CM0081 Automata and Formal Languages
Introduction

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2024-1

Pacto pedagógico

Como miembros de la Universidad EAFIT, nos comprometemos a actuar de manera íntegra siguiendo los más altos estándares éticos y morales.

- Respeto
- Tolerancia
- Honradez
- Compromiso

Pacto pedagógico

Página web del curso

`http://www1.eafit.edu.co/asr/courses/cm0081-automata/`

Pacto pedagógico

Página web del curso

`http://www1.eafit.edu.co/asr/courses/cm0081-automata/`

Pacto pedagógico

Página web del curso

`http://www1.eafit.edu.co/asr/courses/cm0081-automata/`

Conducto regular, fechas y porcentajes de las evaluaciones

La información está en la página web del curso.

Pacto pedagógico

Página web del curso

http://www1.eafit.edu.co/asr/courses/cm0081-automata/

Conducto regular, fechas y porcentajes de las evaluaciones

La información está en la página web del curso.

Responsabilidad compartida

- Profesor
- Estudiantes

Pacto pedagógico

Orientaciones para el curso

- Se recomienda seis horas de trabajo por semana (dos horas por cada hora de clase).
- Las clases son presenciales.
- La evaluación a la docencia es obligatoria.
- Se recomienda revisar periódicamente los canales de comunicación institucionales (EAFIT Interactiva, correo institucional, Microsoft Teams).
- Las prácticas no se pueden realizar de manera individual y se deben realizar máximo entre dos estudiantes.

Preliminaries

Conventions

- The number and page numbers assigned to chapters, examples, exercises, figures, quotes, sections and theorems on these slides correspond to the numbers assigned in the textbook [Hopcroft, Motwani and Ullman 2007].
- The natural numbers include the zero, that is, $\mathbb{N} = \{0, 1, 2, \dots\}$.
- The power set of a set A, that is, the set of its subsets, is denoted by $\mathcal{P}\,A$.

Computability (Decidability)

Question (informal)

What can a computer do at all?

Computability (Decidability)

Question (informal)

What can a computer do at all?

Definition (informal)

A **computable** (or **decidable**) **problem** is a problem than can be solved by an algorithm.

Computability (Decidability)

Question (informal)

What can a computer do at all?

Definition (informal)

A **computable** (or **decidable**) **problem** is a problem than can be solved by an algorithm.

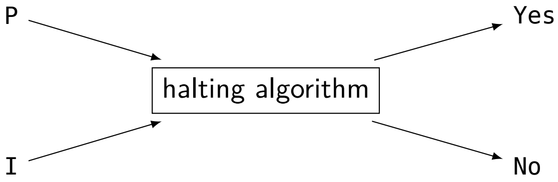
Question

Are there undecidable problems?

Computability (Decidability)

Example (The halting problem: An undecidable problem)

Given an program `P` and an input `I`, to decide if the program will halt or will run forever.



The halting algorithm does not exist.

Algorithmic Complexity (Tractability)

Question

What can a computer do efficiently?

Algorithmic Complexity (Tractability)

Question

What can a computer do efficiently?

Definition

A **tractable problem** is a problem than can be solved by a computer algorithm that run in polynomial time.

Algorithmic Complexity (Tractability)

Example (3-SAT: An intractable problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

Algorithmic Complexity (Tractability)

Example (3-SAT: An intractable problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

A (propositional logic) formula F is in **conjunctive normal form** iff

$$F \text{ has the form } F_1 \wedge \cdots \wedge F_n,$$

where each $F_1, \ldots, F_n$ is a disjunction of literals.

Algorithmic Complexity (Tractability)

Example (3-SAT: An intractable problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

A (propositional logic) formula F is in **conjunctive normal form** iff

$$F \text{ has the form } F_1 \wedge \cdots \wedge F_n,$$

where each $F_1, \ldots, F_n$ is a disjunction of literals.

3-SAT problem: To determine the satisfiability of a propositional formula in conjunctive normal form where each disjunction of literals is limited to at most three literals.

Algorithmic Complexity (Tractability)

Example (3-SAT: An intractable problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

A (propositional logic) formula F is in **conjunctive normal form** iff

$$F \text{ has the form } F_1 \wedge \cdots \wedge F_n,$$

where each $F_1, \dots, F_n$ is a disjunction of literals.

3-SAT problem: To determine the satisfiability of a propositional formula in conjunctive normal form where each disjunction of literals is limited to at most three literals.

The 3-SAT problem is an intractable problem. The problem was proposed in Karp's 21 NP-complete problems [Karp 1972].

Computability and Algorithmic Complexity

Classification of problems

- Problem (Computability)
  - Computable (decidable) (Algorithm Complexity)
    - Tractable
    - Intractable
  - Non-computable (undecidable) (Hypercomputation)

Course Outline

Language	Machine	Other models
Regular	DFA	▪ Regular expressions ▪ NFA ▪ ε-NFA
Context-free	Pushdown automata	
Recursive	Halting TMs	▪ λ-calculus ▪ Total recursive functions
Recursively enumerable	TMs	▪ λ-calculus ▪ Partial recursive functions

DFA: Deterministic finite automata
NFA: Non-deterministic finite automata
ε-NFA: Non-deterministic finite automata with ε-transitions
TM: Turing machine

Paradigms of Programming

Some paradigms

- Imperative/object-oriented: Describe computation in terms of state-transforming operations such as assignment. Programming is done with statements.
- Functional: Describe computation in terms of (mathematical) functions. Programming is done with expressions.
- Logic: Predicate calculus as a programming language. Programming is done with sentences.

Examples

Imperative/OO:	C, C++, Java, Python
Functional:	Erlang, Haskell, Standard ML
Logic:	CLP(R), Prolog

Pure Functional Programming

Description

*'A **side effect** introduces a dependency between the global state of the system and the behaviour of a function... Side effects are essentially invisible inputs to, or outputs from, functions.' [O'Sullivan, Goerzen and Stewart 2008, p. 27]*

Pure Functional Programming

Description

- A **pure function** *'take all their input as explicit arguments, and produce all their output as explicit results.' [Hutton 2016, § 10.1]*

Pure Functional Programming

Description

- A **pure function** *'take all their input as explicit arguments, and produce all their output as explicit results.' [Hutton 2016, § 10.1]*
- A function is a **pure function** if it satisfies both of the following statements (Wikipedia: Pure function (July 28, 2014)):

  (i) *'The function always evaluates the same result value given the same argument value(s). The function result value cannot depend on any…state that may change as program execution proceeds or between different executions of the program, nor can it depend on any external input from I/O devices.'*

Pure Functional Programming

Description

- A **pure function** *'take all their input as explicit arguments, and produce all their output as explicit results.' [Hutton 2016, § 10.1]*
- A function is a **pure function** if it satisfies both of the following statements (Wikipedia: Pure function (July 28, 2014)):

  (i) *'The function always evaluates the same result value given the same argument value(s). The function result value cannot depend on any…state that may change as program execution proceeds or between different executions of the program, nor can it depend on any external input from I/O devices.'*

  (ii) *'Evaluation of the result does not cause any semantically observable side effect or output, such as mutation of mutable objects or output to I/O devices.'*

Pure Functional Programming

Referential transparency

- Equals can be replaced by equals

Pure Functional Programming

Referential transparency

- Equals can be replaced by equals
- *'By definition, a function in Haskell defines a fixed relation between inputs and output: whenever a function f is applied to the argument value arg it will produce the same output no matter what the overall state of the computation is. Haskell, like any other pure functional language, is said to be "referentially transparent" or "side-effect free". This property does not hold for imperative languages.' [Grune, Bal, Jacobs and Langendoen 2003, pp. 544–545]*

Pure Functional Programming

Reasoning about (pure) functional programs

Equational reasoning + induction + co-induction + ...

Reading

Homework

To read from the textbook the following sections:

§ 1.1. Why Study Automata Theory?

§ 1.2. Introduction to Formal Proofs

§ 1.3. Additional Forms of Proofs

References

- Grune, D., Bal, H. E., Jacobs, C. J. H. and Langendoen, K. G. (2003). Modern Compiler Desing. Worldwide Series in Computer Science. John Wiley & Sons (cit. on pp. 26, 27).
- Hopcroft, J. E., Motwani, R. and Ullman, J. D. [1979] (2007). Introduction to Automata Theory, Languages, and Computation. 3rd ed. Pearson Education (cit. on p. 8).
- Hutton, G. [2007] (2016). Programming in Haskell. 2nd ed. Cambridge University Press (cit. on pp. 23–25).
- Karp, R. M. (1972). Reducibility Among Combinatorial Problems. In: Complexity of Computer Computations. Ed. by Miller, R. E. and Thatcher, J. W. Plenum Press, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9 (cit. on pp. 15–18).
- O'Sullivan, B., Goerzen, J. and Stewart, D. (2008). Real World Haskell. O'Really Media, Inc. (cit. on p. 22).