
- Fundamental questions such as
  - What can be computed at all?
  - What can be computed efficiently?

Example: Languages and Grammars (1)

Consider the following Java fragment:

```
class Foo {
    int n;
    void printNSqrd() {
        System.out.println(n * n);
    }
}
```

Example: Languages and Grammars (1)

Consider the following Java fragment:

```
class Foo {
    int n;
    void printNSqrd() {
        System.out.println(n * n);
    }
}
```

- Fundamentally a string of characters.

Example: Languages and Grammars (1)

Consider the following Java fragment:

```
class Foo {
     int n;
     void printNSqrd() {
          System.out.println(n * n);
     }
}
```

- Fundamentally a string of characters.
- But lots of structure to valid Java code, e.g.:
  - Keywords, identifiers, operators
  - Nesting; e.g. method inside class

Example: Languages and Grammars (2)

- How to describe the set of strings that are valid Java?

Example: Languages and Grammars (2)

- How to describe the set of strings that are valid Java?
- Given a string, how to determine if it is a valid Java program or not?

Example: Languages and Grammars (2)

- How to describe the set of strings that are valid Java?
- Given a string, how to determine if it is a valid Java program or not?
- How to recover the structure of a Java program from a “flat” string?

Example: Languages and Grammars (2)

- How to describe the set of strings that are valid Java?
- Given a string, how to determine if it is a valid Java program or not?
- How to recover the structure of a Java program from a “flat” string?

We will study:

Example: Languages and Grammars (2)

- How to describe the set of strings that are valid Java?
- Given a string, how to determine if it is a valid Java program or not?
- How to recover the structure of a Java program from a “flat” string?

We will study:

- ***Regular expressions*** and ***grammars***: precise descriptions of languages.

Example: Languages and Grammars (2)

- How to describe the set of strings that are valid Java?
- Given a string, how to determine if it is a valid Java program or not?
- How to recover the structure of a Java program from a “flat” string?

We will study:

- ***Regular expressions*** and ***grammars***: precise descriptions of languages.
- Various kinds of ***automata***: decide if a string belongs to a language or not.

Noam Chomsky (1)

Noam Chomsky (1928–):

Noam Chomsky (1)

Noam Chomsky (1928–):

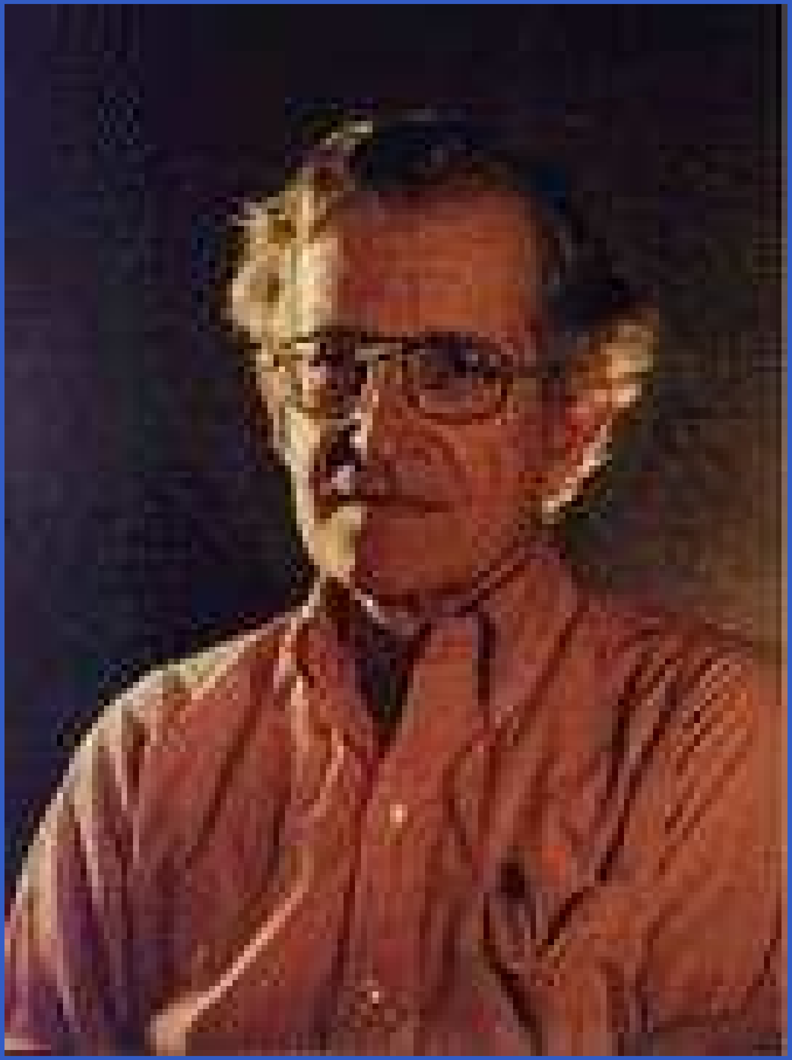
- American linguist who introduced ***Context Free Grammars*** in an attempt to describe natural languages formally.

Noam Chomsky (1)

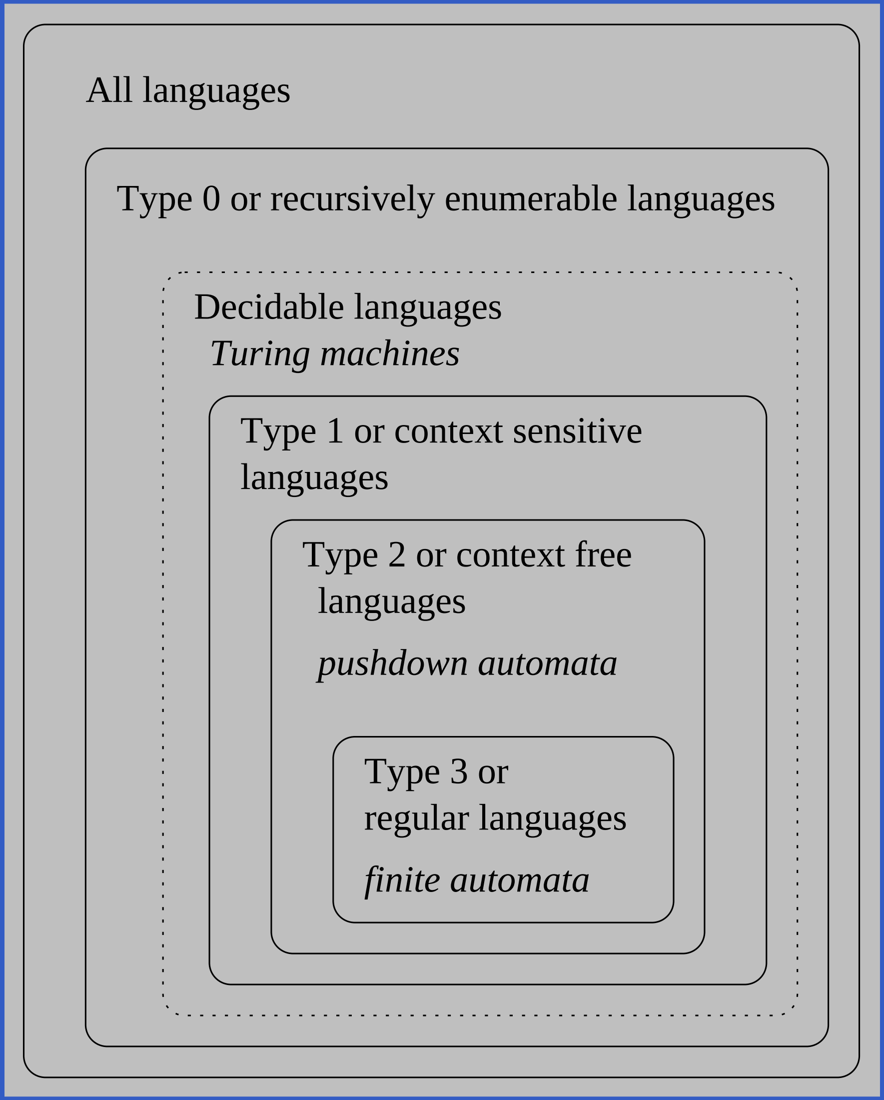
Noam Chomsky (1928–):

- 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 (2)



The Chomsky Hierarchy



Example: The Halting Problem (1)

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: The Halting Problem (2)

Not as easy to answer as it might first seem.

Example: The Halting Problem (2)

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

Example: The Halting Problem (2)

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

The sequence involved is known as the ***hailstone sequence*** and ***Collatz conjecture*** says that the number 1 will always be reached.

Example: The Halting Problem (2)

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

The sequence involved is known as the ***hailstone sequence*** and ***Collatz conjecture*** says that the number 1 will always be reached.

In fact, for all numbers that have been tried (***up to*** 2^{60} ***!***), it does terminate . . .

Example: The Halting Problem (2)

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

The sequence involved is known as the ***hailstone sequence*** and ***Collatz conjecture*** says that the number 1 will always be reached.

In fact, for all numbers that have been tried (***up to*** 2^{60} ***!***), it does terminate . . .

. . . but so far, ***no proof***! (See e.g. Wikipedia.)

Example: The Halting Problem (3)

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

Example: The Halting Problem (3)

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.***

Example: The Halting Problem (3)

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.***

This was first proved by the British mathematician ***Alan Turing*** using Turing Machines.

Alan Turing (1)

Alan Turing (1912–1954):

Alan Turing (1)

Alan Turing (1912–1954):

- Introduced an abstract model of computation, ***Turing Machines*** (1936), to give a precice definition of what problems are “effectively calculable” (can be solved mechanically).

Alan Turing (1)

Alan Turing (1912–1954):

- Introduced an abstract model of computation, ***Turing Machines*** (1936), to give a precice definition of what problems are “effectively calculable” (can be solved mechanically).
- Instrumental in the success of British code breaking efforts during WWII.

Alan Turing (1)

Alan Turing (1912–1954):

- Introduced an abstract model of computation, ***Turing Machines*** (1936), to give a precice definition of what problems are “effectively calculable” (can be solved mechanically).
- Instrumental in the success of British code breaking efforts during WWII.
- PhD student of Alonzo Church

Alan Turing (2)

Example: the λ-Calculus

- λ-calculus is a theory of pure functions:

$$(\lambda x.x)(\lambda y.y)$$

Example: the λ-Calculus

- λ-calculus is a theory of pure functions:

$$(\lambda x.x)(\lambda y.y)$$

- Functional programming languages like Haskell implements the λ-calculus.

Example: the λ-Calculus

- λ-calculus is a theory of pure functions:

$$(\lambda x.x)(\lambda y.y)$$

- Functional programming languages like Haskell implements the λ-calculus.
- Both the Turing machine and the λ-calculus are ***universal models of computation***: equivalent in capabilities.

Alonzo Church (1)

Alonzo Church (1903–1995):

- Alan Turing's PhD advisor

Alonzo Church (1)

Alonzo Church (1903–1995):

- Alan Turing's PhD advisor
- Introduced the ***λ-calculus*** (1936) to give a precise definition of what problems are "effectively calculable".

Alonzo Church (1)

Alonzo Church (1903–1995):

- Alan Turing's PhD advisor
- Introduced the ***λ-calculus*** (1936) to give a precise definition of what problems are "effectively calculable".
- Church-Turing thesis: What is "effectively calculable" is exactly what can be computed by a Turing machine.

Alonzo Church (2)

Example: P versus NP (1)

"Can every problem whose solution can be ***checked*** quickly by a computer also be ***solved*** quickly by a computer?"

Example: P versus NP (1)

"Can every problem whose solution can be ***checked*** quickly by a computer also be ***solved*** quickly by a computer?"

- Likely the most famous open problem in computer science, dating back to the 1950s.

Example: P versus NP (1)

"Can every problem whose solution can be ***checked*** quickly by a computer also be ***solved*** quickly by a computer?"

- Likely the most famous open problem in computer science, dating back to the 1950s.
- "Quickly" here means in time proportional to a ***polynomial*** in the size of the problem.

Example: P versus NP (1)

"Can every problem whose solution can be ***checked*** quickly by a computer also be ***solved*** quickly by a computer?"

- Likely the most famous open problem in computer science, dating back to the 1950s.
- "Quickly" here means in time proportional to a ***polynomial*** in the size of the problem.
- There is an abundance of important problems where solutions can be checked quickly, but where the best ***known*** algorithm for finding a solution is ***exponential*** in the size of the problem.

Example: P versus NP (2)

Subset sum problem: Does some non-empty subset of given set of integers sum to zero?

Example: P versus NP (2)

Subset sum problem: Does some non-empty subset of given set of integers sum to zero? E.g. given $\{3, -2, 8, -5, 4, 9\}$, the non-empty subset $\{-5, -2, 3, 4\}$ sums to 0.

Example: P versus NP (2)

Subset sum problem: Does some non-empty subset of given set of integers sum to zero?
E.g. given $\{3, -2, 8, -5, 4, 9\}$, the non-empty subset $\{-5, -2, 3, 4\}$ sums to 0.

- Easy to check proposed solution: just add all numbers.

Example: P versus NP (2)

Subset sum problem: Does some non-empty subset of given set of integers sum to zero? E.g. given $\{3, -2, 8, -5, 4, 9\}$, the non-empty subset $\{-5, -2, 3, 4\}$ sums to 0.

- Easy to check proposed solution: just add all numbers. (How long would it take for set of size n?)

Example: P versus NP (2)

Subset sum problem: Does some non-empty subset of given set of integers sum to zero? E.g. given $\{3, -2, 8, -5, 4, 9\}$, the non-empty subset $\{-5, -2, 3, 4\}$ sums to 0.

- Easy to check proposed solution: just add all numbers. (How long would it take for set of size n?)
- But for finding a solution, no better way known than essentially trying each possible subset in turn.

Example: P versus NP (2)

Subset sum problem: Does some non-empty subset of given set of integers sum to zero? E.g. given $\{3, -2, 8, -5, 4, 9\}$, the non-empty subset $\{-5, -2, 3, 4\}$ sums to 0.

- Easy to check proposed solution: just add all numbers. (How long would it take for set of size n?)
- But for finding a solution, no better way known than essentially trying each possible subset in turn. (How long would it take for set of size n? How many subsets are there?)

Introduction to Languages

The terms ***language*** and ***word*** are used in a strict technical sense in this course:

Introduction to 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.

Introduction to 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.

Introduction to 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.

Introduction to 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?

Symbols and Alphabets

What is a symbol, then?

Anything, but it has to come from an ***alphabet*** Σ which is a ***finite*** set.

Symbols and Alphabets

What is a symbol, then?

Anything, but it has to come from an ***alphabet*** Σ which is a ***finite*** set.

A common (and important) instance is $\Sigma = \{0, 1\}$.

Symbols and Alphabets

What is a symbol, then?

Anything, but it has to come from an ***alphabet*** Σ which is a ***finite*** set.

A common (and important) instance is $\Sigma = \{0, 1\}$.

ϵ, the empty word, is ***never*** a symbol of an alphabet.

Languages: Examples

alphabet	$\Sigma = \{a, b\}$
words	?

Languages: Examples

alphabet	$\Sigma = \{a, b\}$
words	$\epsilon, a, b, aa, ab, ba, bb,$

Languages: Examples

alphabet	$\Sigma = \{a, b\}$
words	$\epsilon, a, b, aa, ab, ba, bb,$
	$aaa, aab, aba, abb, baa, bab, \ldots$

Languages: Examples

alphabet	$\Sigma = \{a, b\}$
words	$\epsilon, a, b, aa, ab, ba, bb,$
	$aaa, aab, aba, abb, baa, bab, \ldots$
languages	?

Languages: Examples

alphabet	$\Sigma = \{a, b\}$
words	$\epsilon, a, b, aa, ab, ba, bb,$
	$aaa, aab, aba, abb, baa, bab, \ldots$
languages	$\emptyset, \{\epsilon\}, \{a\}, \{b\}, \{a, aa\},$

Languages: Examples

alphabet	$\Sigma = \{a, b\}$
words	$\epsilon, a, b, aa, ab, ba, bb,$
	$aaa, aab, aba, abb, baa, bab, \ldots$
languages	$\emptyset, \{\epsilon\}, \{a\}, \{b\}, \{a, aa\},$
	$\{\epsilon, a, aa, aaa\},$

Languages: Examples

alphabet	$\Sigma = \{a, b\}$
words	$\epsilon, a, b, aa, ab, ba, bb,$
	$aaa, aab, aba, abb, baa, bab, \ldots$
languages	$\emptyset, \{\epsilon\}, \{a\}, \{b\}, \{a, aa\},$
	$\{\epsilon, a, aa, aaa\},$
	$\{a^n \vert n \geq 0\},$

Languages: Examples

alphabet	$\Sigma = \{a, b\}$
words	$\epsilon, a, b, aa, ab, ba, bb,$
	$aaa, aab, aba, abb, baa, bab, \ldots$
languages	$\emptyset, \{\epsilon\}, \{a\}, \{b\}, \{a, aa\},$
	$\{\epsilon, a, aa, aaa\},$
	$\{a^n \mid n \geq 0\},$
	$\{a^n b^n \mid n \geq 0, n \text{ even}\}$

Languages: Examples

alphabet	$\Sigma = \{a, b\}$
words	$\epsilon, a, b, aa, ab, ba, bb,$
	$aaa, aab, aba, abb, baa, bab, \ldots$
languages	$\emptyset, \{\epsilon\}, \{a\}, \{b\}, \{a, aa\},$
	$\{\epsilon, a, aa, aaa\},$
	$\{a^n \mid n \geq 0\},$
	$\{a^n b^n \mid n \geq 0, n \text{ even}\}$

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

All Words Over an Alphabet (1)

Given an alphabet Σ we define the set Σ^* as set of words (or sequences) over Σ:

- The empty word $\epsilon \in \Sigma^*$.
- given a symbol $x \in \Sigma$ and a word $w \in \Sigma^*$, $xw \in \Sigma^*$.
- These are all elements in Σ^*.

This is called an ***inductive definition***.

All Words Over an Alphabet (1)

Given an alphabet Σ we define the set Σ^* as set of words (or sequences) over Σ:

- The empty word $\epsilon \in \Sigma^*$.
- given a symbol $x \in \Sigma$ and a word $w \in \Sigma^*$, $xw \in \Sigma^*$.
- These are all elements in Σ^*.

This is called an ***inductive definition***.

Is Σ^* always non-empty?

All Words Over an Alphabet (1)

Given an alphabet Σ we define the set Σ^* as set of words (or sequences) over Σ:

- The empty word $\epsilon \in \Sigma^*$.
- given a symbol $x \in \Sigma$ and a word $w \in \Sigma^*$, $xw \in \Sigma^*$.
- These are all elements in Σ^*.

This is called an ***inductive definition***.

Is Σ^* always non-empty? Always infinite?

All Words over an Alphabet (2)

Example: Given $\Sigma = \{0, 1\}$, some elements of Σ^* are

- ϵ (the empty word)
- $0, 1$
- $00, 10, 01, 11$
- $000, 100, 010, 110, 001, 101, 011, 111$
- $\ldots$

All Words over an Alphabet (2)

Example: Given $\Sigma = \{0, 1\}$, some elements of Σ^* are

- ϵ (the empty word)
- 0, 1
- 00, 10, 01, 11
- 000, 100, 010, 110, 001, 101, 011, 111
- $\ldots$

We are just applying the inductive definition.

All Words over an Alphabet (2)

Example: Given $\Sigma = \{0, 1\}$, some elements of Σ^* are

- ϵ (the empty word)
- 0, 1
- 00, 10, 01, 11
- 000, 100, 010, 110, 001, 101, 011, 111
- $\ldots$

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!

Examples of Languages (1)

Some examples of languages:

Examples of Languages (1)

Some examples of languages:

- The set $\{0010, 00000000, \epsilon\}$ is a language over $\Sigma = \{0, 1\}$.

Examples of Languages (1)

Some examples of languages:

- The set $\{0010, 00000000, \epsilon\}$ is a language over $\Sigma = \{0, 1\}$.
  This is an example of a ***finite*** language.

Examples of Languages (1)

Some examples of languages:

- The set $\{0010, 00000000, \epsilon\}$ is a language over $\Sigma = \{0, 1\}$.
  This is an example of a ***finite*** language.
- The set of words with odd length over $\Sigma = \{1\}$. (Finite or infinite?)

Examples of Languages (1)

Some examples of languages:

- The set $\{0010, 00000000, \epsilon\}$ is a language over $\Sigma = \{0, 1\}$.
  This is an example of a ***finite*** language.
- The set of words with odd length over $\Sigma = \{1\}$. (Finite or infinite?)
- The set of words that contain the same number of 0s and 1s is a language over $\Sigma = \{0, 1\}$. (Finite or infinite?)

Examples of Languages (2)

- The set of palindromes (words that read the same forwards and backwards, like `abba`) is a language for any alphabet.

Examples of Languages (2)

- The set of palindromes (words that read the same forwards and backwards, like `abba`) is a language for any alphabet.
- The set of correct Java programs. This is a language over the set of UNICODE characters.

Examples of Languages (2)

- The set of palindromes (words that read the same forwards and backwards, like `abba`) is a language for any alphabet.
- The set of correct Java programs. This is a language over the set of UNICODE characters.
- The set of programs that, if executed successfully on a Windows machine, prints the text “Hello World!” in a window. This is a language over $\Sigma = \{0, 1\}$.

Language Membership

Fundamental question for a language L: $w \in L$?

Language Membership

Fundamental question for a language L: $w \in L$?

- L finite:

Language Membership

Fundamental question for a language L: $w \in L$?

- L finite: ?

Language Membership

Fundamental question for a language L: $w \in L$?

- L finite: Easy! (Enumerate L and check)

Language Membership

Fundamental question for a language L: $w \in L$?

- L finite: Easy! (Enumerate L and check)
- L infinite:

Language Membership

Fundamental question for a language L: $w \in L$?

- L finite: Easy! (Enumerate L and check)
- L infinite: ?

Language 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.

Language 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.

Language 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.