UNIVERSITÀ DELLA CALABRIA

Theoretical Computer Science

Lecture 1 – Introduction

Contents and Methods

Overview

The course covers the fundamental aspects of theoretical computer science, such as the notions of *decidability* and *computational complexity*.

The **first module (Part A)** introduces students to the basics of Theoretical Computer Science, with particular attention to formal languages, Turing machines, computability theory, undecidability and incompleteness in first-order logic.

The **second module (Part B)** provides the basics of computational complexity theory. Students will be able to distinguish among tractable, untractable and presumably untractable problems. They will be also able to formally determine the exact computational complexity of a number of problems which are of practical interest.

Course organization

The course consists of 56 lessons organized as follows:

- 16 ***lectures*** delivered by Prof. Marco Manna (Part A)
- 16 ***lectures*** delivered by Prof. Georg Gottlob (Part B)
- 24 ***practice sessions*** delivered by Dr. Cinzia Marte (12 Part A + 12 Part B)

Each lesson includes:

- A ***set of slides*** tailored for the specific content covered in the lesson
- ***Representative questions and exercises*** aimed at self-assessment, designed to test comprehension and allow independent progress evaluation

To qualify for the final exam, students are required to attend at least 70% of each module:

- A minimum of 12 *lectures* per module
- A minimum number of 9 *practice sessions* per module

Part A – Decidability and Logic (lectures)

1. Introduction
2. Strings and Languages
3. Regular Languages and Automata Theory
4. Turing Machines
5. Multiple tape Turing Machines
6. The Universal Turing Machine
7. The RAM Machine
8. Semi-decision procedures (1/2)
9. Semi-decision procedures (2/2)
10. Properties of D and SD
11. Undecidability of the halting problem
12. Turing Reductions and Mapping Reductions
13. Boolean Logic
14. First-order Logic
15. Undecidability in First-order Logic
16. Undecidability and Incompleteness in Number Theory

Part A – Decidability and Logic (practice sessions)

1. Mathematical Background
2. Regular Expressions and Finite State Machines
3. Turing Machines
4. Turing Machines
5. Multiple tape Turing Machines
6. The RAM Machine
7. Semi-decision procedures
8. Semi-decision procedures
9. Reductions
10. Reductions
11. Boolean Logic and First-Order Logic
12. Summary Exercises

Part B – Computational Complexity (lectures)

1. Introduction to complexity
2. Measuring the computational complexity
3. Deterministic complexity classes
4. Polynomial-time problems and properties
5. Polynomial-time complete problems
6. Nondeterministic Turing Machine
7. The complexity class NP
8. The Cook-Levin Theorem
9. More NP-complete problems
10. The polynomial hierarchy
11. Inside the Polynomial Hierarchy
12. Circuit Complexity
13. Beyond the Polynomial Hierarchy
14. Deterministic Logarithmic Space
15. Nondeterministic Logarithmic Space
16. Conjunctive query evaluation

Part B – Computational Complexity (practice sessions)

1. Polynomial-time problems
2. Membership in NP via certificate and verifier
3. Membership in NP via reductions
4. Membership in NP via guess and check
5. Completeness in NP
6. Completeness in NP
7. Completeness in NP
8. QBFs and Circuits
9. Problems in DP
10. Logarithmic Space
11. Conjunctive query evaluation
12. Completeness in P

Textbooks and further references

PRIMARY TEXT:

- Elaine A. Rich. Automata, Computability, and Complexity: Theory and Applications. Prentice Hall, 2008.

RECOMMENDED BOOKS:

- Carlo Ghezzi, and Dino Mandrioli. Informatica teorica. Città-Studi Edizioni, 1989.
- Giorgio Ausiello, Fabrizio D'Amore, and Giorgio Gambosi. Linguaggi, Modelli, Complessità. Franco Angeli Ed., 2003.
- Harry Lewis, and Christos Papadimitriou. Elements of the Theory of Computation (2nd Edition). Prentice-Hall, 199
- J. Glenn Brookshear. Informatica - Una panoramica generale, 11/ed. Pearson Italia, 2010.

EXTRA:

- Christos H. Papadimitriou. Computational Complexity. Addison Wesley, 1994.
- Jon Kleinberg, and Éva Tardos. Algorithm Design. Addison-Wesley, 2005.
- Sanjeev Arora, and Boaz Barak. Complexity Theory: A Modern Approach. Cambridge University Press, 2009.

Methods and criteria for learning assessment

Students' knowledge, skills, and expertise are assessed through an ***integrated written and oral examination***. For each module, the exam consists of:

- 15 multiple-choice questions (all written)
  - 10 about ***basic knowledge*** (to assess students' understanding of fundamental concepts, definitions, and principles in TCS)
  - 5 about ***practical skills*** (to evaluate students' ability to solve specific problems or tasks effectively)
- 3 open-ended questions (1 written and 2 oral)
  - about ***comprehensive understanding*** (to verify students' capacity to articulate and discuss broader topics or themes in TCS in a clear, accurate, and concise manner)

Open-ended questions are graded as follows:

- -1 (incorrect/unsatisfactory), 0 (no response), 1 (satisfactory), 2 (good), or 3 (excellent)

Methods and criteria for learning assessment

Let x denote either "A" or "B". Accordingly, let $M_{\text{part\_}x}$ denote the count of correct multiple-choice questions about part x, and let $O^{i}_{\text{part\_}x}$ denote the grade obtained for the i-th open-ended question of part x. We calculate the $\mathbf{FinalGrade}$ as follows:

1. Calculate the grade $G_{\text{part\_}x}$ of each part x:
   - $G_{\text{part\_}x} = 9 + M_{\text{part\_}x} + O^{1}_{\text{part\_}x} + O^{2}_{\text{part\_}x} + O^{3}_{\text{part\_}x}$
2. Compute the arithmetic mean of the grades for the two modules:
   - $\mathbf{FinalGrade} = \text{round}\left(\frac{G_{\text{part\_A}} + G_{\text{part\_B}}}{2}\right)$
3. To pass the exam and effectively obtain the $\mathbf{FinalGrade}$:
   - Each $M_{\text{part\_}x}$ must be at least 9
   - Each $G_{\text{part\_}x}$ must be at least 18
4. If $\mathbf{FinalGrade} \geq 30$, then the committee considers whether to award honors (30 cum laude).

Example of question about "*basic knowledge*"

Which of the following statements is syntactically correct and can be used to define Hany?

- ☐ Hany = {<M,w> | ∃w s.t. M(w) = h}
- ☑ Hany = {<M> | ∃w s.t. M(w) = h}
- ☐ Hany = {<M> | ∃w s.t. M(w) = ↗}
- ☐ Hany = {<M> | ∃w and ∃M s.t. M(w) ∈ {y,n,h} }

Example of question about "*practical skills*"

From (q2,a)→q1 and (q2,b)→q3, determine the *yield-in-one-step* of the configuration (q2, babbaa).

- ☐ (q3, babbaa)
- ☐ (q1, babba)
- ☐ (q2, abbaa)
- ☑ (q3, abbaa)
- ☐ (q1, bbbaa)

Example of question about “*comprehensive understanding*”

Explain the Universal Turing Machine in 1000 characters or less, covering its role, its importance, and its fundamental properties.

The Universal Turing Machine (UTM) is a foundational concept in computer science, pivotal for its ability to simulate any TM. Its role in computability theory is crucial, showcasing the universality of computation, despite a potential computational overhead. Generally, the UTM is a 3-tape TM denoted by U. Essentially, U takes in input on Tape 1 the pair <M, w>, encoding a TM M and string w over M’s alphabet. Upon initiation, <M> shifts to Tape 2, leaving Tape 1 with input string <w>, mimicking M’s tape over w. Tape 2 stores TM M’s encoding, serving as the “program” executed by U. Tape 3 encodes an M state, essential for mimicking M over w. A typical alphabet of U is

$$\Sigma_U = \{\ (\ ,\ )\ ,\ \mathtt{a}\ ,\ \mathtt{q}\ ,\ \mathtt{y}\ ,\ \mathtt{n}\ ,\ \mathtt{h}\ ,\ \mathtt{0}\ ,\ \mathtt{1}\ ,\ ;\ \ ,\ \rightarrow\ ,\ \leftarrow\}.$$

The encoding of an M’s state q4 may look like `q0100`, whereas the encoding of an M’s symbol `c` may look like `a011`. The encoding of a transition of the form $(q, \square, s, \mathtt{a}, \rightarrow)$ may look like the string $\mathtt{(q01;a00;q00;a01;}\rightarrow\mathtt{)}$ over Σ_U.

Practical protocols

	Marco Manna	Cinzia Marte	Georg Gottlob
Email	marco.manna@unical.it	cinzia.marte@unical.it	georg.gottlob@unical.it
Office hours	**Wednesday 16.30 – 17.30** *(right after class)*		**By appointment**

- Link to the Team of the course (code **`hxctevq`**): https://bit.ly/3Isoz2J
  - "Class Materials" are accessible by the menu "File" of the "General" channel
- Link for attendance tracking
  - https://forms.gle/XLSPLoDb21ti5iAs5

Why Studying the Theory of Computation?

Real Problems and Ambitions

- How can we:
  - Optimize cost-saving strategies for efficient goods delivery?
  - Navigate through a maze to reach the exit?
  - Develop winning strategies for various games?
- Can we determine/decide whether:
  - An arbitrary theorem in number theory holds true?
  - An arbitrarily given Java program enters an infinite loop?
  - Two arbitrarily given Python programs exhibit equivalent functionality?
  - An arbitrary sentence logically follows some axioms in First-Order Logic?
- Is it possible for a machine to emulate human behavior?

General Questions

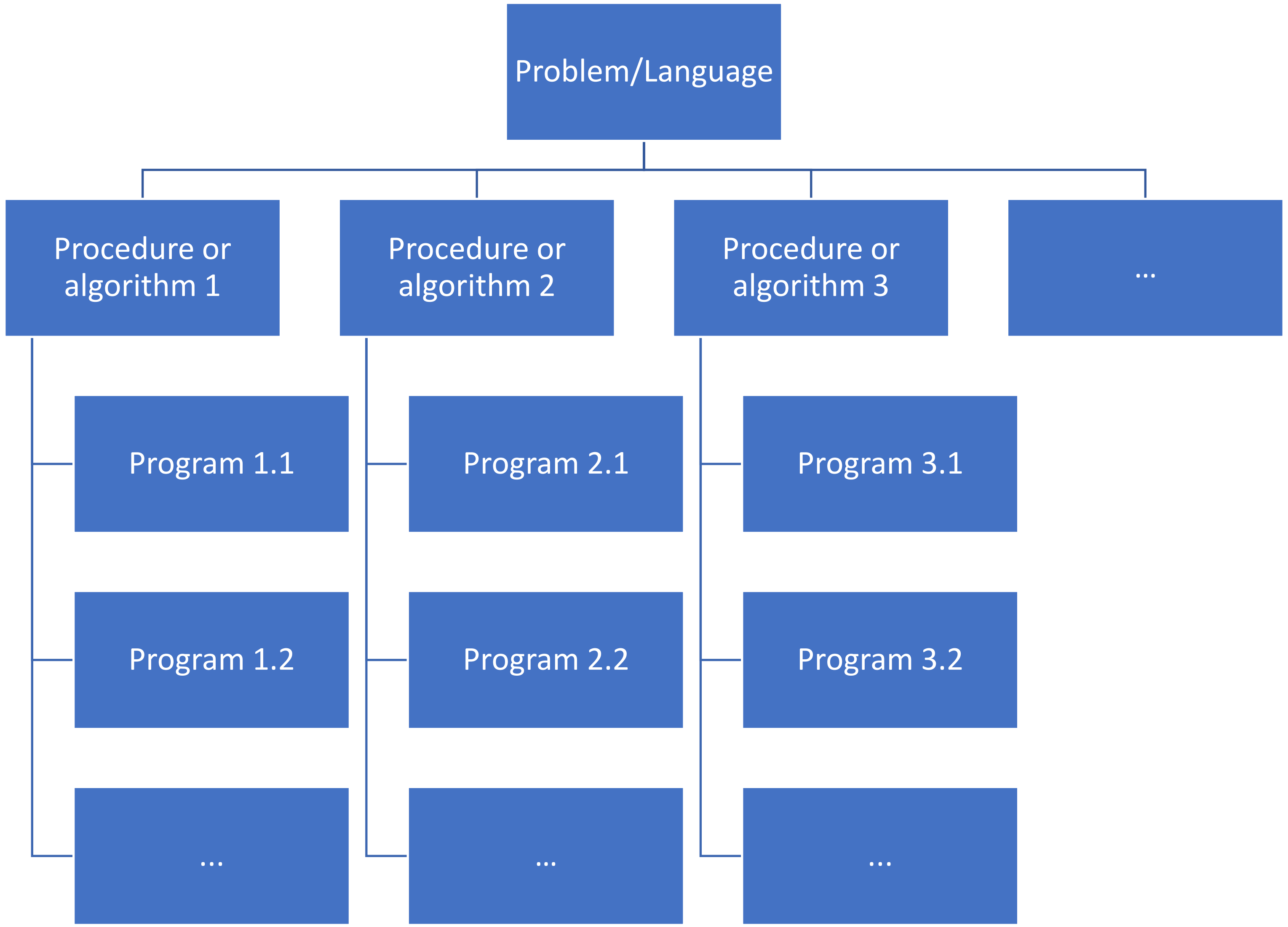
- What does it mean to say that a function is computable?
  - When is a function not computable?
  - Are there cases when we are sure that a function is NOT computable?

- How can one solve a “solvable” problem?
  - How can we characterize the intrinsic level of difficulty for solving problems that admit algorithmic solutions?
  - Is it possible to manage all “solvable” problems?

Theory of Computation

- Branch of Computer Science and Mathematics including:
  - **Computability theory**: is a/any problem solvable on a computer?
    - Decidable Problems
    - Semi-decidable Problems
    - Completely Undecidable Problems
  - **Complexity theory**: how efficiently a “solvable” problem can be solved?
    - Time complexity (~amount of needed time)
    - Space complexity (~amount of needed space)
  - **Logic in Computer Science**
    - Propositional logics
    - First-order logic
    - …
  - **Algorithm Design and Analysis**
  - …

Problems, procedures/algorithms and programs



IBM 7090 Programming in the 1950's

```
ENTRY     SXA    4,RETURN
          LDQ    X
          FMP    A
          FAD    B
          XCA
          FMP    X
          FAD    C
          STO    RESULT
RETURN    TRA    0
A         BSS    1
B         BSS    1
C         BSS    1
X         BSS    1
TEMP      BSS    1
STORE     BSS    1
          END
```

Programming in the 1970's (IBM 360)

```
//MYJOB      JOB (COMPRESS),
             'VOLKER BANDKE',CLASS=P,COND=(0,NE)
//BACKUP     EXEC PGM=IEBCOPY
//SYSPRINT   DD  SYSOUT=*
//SYSUT1     DD  DISP=SHR,DSN=MY.IMPORTNT.PDS
//SYSUT2     DD  DISP=(,CATLG),
             DSN=MY.IMPORTNT.PDS.BACKUP,
//           UNIT=3350,VOL=SER=DISK01,
//           DCB=MY.IMPORTNT.PDS,
             SPACE=(CYL,(10,10,20))
//COMPRESS   EXEC PGM=IEBCOPY
//SYSPRINT   DD  SYSOUT=*
//MYPDS      DD  DISP=OLD,DSN=*.BACKUP.SYSUT1
//SYSIN      DD  *
            COPY INDD=MYPDS,OUTDD=MYPDS
//DELETE2    EXEC PGM=IEFBR14
//BACKPDS    DD  DISP=(OLD,DELETE,DELETE),
             DSN=MY.IMPORTNT.PDS.BACKUP
```

Today’s Programming

Today's programming landscape is vastly different from the past:

- We now work with high-level languages like C++, Java, Python, and more
- Modern programmers may struggle to understand code written decades ago

*Why is it important to study the **Science of Computing**?*

- Learning the science of computing goes beyond mastering programming languages.
- It involves understanding the mathematical properties of problems and algorithms that remain constant regardless of technological advancements or programming trends.
- Fundamental principles of computing remain valid across all hardware platforms, defining the boundaries of computability, independent of processing power or memory.

Why TCS Remains Essential in the Age of AI

Theoretical Computer Science (TCS) is fundamental to computing, even as Artificial Intelligence (AI) continues to evolve. Its relevance lies in its deep exploration of computation itself:

- TCS delves into the core principles of computing, addressing questions about what is computable (computability), the complexity of solving problems (computational complexity), ensuring information security (cryptography), logical reasoning (logic), and even exploring quantum computing (beyond the scope of this course).
- Using tools like circuits, complexity classes, decision trees, and Turing machines, TCS defines the limits and possibilities of computation.

Meanwhile, **Artificial Intelligence** focuses on building systems that mimic human cognitive abilities, such as speech recognition, natural language understanding, decision-making, and image processing:

- AI relies on techniques like machine learning, expert systems, and rule-based models. However, TCS provides the theoretical foundation necessary for designing, analyzing, and improving these AI systems.
- In essence, TCS enriches our understanding of computation, guiding AI's evolution and ensuring its rigorous development.

Self-assessment

Nothing for this lecture!