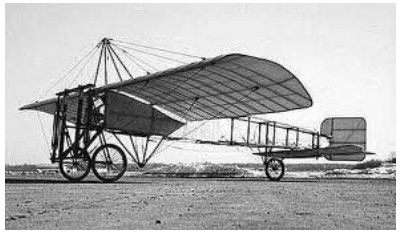
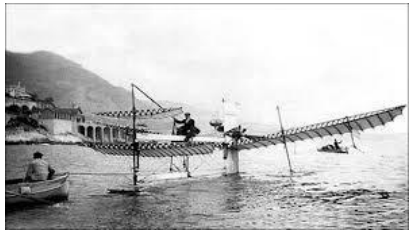
CentraleSupélec

Elements of Complex Systems Engineering

Antoine B. Rauzy
Department of Mechanical and Industrial Engineering (MTP)
Norwegian Science and Technology University (NTNU)
and
Chaire Blériot-Fabre
Centrale-Supélec, SAFRAN


CentraleSupélec

LECTURE 10.
LIMITS OF CALCULATION

Notions:

- Undecidability
- Computational Intractability
- Sensitivity to Initial Conditions

CentraleSupélec

LECTURE 10. PART 1. INTRODUCTION

CentraleSupélec

Objective of this lecture

Designing models is the only way to tackle the complexity of systems. However, there are **intrinsic** and **extrinsic limits** to modeling.

Obviously, we can only model what we know. This is the problem "**black swans**", i.e. of phenomena that were completely unexpected (extrinsic limits).

But there are also limits due to models themselves:

- Some problems are **undecidable**, i.e. that there is provably no algorithm to solve them.
- Some problems are **hard/intractable** in the sense of the **computational complexity theory**, i.e. there is provably no efficient algorithm to solve them.
- Finally, some problems are extremely **sensitive to initial conditions**, i.e. that it is extremely hard to predict the evolution of the system under study.

The objective of this course is to present these three types of difficulties.

CentraleSupélec

LECTURE 10. PART 2. UNDECIDABILITY

CentraleSupélec

Case Study: Program Equivalence

One of the big issues of software development is to warranty that a new version of a software does not degrade the functionalities provided by the previous version. The question is therefore: is there an automatic mean to check that? More formally:

Program equivalence problem:

Is it possible to design a program E such that for any two programs P and Q,

- E(P, Q) = true if for any input I, P(I) = Q(I), and
- E(P, Q) = false, otherwise.

Important remark: I, P(I), Q(I) but also P, Q and E are eventually sequences of symbols (strings).

CentraleSupélec

What to Do?

To solve the program equivalence problem, we bump immediately into a first problem: there may be infinitely many possible input string I for P and Q. Therefore, it is possible for our program E to test them one by one.

But even if we let aside this question, we shall see that problem much deeper than that: in fact, no such program E can exist.

The program equivalence problem is **undecidable**.

The proof of this result use some of the most important concepts of **mathematical logic**.

CentraleSupélec

Halting Problem

To show the undecidability of the program equivalence problem, we shall reduce it to the halting problem. The halting problem consists in determining whether a program terminates on any input. More formally:

Halting problem:

Is it possible to design a program H such that for any program P:

- H(P) = true if P terminates on any input I,
- H(P) = false, otherwise.

As we shall see, this problem also is undecidable and its undecidability will prove the undecidability of the program equivalence problem.

CentraleSupélec

Reduction

Claim: Solving the program equivalence problem is at least as hard as solving the halting problem.

Proof: Let P be a program.

- Let P_0 be the program that returns immediately 0 on any input.
- Now consider the following program P'.

    P'(I): P(I); return 0

Clearly, P' is equivalent to P_0 if (and only if) P terminates on any input. Therefore, if we have a program E to test the equivalence of two programs (in that case P' and P_0), we can easily design a program H to test the termination of P. QED.

This argument is called a **reduction**. The program equivalence problem reduces to the halting problem. Reductions play a central role in computational complexity theory.

Undecidability of the Halting Problem (1)

Assume for a contradiction that there exists a program H such that for any program P and input I,

- H(P, I) = true if P terminates on I; and
- H(P, I) = false otherwise.

H is of course assumed to terminates on pair (P, I)

Note that as P is a sequence of symbols, we can always apply P… to itself. It is thus easy to design a program D such that for any program P,

- D(P) = true if P applied to itself does not terminate, and
- D(P) loops forever otherwise.

Here is such a program D.

D(P) = if H(P,P) then loop-forever else true.

Undecidability of the Halting Problem (2)

Now we can apply D to itself:

- If D(D) terminates (and thus returns true), then it means that H(D,D) = false, that is D applied to itself does not terminate. A contradiction.
- If D(D) does not terminate, then it means that H(D,D) returns true, i.e. that D applied to itself terminates. A contradiction.

It follows that the program H cannot exist!

Theorem: The **halting problem** is **undecidable**.

Corollary: The **program equivalence problem** is also **undecidable**.

CentraleSupélec

Diagonalization (1)

The proof of the undecidability of the halting problem relies on a **diagonalization technique**.

In its simplest form, the diagonalization principle is nothing but the so-called the **liar paradox**, also called Epimenides the Cretan paradox:

"Cretans are liars*"

Or, even more simply:

"I lie"

Several other versions of this paradox have been proposed, like **Russel**'s paradox:

"The barber shaves all the men of the city that do not shave themselves"

(*) Cité dans l'épître à Tite, l'un des livres du Nouveau Testament par Paul de Tarse

Diagonalization (2)

In mathematics, this technique is used for instance to show that the cardinal of the power set of a set is always greater than the cardinal of the set. (**Cantor's theorem**):

Let f be a function from E to its power set 2^E. It is possible to build the subset D of E of elements that do not belong to their image by f:

$$D = \{x \in E, x \notin f(x) \}$$

D cannot have any antecedent by f. Assume for a contradiction that there is an element y of E such that D = f(y). Then,

- If $y \in D$, then by construction D, $y \notin f(y) = D$. A contradiction ;
- if $y \notin f(y) = D$, then still by construction, $y \in D$. A contradiction.

It follows that f cannot be a one-to-one correspondence. Therefore, E and 2^E do not have the same cardinal. QED.

An Old and Fascinating Story

19th century: discovery of various mathematical paradoxes.

Beginning of 20th century: David Hilbert's program:

Formalization of all mathematics.

David Hilbert
(1862-1943)

1931: Kurt Goedel's incompleteness theorem:

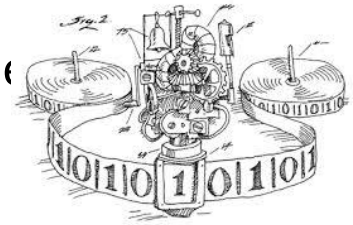
There is no algorithm to decide the truth (or provability) of statements in any consistent extension of Peano arithmetic (set theory).

Kurt Goedel
(1906-1978)

1936: Alan Turing's seminal articles about calculation

There exist universal calculators.

There is no algorithm to decide whether a calculator halts on a given input.

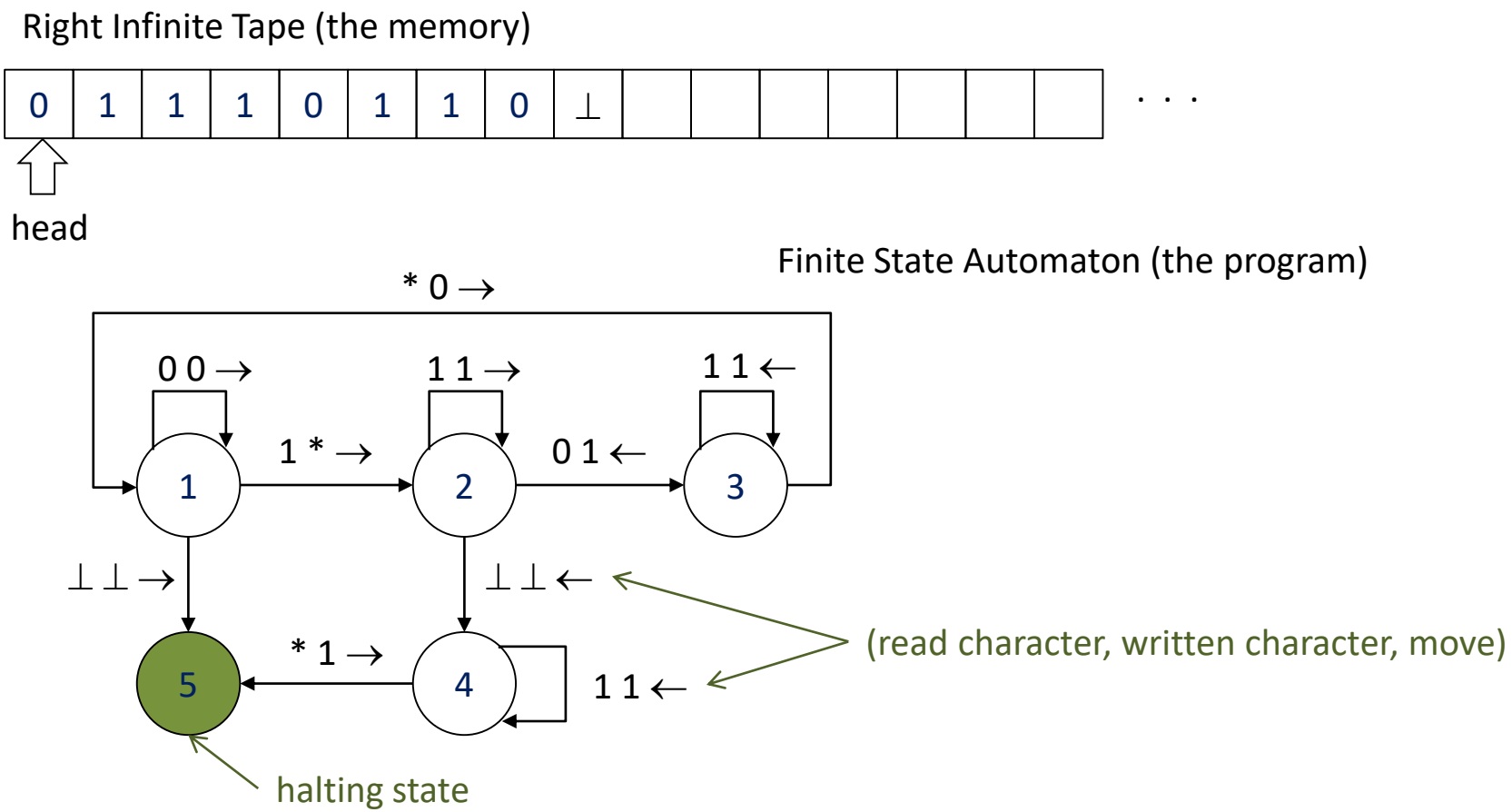


Turing machine

Alan Turing
(1912-1954)

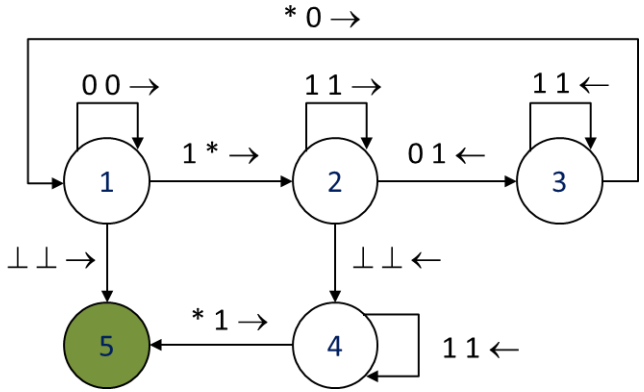
Turing Machines

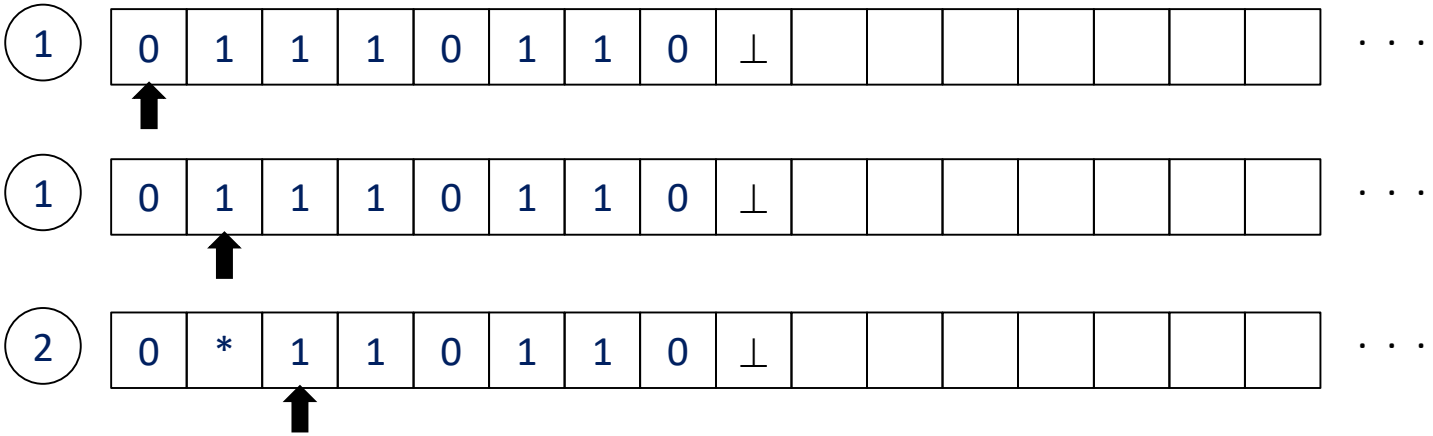
Gödel's proof is quite technical. So, Alan Turing decided to seek for a more understandable proof. He invented a very primitive yet very powerful **calculator**: the **Turing machine**.



CentraleSupélec

Executions





and so on…

CentraleSupélec

Universal Turing Machine

- An universal Turing machine U takes the description of a program P (a Turing machine) and a data D as input. Its execution must be such that U(P, D) = P(D).

- There exists **Universal Turing Machines**!

- This result reinforces the **Church Thesis**:

  "We shall use the expression 'computable function' to mean a function calculable by a machine, and let 'effectively calculable' refer to the intuitive idea without particular identification with any one of these definitions. Then, every effectively calculable function is a computable function." [Alonzo Church]

  i.e. **any computable function can be computed with a Turing machine**.

Alonzo Church
(1903-1995)

CentraleSupélec

The Halting Problem revisited

Halting Problem: given a Turing machine M and a data D, does M halt on D?

Theorem: the Halting Problem is **undecidable**.

Proof: by a **diagonal argument**:

Assume there exists a machine HALT:

$$\mathrm{HALT}(\mathrm{M},\mathrm{D}) = \begin{cases} 1 \text{ if M halts on D} \\ 0 \text{ otherwise} \end{cases}$$

It is easy to create a machine DIAGONAL(M) such that:

$$\mathrm{DIAGONAL}(\mathrm{M}) = \begin{cases} 1 \text{ if } \mathrm{HALT}(\mathrm{M},\mathrm{M}) = 0 \\ \text{loops forever otherwise} \end{cases}$$

Now,

- If DIAGONAL(DIAGONAL) returns 1 (halts), then HALT(DIAGONAL, DIAGONAL)=0 which means that DIAGONAL(DIAGONAL) does not halt. A contradiction
- If DIAGONAL(DIAGONAL) loops forever, then HALT(DIAGONAL, DIAGONAL)=1, which means that DIAGONAL(DIAGONAL) halts. A contradiction

Some Undecidable Problems

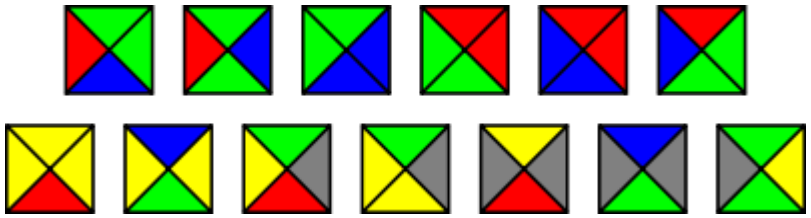
- **Diophantine Equation (10th Hilbert problem):** Does a Diophantine equation admit an integral solution?
  - $\sum a_i x_i^{k_i} = 0$ with the a_i's and the k_i's integers

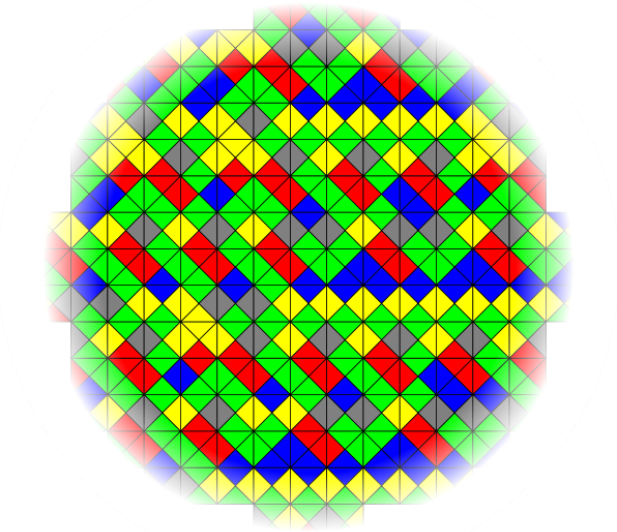
  [Matiyasevich 1970]

- **Post Correspondence Problem:** The input of the problem consists of two finite lists of words $v_1,\ldots,v_n$ and $w_1,\ldots,w_m$ of words over some alphabet Σ having at least two symbols. Is there a sequence of indices $i_1,\ldots i_k$, such that
  - $v_{i_1} \ldots v_{i_k} = w_{i_1} \ldots w_{i_k}$

- **Wang Tiles Problem:**

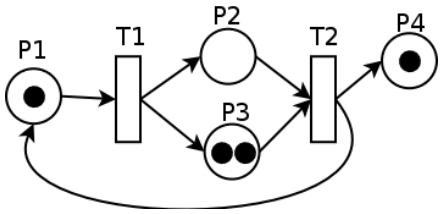
  Can a given finite set of Wang tiles tile the plane?





Consequences for Models Engineering

The **marking** of a **Petri net** (a kind of discrete event system) is a function that associates with each place the number of tokens in that place.



A Petri net

The **reachability problem** consists in determining whether a given marking M is reachable from a given initial marking M_0.

Regular Petri Nets	Petri nets with inhibitor arcs
Decidable (Mayr, 1981)	**Undecidable** (see Esparza & Nielsen 1995)

And also:

- Reachability in discrete event systems: **undecidable**
- Equivalence of discrete event systems: **undecidable**
- …

CentraleSupélec

Practical Consequences?

Results by Gödel, Turing et alii have extremely important **scientific and philosophical consequences**. Among other things, they show that the complexity of the world can be described in a book, would it be a “holy” book… But is there any **practical consequence**?

The answer is both yes and no:

- Yes, because this forces scientists and engineers to be modest: not everything is calculable, there are limits to the human genius.
- No, because this does not prevent us to go forward.

In practice, two other **limits** of the calculability have much more important consequences on daily science and engineering:

- The **complexity of calculations**;
- The **sensitivity to initial conditions**.

CentraleSupélec

LECTURE 10. PART 3. COMPUTATIONAL COMPLEXITY

CentraleSupélec

Case Study: SAT

We shall consider **Boolean formulas**, i.e. formulas written with variables and the three connectives (operators) "$\vee$" (or), "$\wedge$" (and) and "$\neg$" (not). E.g.

$$(A \vee B) \wedge (\neg A \vee C)$$

A **variable assignment** is a function from variables to {0, 1} (false and true). Variables assignments are lifted up into functions from Boolean formulas into {0, 1} using the truth tables of "$\vee$", "$\wedge$" and "$\neg$". Let f and g be two Boolean formulas and σ be a variable assignment, then:

$$\sigma(f \vee g) = \max(\sigma(f), \sigma(g)),\ \ \sigma(f \wedge g) = \min(\sigma(f), \sigma(g)),\ \ \sigma(\neg f) = 1 - \sigma(f)$$

A variable assignment σ **satisfies** a Boolean formula f, if $\sigma(f) = 1$, otherwise it **falsifies** f.

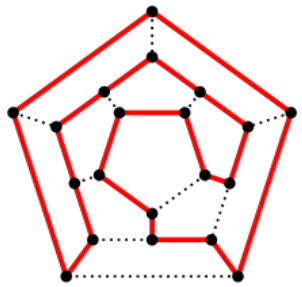
SAT problem: Let f be a Boolean formula, is there an assignment that satisfies f?

Decision Problems

SAT is a **decision problem**, i.e. a problem with a yes/no answer.

Examples decision problems:

- **Constraint satisfaction problems**
- **Halmitonian-path**: Let G = (V,E) be a graph. Is there an path in that graph visiting all vertices and never visiting twice the same vertex?
  The Hamiltonian path is indeed strongly related to the traveling salesman problem.



It is easy to verify that many other decision problems, including the two above, can be encoded as SAT instances.

CentraleSupélec

What to Do?

SAT problem: Let f be a Boolean formula, is there an assignment that satisfies f?

Candidate solution: try one by one all possible variable assignments until a solution is found or all possible assignments have been tried.

Problem: there are 2^N candidate variable assignments if the formula involves N variables.

A small exercise…

Take a sheet of paper. Fold it. Fold it again. And again.

Each time you fold it, it gets twice thicker.

?

Question: how many times do you need to fold the sheet to reach the moon?

Solution

A packet of 500 sheets is about 5 cm thick.

Let us start:

- After 10 folding, we get a $2^{10} \approx 1.000$ sheets (10 cm) stack
- After 20 folding, we get a 2^{10} x 10 cm ≈ 1.000 x 10 cm = 100 m stack
- After 30 folding, we get a 2^{10} x 100 m ≈ 1.000 x 100 m =100 km stack
- After 40 folding, we get a 2^{10} x 100 m ≈ 1.000 x 100 km = 100.000 km stack

World record at MIT 13 folding!

- The **distance from the earth to the moon varies between** 356.375 km and 406.720 km. On average it is 384.400 km.
- In **42 folding**, we get a stack of 2^2 x 100.000 km = 400.000 km which is way enough to go to the moon…
- … and if we fold it once more, we can come back

CentraleSupélec

Computation Times

We could for instance test first variable assignments with 1 variable set to 1, then with 2 variables set to 1, and so on.

Computation times to evaluate all configurations with k components out of n on a computer that evaluates 10^6 configurations per second.

n/k	2	3	4	5	6	7	10
50	0.001”	0.02”	0.2”	2”	20”	2h	2h50’
100	0.005”	0.2”	4”	1’20”	21’	4h	6 months
200	0.02”	1”	1’	40’	21h	26 days	7 centuries
1000	0.5”	2’50”	10h30’	6 days	40 years	60 centuries	$8\ 10^7$ centuries

CentraleSupélec

What to Do?

SAT problem: Let f be a Boolean formula, is there an assignment that satisfies f?

Candidate solution: try one by one all possible variable assignments until a solution is found or all possible assignments have been tried.

Problem: there are 2^N candidate variable assignments if the formula involves N variables.

A brute force algorithm is not suitable.

But can we do better?

Is there an algorithm that performs significantly better?

To answer this question, we shall use… Turing machines.

CentraleSupélec

Complexity of Executions

- Complexity of the execution of the Turing machine T on a data D
  - **Time complexity**: number of steps of the execution
  - **Space complexity**: number of cells used (index of the right most visited cell)
- The **Big O** notation:
  - A Turing machine T has a time/space complexity in O(f), where f(.) is any function from integer to integer if there exists a constant c such that for any data D of size n, the time/space complexity of the execution of T on D is less than c.f(n)
  - Linear complexity: O(n)
  - Polynomial complexity: $O(n^k)$ for an integer k
  - Exponential complexity: $O(2^n)$
- We are speaking of **worst case complexity**.
- Space complexity is much less constraining than time complexity, e.g. a Turing machine can be of linear complexity in space, but exponential complexity in time

From Decidability down to Complexity

There exist universal Turing machines U such that for any program P If P(.) is in O(f) then U(P, .) is in O(f log f).
Turing machines are thus not only a model of calculator, they are also very helpful to characterize the **complexity of a calculation**.

The **complexity of an algorithm** is in **O(f(n))** if the maximum number of steps of that algorithm on an input of size n is O(f(n)).

The **complexity of a problem** is in **O(f(n))** if the complexity of the fastest algorithm to solve that problem is in O(f(n)).

Complexity of problems seen by mathematicians (and computer scientists):

Easy	**Hard**
Complexity in g(n) where g is a polynomial	Complexity in g(n) where g is not bounded by any polynomial

CentraleSupélec

First Complexity Classes

We can group decision problems into classes of problems having the same complexity. E.g.

- The class **PTIME** (resp. **PSPACE**) if the class of all decision problems that can be solved in polynomial time (resp. space), i.e. in $O(n^k)$. Most often, **PTIME** is abbreviated in **P**.
- The class **EXPTIME** (resp. **EXPSPACE**) is the class of all decision problems that can be solved in exponential time (resp. space), i.e. is in $O(2^{p(n)})$, where p(n) is a polynomial in n.
- The class **L** is the class of all problems that can be solved in O(log(n)) space. As Universal Turing Machines are only equivalent only up to a log factor, it is better not to define **L** in terms of time complexity.

A decision problem whose time/space complexity is at least (resp. at most) C is said C-**hard** (resp. C-**easy**).

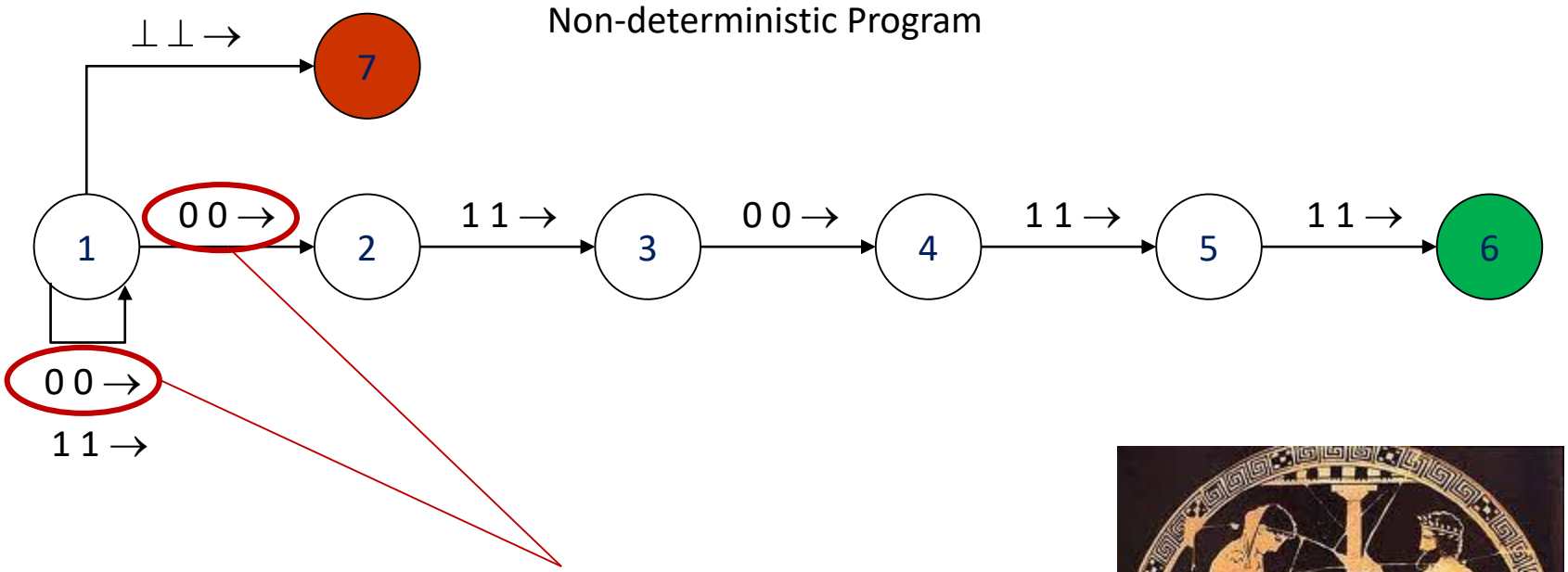
CentraleSupélec

Reductions

- Recall that a reduction is a transformation of one problem into another problem. It captures the informal notion of a problem being at least as difficult as another problem.
- If any instance of a problem P can be encoded as an instance of problem Q, we say that P **reduces** to Q.
- There are many different types of reductions. The usually considered reductions are such as **polynomial-time reductions** or **log-space reductions**.
- A decision problem P is **complete** for a complexity class C if any instance of any C-easy problem Q can be encoded as an instance of P and the complexity of the encoding E is itself C-easy, i.e. E(Q) is in C. In general, more strict conditions are set on the encoding, such as polynomial-time complexity or log-space complexity.
- **C-complete = C-easy ∩ C-hard**

Non-Deterministic Turing Machines

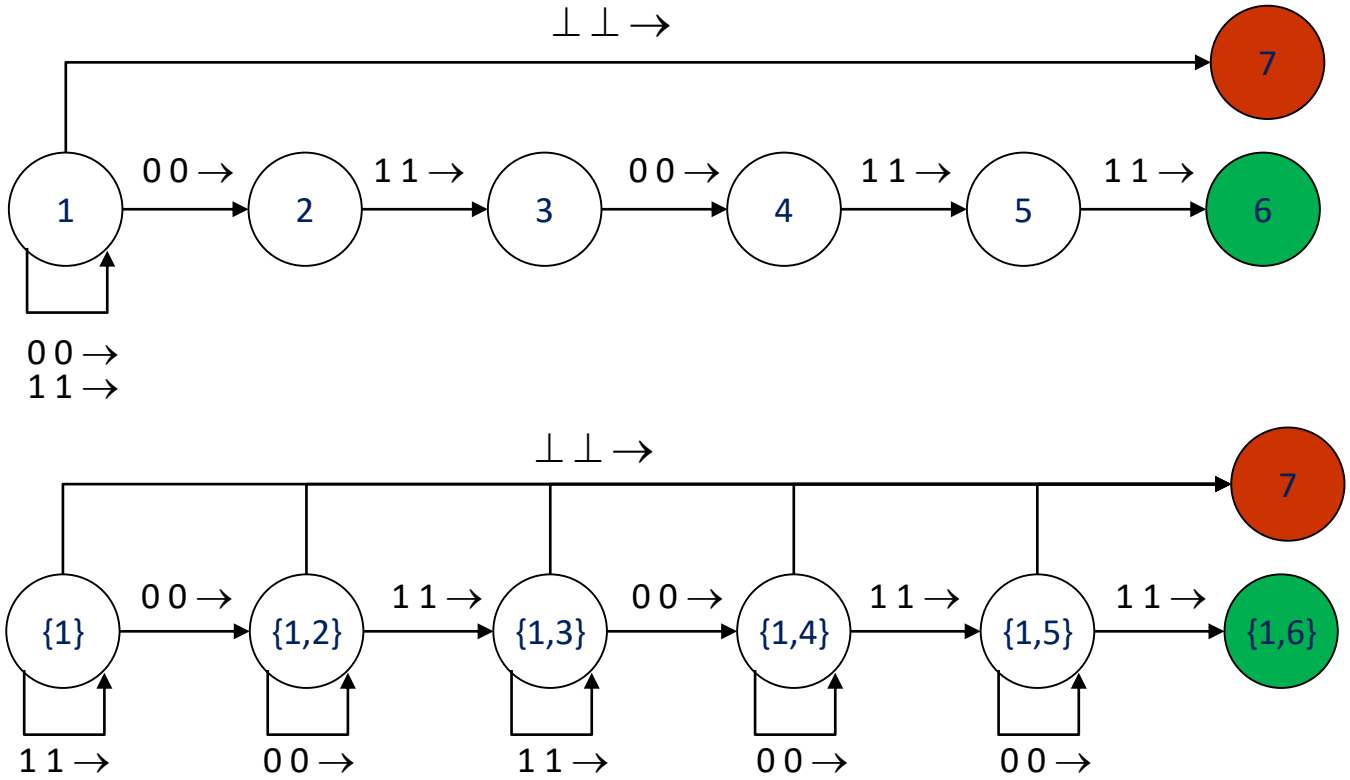
Problem: does the input contains the sequence 01011?



The choice of which transition to take is made by an **oracle**!

Non-Deterministic Turing Machines

It is possible to transform a non-deterministic finite state automaton into a deterministic one.



… but this transformation is exponential in the worst case (states of the deterministic automaton are sets of states of the non-deterministic one).

CentraleSupélec

Non-Deterministic Complexity Classes

- It is possible to transform any non-deterministic Finite Automaton into a deterministic one. But what is the cost for the execution?

- We can define complexity classes for non-deterministic Turing machine:
  - **NP** (NPTime): class of problems that can be solved in **polynomial time** with a **non-deterministic Turing machine**
  - **NPSPACE**: class of problems that can be solved in polynomial space with a non-deterministic Turing machine
  - …

- Stephen Cook showed in 1971 that

  SAT is NP-complete

  i.e. the fundamental result of complexity theory

Stephen Cook
(1939-…)

CentraleSupélec

Cook's Demonstration

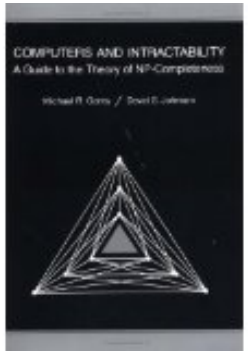
- Cook's demonstration is actually quite close to the reduction we used for the HAMILTONIAN-PATH problem.

- Assuming that there is a Non-Deterministic Turing machine that solves any instance I of a problem P in at most n steps where $n = c.|I|^k$ steps for some constants c and k.
- Then we can reduce P to SAT as follows (sketch).

- We create the following variables:
  - Tijk : true if the cell of index i of the tape contains the symbol j at step k.
  - Hik : true if the head is on cell i at step k.
  - Qqk : true if the program is state q at step k.
- Then we encode the behavior of the machine as clauses, e.g.
  - Exactly one of the Ti1k, Ti2k,… is true (on symbol per cell at each step)
  - Exactly one of the H1k H2k… is true (the head is one cell at each step)
  - The successor relation…

CentraleSupélec

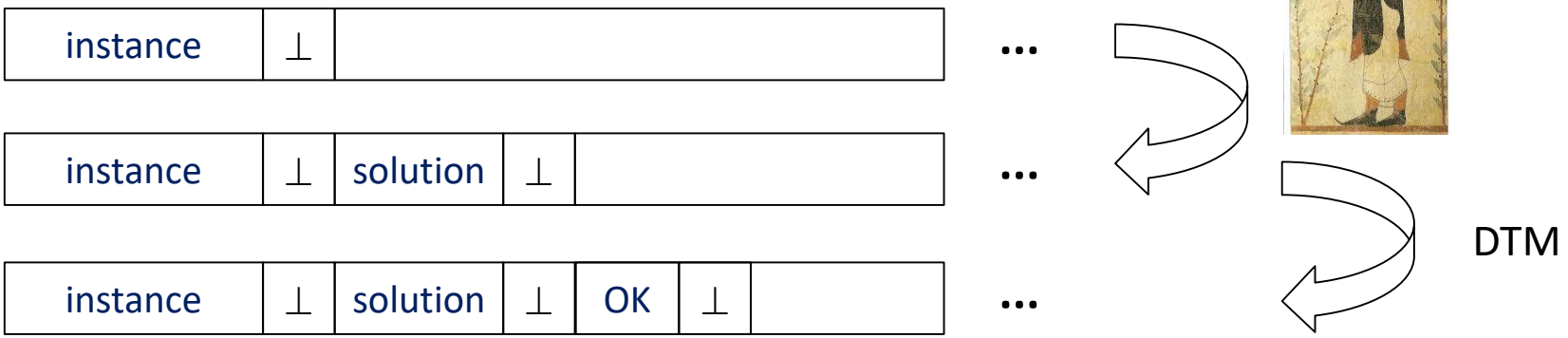
Polynomial Certificate



- Many problems have been shown NP-complete since Cook's result. See [Garey & Johnson 79]
- The class NP may seem quite unnatural: there is no such a thing as an oracle in our computers, is there?
- … but this class is better characterized by the so-called

Polynomial Certificate:

A problem is NP-easy if given a **candidate solution** of an instance of that problem, I can **check it in deterministic polynomial time** whether is it actually a solution.



CentraleSupélec

A $1 Million Question

$$P \stackrel{?}{=} NP$$

http://www.claymath.org/millennium/P_vs_NP/

CentraleSupélec

Practical Consequences?

Many practical engineering problems are **provably difficult**, i.e. for each not only no efficient algorithm is known, but we can prove that no such an algorithm can exist. In other words, the complexity of this problem is independent of any particular technology to solve them.

This applies not only for exact solutions, but also for approximations.

The **complexity of a system** depends not only on the organization of this system, but also on our ability to **answer questions** about its behavior. Many such questions are either undecidable or difficult (in the sense of computational complexity theory). An apparently simple system can be thus in reality a complex one.

CentraleSupélec

LECTURE 10. PART 4.
SENSITIVITY TO INITIAL CONDITIONS

Laplace's Demon

Encouraged by successes of celestial mechanics, **Laplace**, writes in 1814, in the introduction of his "Philosophical Essay on Probabilities":

"We must consider the present state of the universe as the consequence of its previous state and as the cause of its next state. An intelligence that, at a given point in time, would know all the forces that animate the nature and the respective situation of each thing that compose it, and that moreover would be vast enough to analyze all these data and to embrace in the same formula the movements of the biggest celestial bodies as well as those of the lightest atoms, then nothing would uncertain for this intelligence. The future and the past would be present to its eyes."

"God? Majesty, I don't need this hypothesis"
Answer of Laplace to Napoléon the first who asked him why his treaty on cosmology did not mention God.

CentraleSupélec

Poincaré's Answer

About one century after Laplace, **Poincaré** wrote in the introduction of his *Probability Calculus :*

"A very small cause, that we cannot see, may determine an effect that we cannot not to see. In this case, we say that this effect is at random. True that if we knew exactly all laws of the Nature and the exact state of the Universe, we could predict its evolution. But, even assuming that laws of Nature would have no more secret for us, we could only know approximately the state of the Universe. If this principle makes it possible for us to predict the evolution of a phenomenon with the same approximation as we know its initial condition, we use to say that this evolution obeys laws. But not all phenomena are such. **It may be the case that small differences in initial conditions produce very big differences in the evolution**. A small approximation on the initial conditions would thus induce an enormous mistake in prediction. **The prediction is then impossible and we face a random phenomenon**."

Lorenz Oscillator (1)

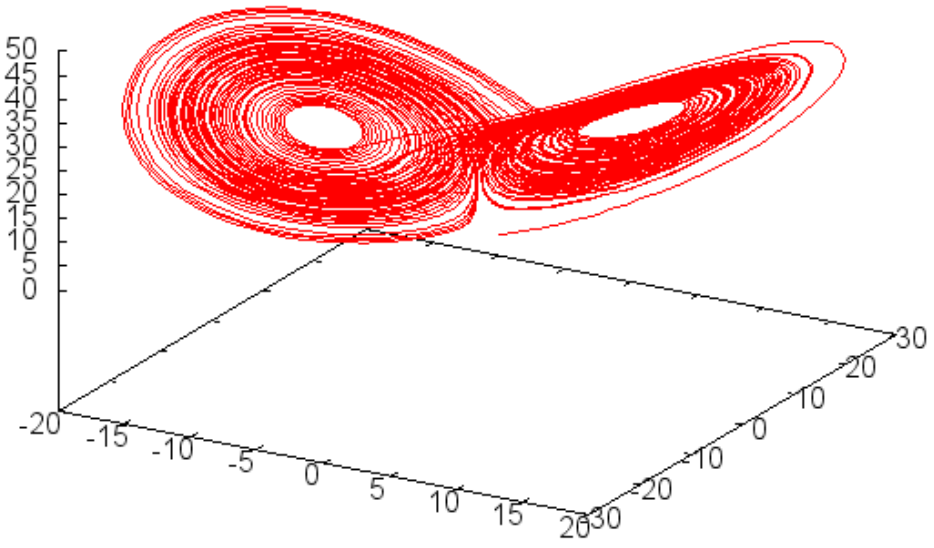
In 1963, the meteorologist **Edward Lorenz** gave evidence the chaotic nature of weather forecast. Lorenz equations are a simplified model of weather derived from fluid mechanics. It shows a **chaotic behavior** in certain initial condition and an extreme sensitivity to **initial conditions**.

$$\frac{dx(t)}{dt} = \sigma\big(y(t) - x(t)\big)$$

$$\frac{dy(t)}{dt} = \rho x(t) - y(t) - x(t)z(t)$$

$$\frac{dz(t)}{dt} = x(t)y(y) - \beta z(t)$$

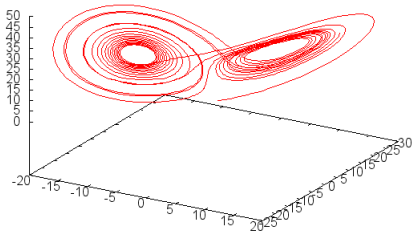
$\sigma = 10$, $\beta = 8/3$, $\rho = 28$



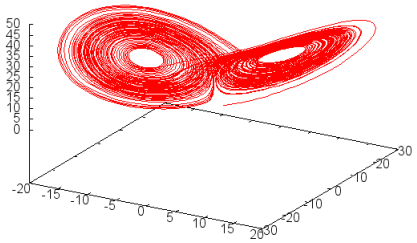
"Predictability: Does the Flap of a Butterfly's Wings in Brazil Set off a Tornado in Texas?"

Lorenz Oscillator (2)

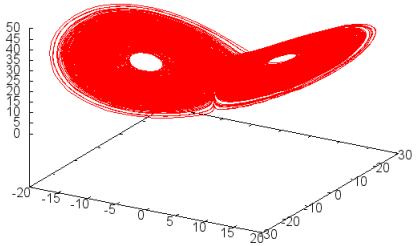
$x_0 = 1.0, y_0 = 2.0, z_0 = 3.0$



t=20



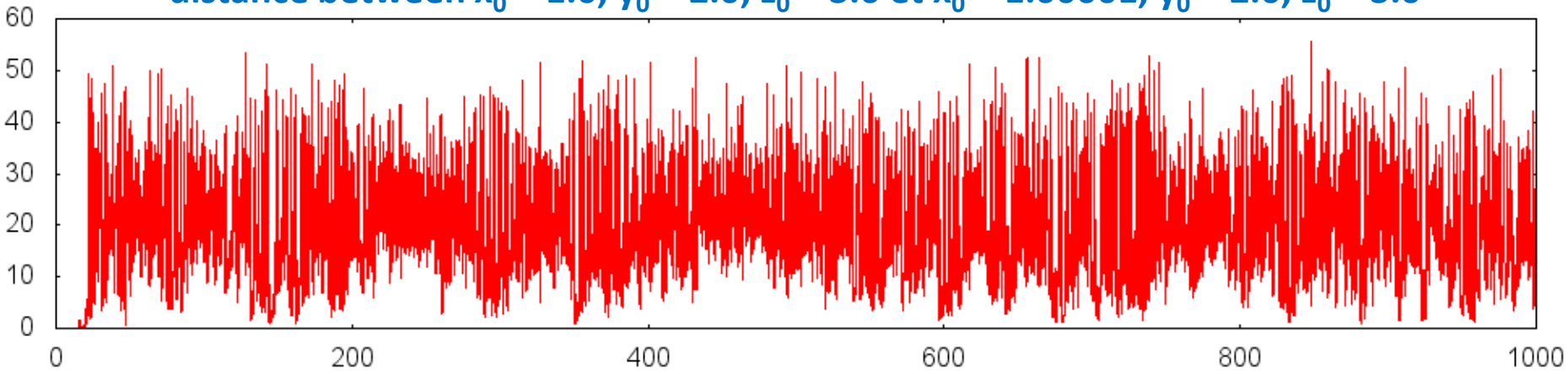
t=100



t=1000

The curve never passes twice by the same point. The space is progressively filled out, but only in a limited region (looking like butterfly wing)

distance between $x_0 = 1.0, y_0 = 2.0, z_0 = 3.0$ et $x_0 = 1.00001, y_0 = 2.0, z_0 = 3.0$



CentraleSupélec

LECTURE 10. PART 5.
WRAP-UP & ASSIGNMENT

CentraleSupélec

Wrap-Up

Some practical engineering problems are **undecidable**, i.e. there is provably no way to solve them.

Two other **limits** of the calculability have much more important consequences on daily science and engineering:

- The **complexity of calculations**;
- The **sensitivity to initial conditions**.

This does not prevent engineers and scientists to go forward, but they should be aware of these limitations.

CentraleSupélec

Assignment

See separated presentation.

Recommend Readings

Reference books on Gödel's theorem and undecidability:

- Gregory-J Chaitin. *The Limits of Mathematics: A Course on Information Theory and the Limits of Formal Reasoning.* Springer London Ltd. 2002. ISBN-13: 978-1852336684.
- Douglas Hofstadter. G*ödel Escher Bach: an Eternal Golden Braid*. Basic Books 1999. ISBN-13: 978-0465026562
- Apóstolos K. Doxiàdis and Christos Papadimitriou. *Logicomix: An Epic Search for Truth*. Bloomsbury Publishing PLC. 2009. ISBN-13: 978-1596914520

Reference books on computational complexity:

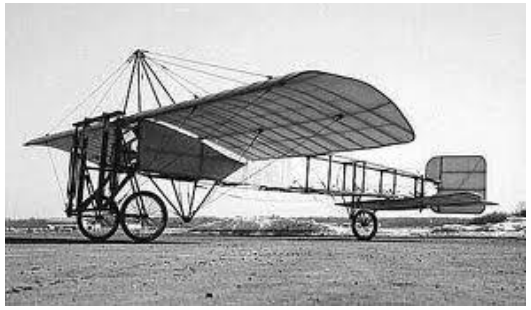
- Michael R. Garey et David S. Johnson. *Computers and Intractability: A Guide to the Theory of Np-Completeness*. W.H.Freeman & Co Ltd, 1989. ISBN-13: 978-0716710455.
- Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994. ISBN 0-201-53082-1.

Reference books on chaos theory:

- James Gleick. *Chaos*. Vintage 2007. ISBN-13: 978-0749386061

CentraleSupélec

Louis Charles Joseph Blériot (1872 -1936) is an airplane designer and one of the pioneer pilot of French aviation. He has been the first to cross the channel on July the 25th onboard of the Blériot XI.
He graduated from Ecole Centrale de Paris



Henri Marie Léonce Fabre (1882 -1984) is a French engineer and pilot. He invented the seaplane in 1910.
He graduated from Supélec.