Formal Languages and Automata Theory

TDDD14/TDDD85

2019

Today

- Some administrative info
- Overview of the course
- Some theory: Strings and languages

Administrative Info

TDDD14 vs. TDDD85

TDDD14 and TDDD85 are the same course in practice.

Only difference:
(Same exam), but a few more points required to pass for TDDD14.

Teachers

Christer Bäckström: Examiner, lecturer, teaching assistent

Jonas Wallgren: Course assistant, lecturer, teaching assistant

Organization

- 16 Lectures
  Christer and Jonas will alternate.

- 9 Tutorials (Problem solving sessions)
  Jonas and Christer

Examination

- Two sets of homework

- One written exam

You must pass both these!

Literature

- Course book:
  Michael Sipser *Introduction to the Theory of Computation*

- (The previous book, Dexter Kozen *Automata and Computability*, can be used if you already have it.)

- A compendium for the tutorials
  Download from web page

- Various other material at the web page
  Note! This material may change during the course.

Questions?

Overview of Course

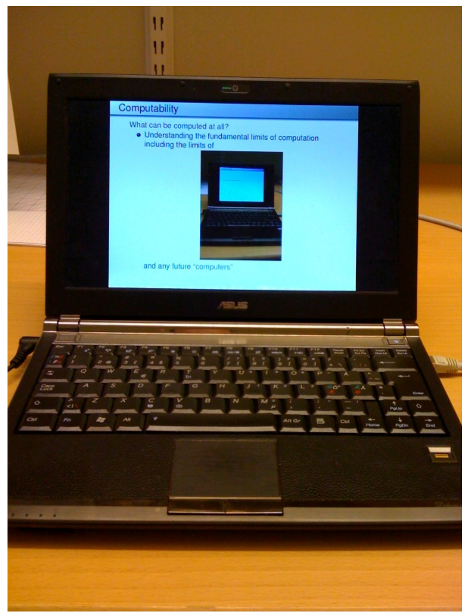
What Can We Compute?

What Can We Compute?

With very limited memory?

What Can We Compute?

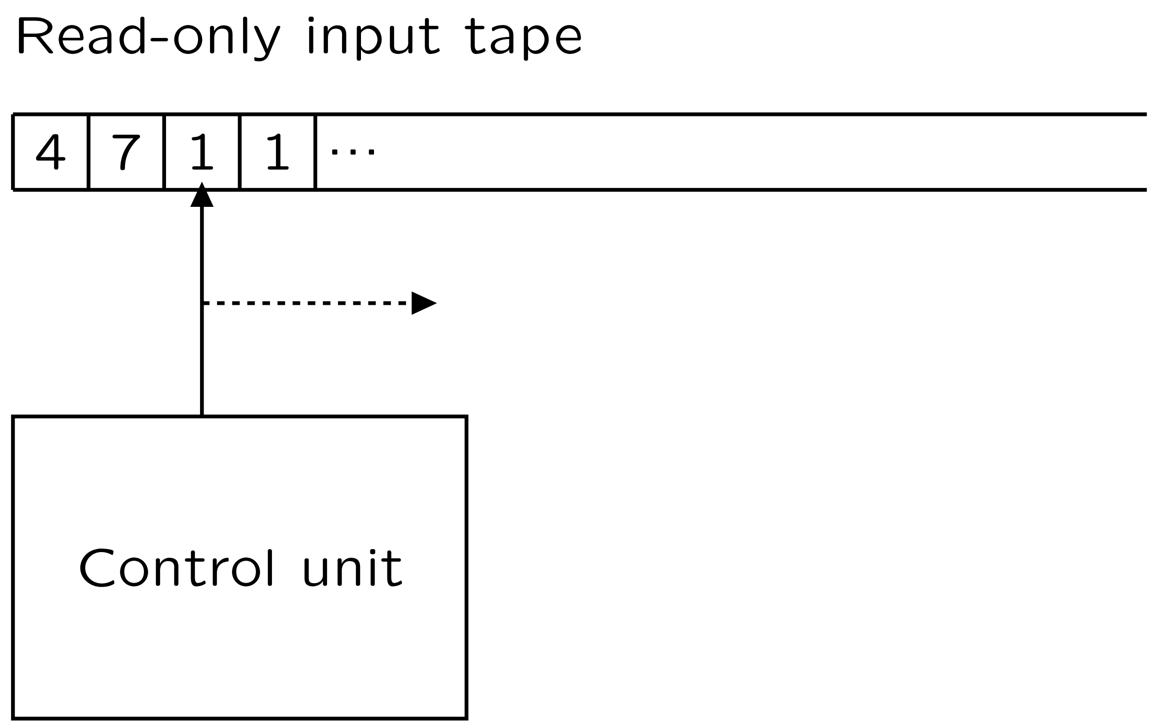
With unlimited memory?



(That is, unlimited in the theoretical model, meaning sufficient memory in practice.)

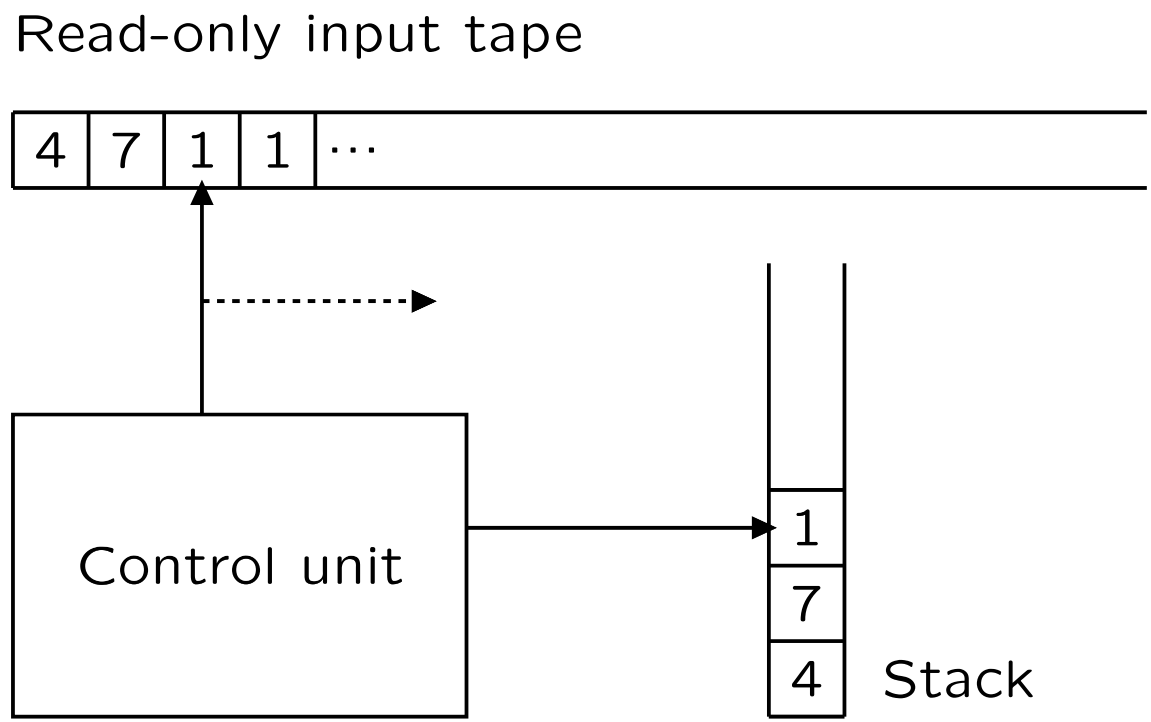
We will study three models of computation

Finitite Automata (FA)



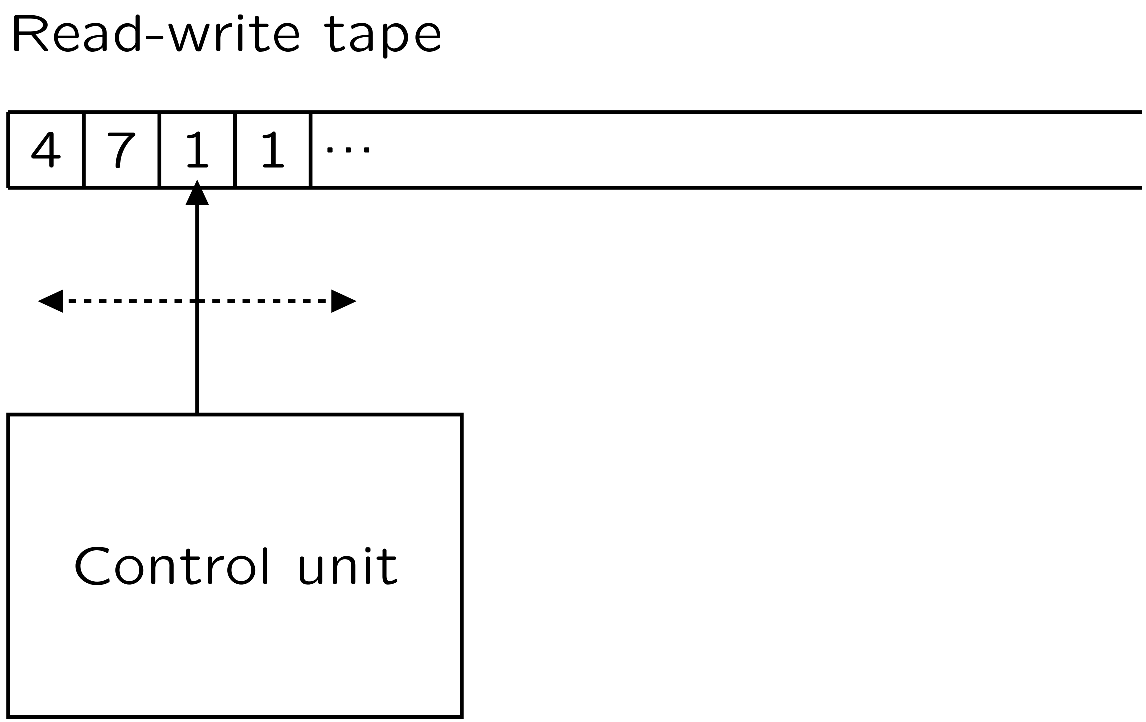
We can read the input sequentially. The control unit has a finite number of states, which is the only memory.

Pushdown Automata (PDA)



The PDA is an FA that has unlimited memory with restricted access in the form of a stack.

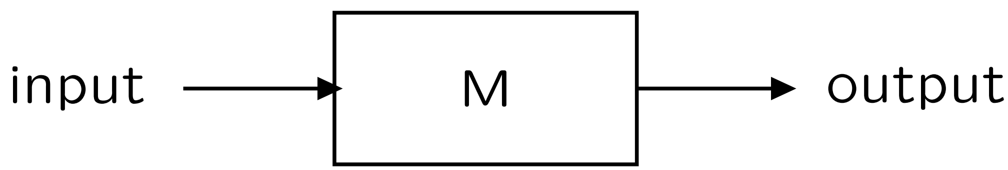
Turing Machine (TM)



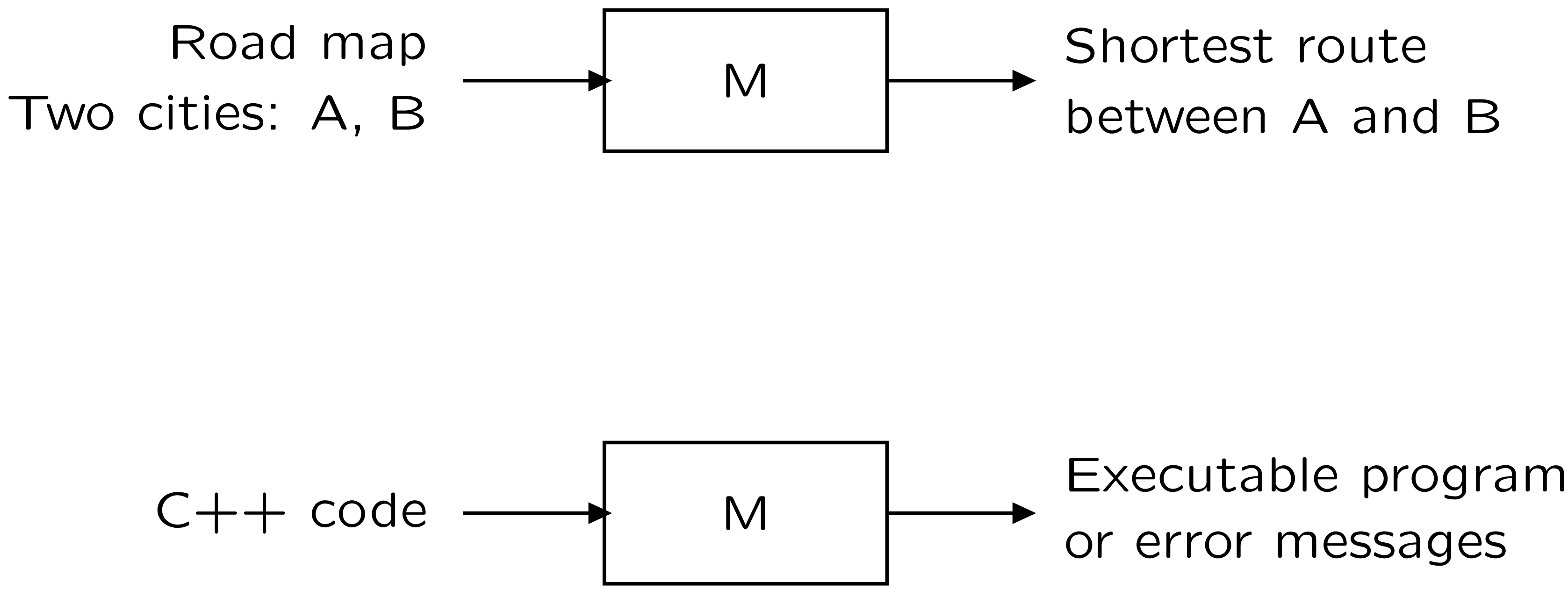
The TM uses the (infinite) tape as a read-write memory with unrestricted access.

Decision Problems

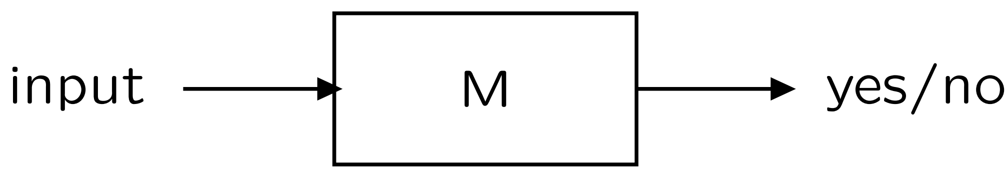
We usually want to compute something.



The output is a function of the input.

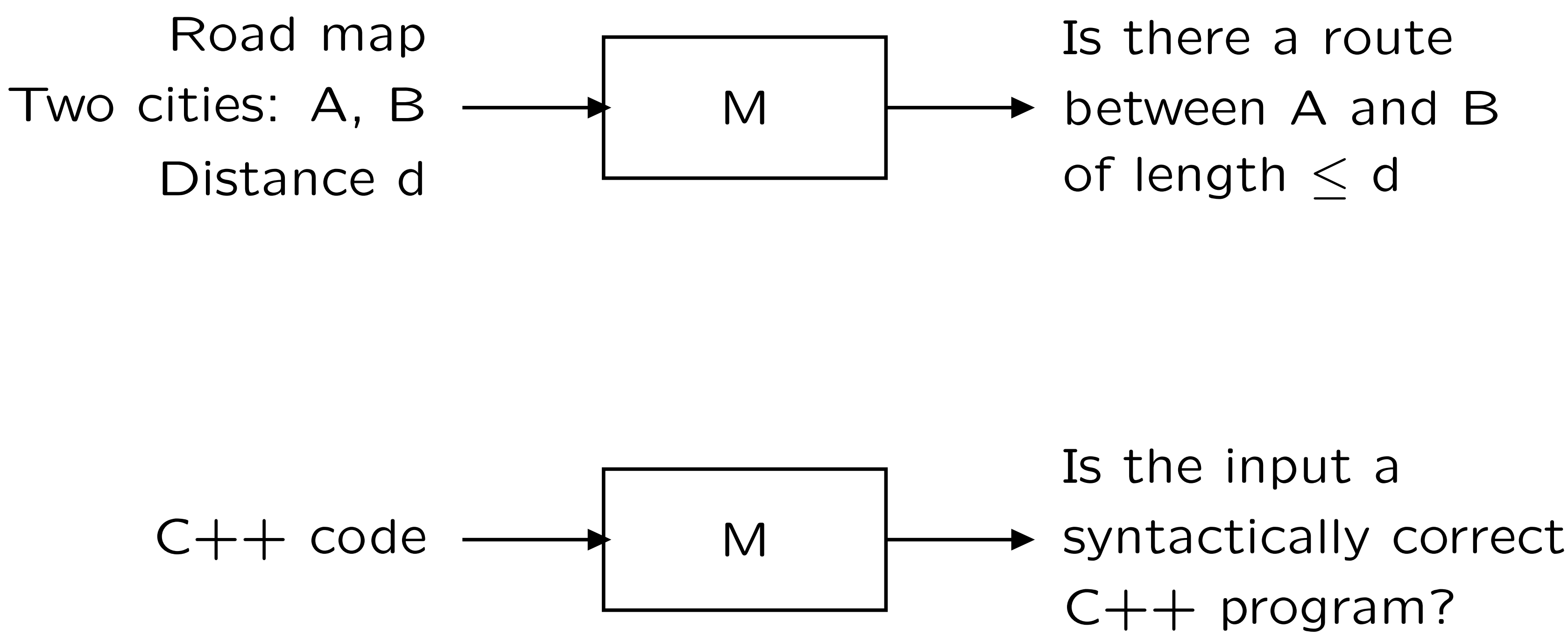

Road map
Two cities: A, B
M
Shortest route
between A and B
C++ code
M
Executable program
or error messages

In theory we often focus on decision problems, which have a yes/no answer.



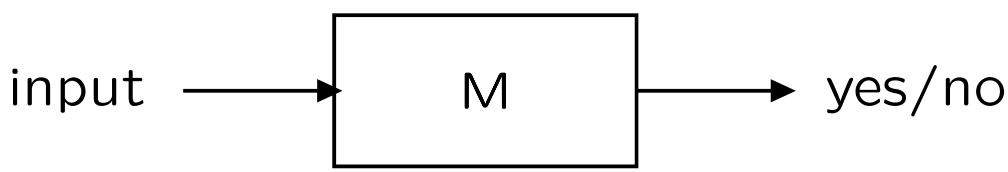
- Simpler to analyse

- Most of the interesting properties remain

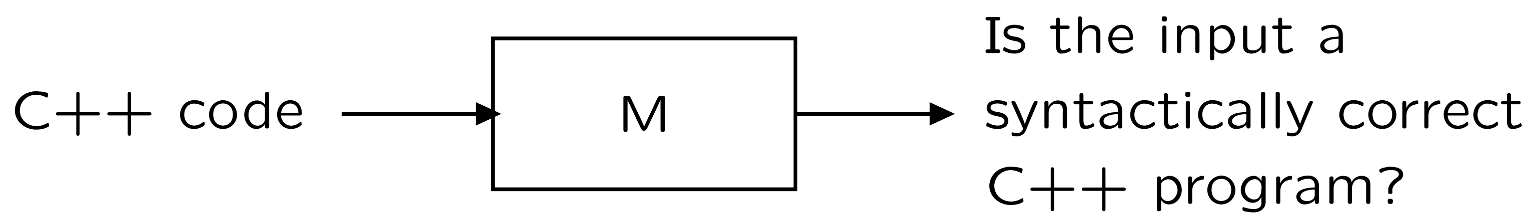

Road map
Two cities: A, B
Distance d
M
Is there a route
between A and B
of length ≤ d
C++ code
M
Is the input a
syntactically correct
C++ program?

Formal Languages

Let M be a deciding machine.



The language L(M) of M is the set of all inputs where M answers yes.



L(M) is the set of all syntactically correct C++ programs.

Correspondance Between Machines and Languages

Machines with various limitations correspond to different formal languages.

$$
\begin{array}{ccc}
\text{TM} & \Leftrightarrow & \text{semi-decidable languages} \\
\uparrow & & \uparrow \\
\text{PDA} & \Leftrightarrow & \text{Context free languages} \\
\uparrow & & \uparrow \\
\text{FA} & \Leftrightarrow & \text{Regular languages}
\end{array}
$$

Natural languges like english and swedish are not formal languages.

Why study limited machines/languages?

- Have nice properties

- Simpler to analyse and use

Finite automata/regular languages

Regular languages are very restricted, but also very fast to use.
Very useful for

- search expressions
- filters in web browsers
- and much more...

Pushdown automata/context-free languages

Most programming languages today are CFL.

- Fast and easy to check the syntax
- Much more powerful than regular languages
- Avoids many types of errors

The earliest programming languages in the 1950's had no formal principles for the syntax (Fortran and Cobol).

The Fortran bug that (almost) crashed a satellite:

Intended code:

DO I = 1,100 *Loop I from 1 to 100*

Actual code:

DO I = 1.100

Blanks are not significant in Fortran and variables are implicitly declared, so no errror.

Interpreted as:

DOI = 1.100 *Assign value 1.100 to variable DOI*

The unlimited case

The Turing Machine (TM) is the theoretical model of computation.

Everything that is computable can be computed on a Turing machine and vice versa.

That is, the Turing machine is a theoretical model of both current and future computers.

Can we compute everything that can be formalized?

The Halting Problem:

Write a computer program as follows:

Input:

An arbitrary computer program P (e.g. in C++)

Output:

Yes, if P terminates on all inputs

No, if P goes into an infinite loop on some input(s).

Such a program cannot exist. There will always be input programs P where it is not possible to give a yes/no answer.

That is, there are things that cannot be computed!

Some Basic Theory and Notation

Basic Notions: Strings and Languages

- An *alphabet* is a finite set of symbols (denoted Σ, Γ)

- A *string* over Σ is a finite sequence of symbols from Σ

Example:
1010 is a string over $\Sigma_1 = \{0, 1\}$
$theory$ is a string over $\Sigma_2 = \{a, b, c, \ldots, z\}$

- The *length* of a string w (denoted $|w|$) is the number of symbols it contains.
  $|1010| = 4$, $|theory| = 6$

- The *empty string* (denoted ϵ) is the string of length 0

- The *concatenation* of strings x and y is written xy.
  $x = red$ and $y = fox$ gives $xy = redfox$.

- x^k denotes the concatenation of x with itself k times.
  $01^50 = 0111110$,
  $x^2y = redredfox$.

- A *language* is a set of strings over an alphabet

- Σ^* is the language consisting of all strings over Σ
  Let $\Sigma = \{a, b\}$.
  Then $\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots\}$

Note that a language is an ordinary set, so we have the usual set operations:

- Union and intersection:
  $A \cup B = \{x \mid x \in A \; or \; x \in B\}$,
  $A \cap B = \{x \mid x \in A \; and \; x \in B\}$

- Complement: $\overline{A} = \{x \in \Sigma^* \mid x \notin A\}$
  (Note: Kozen uses $\tilde{}A$ for complement.)

- Concatenation: $AB = \{xy \mid x \in A \; and \; y \in B\}$

- A^k denotes the concatenation of A with itself k times (note: $A^0 = \{\epsilon\}$)

- Star: $A^* = A^0 \cup A^1 \cup A^2 \cup \ldots$

Examples of Languages:

- the set of odd binary numbers
- the set of prime numbers
- the set of syntactically correct Java programs
- the set of positive integer solutions to $x^n + y^n = z^n$ for $n > 2$
- the set of true mathematical statements