



# Calculabilité, logique et complexité

## Chapitre 1 Introduction

# Chapitre 1 : Introduction

1. Qu'est-ce que la calculabilité ?
2. Information et informatique
3. La notion de problème
4. La notion de programme
5. Calculabilité et complexité
6. Résultats principaux
  - Equivalence des langages de programmation
  - Un problème non calculable
  - Un problème intrinsèquement complexe
7. Objectifs et intérêt

# Acquis d'apprentissage

A l'issue de ce chapitre, les étudiants seront capables de

- Comprendre et expliquer de manière intuitive ce qu'est la calculabilité et la complexité
- Se rendre compte de l'existence de limites de l'informatique pour résoudre des problèmes
- Distinguer la différences entre des limites théoriques (impossibilité) et pratiques (trop de ressources nécessaires)
- Donner un exemple de problème qui ne peut être résolu par un programme
- Donner un exemple de problème trop complexe que pour être résolu par un ordinateur, du moins pour de grandes données
- Représenter de manière schématique l'univers des problèmes, calculable et non calculable ainsi que calculable en pratique et non calculable en pratique

# 1. Qu'est-ce que la calculabilité ?

La *calculabilité* étudie les limites

- de l'informatique
- des langages de programmation
- des ordinateurs
- des systèmes formels

Deux types de limites

- limites théoriques
- limites pratiques

Quels sont les *problèmes* qui (ne) peuvent (pas) être résolus par un *programme* ?

Nécessité de définir

- la notion de problème
- la notion de programme

# 1. Qu'est-ce que la calculabilité ?

## Limites

Des limites théoriques existent dans différents domaines scientifiques

### *Mathématiques*

- Il est impossible de trouver deux nombres entiers  $m$  et  $n$  tel que  $m^2 = 2n^2$
- Il est impossible de trouver une fraction  $m / n$  exprimant le rapport entre le rayon d'un cercle et sa circonférence
- Il est impossible de diviser un angle en trois parties égales en utilisant uniquement une règle et un compas

### *Mécanique*

- Il est impossible de construire une machine réalisant un *mouvement perpétuel*

### *Physique*

- Il est impossible d'observer en même temps la *position* et la *vitesse* d'un électron

# 1. Qu'est-ce que la calculabilité ?

## *Informatique*

- Il est impossible d'écrire un programme ayant pour objectif de déterminer si deux programmes sont équivalents

Les limites de l'informatique ont des conséquences très directes ...

## 2. Information et informatique

### Information

Deux sens distincts

#### Langage courant :

information = affirmation de quelque chose (ayant un sens) qui est vrai ou faux

*il pleuvra demain*

*le stock contient 123 pneus*

#### Jargon informatique :

information = objet formel, symbole, sans signification propre

*243 "OUI"*

## 2. Information et informatique

### Traitement de l'information

#### Langage courant :

A partir d'informations, produire de nouvelles informations en appliquant un *raisonnement*

#### *Objectif :*

- obtenir une affirmation vraie, utile



## 2. Information et informatique

### Traitement de l'information

#### Jargon informatique :

Manipulation automatique d'un objet formel afin de produire un nouvel objet

#### Objectif :

- obtenir un nouvel objet auquel on peut donner un sens  
$$234 + 159 = 393$$
- un programme représente un raisonnement (afin que les résultats aient un sens)

Nécessité de *spécifier* un programme (i.e. donner une description de la fonction qu'il calcule) afin de pouvoir donner un sens aux résultats produits

*préconditions* :                      n, m entiers

*postconditions* :                    la somme de n et m

# 3. La notion de problème


## Exemple

- Additionner deux entiers
- Etant donné une liste d'étudiants et de leur points d'examens, trouver les étudiants dont la moyenne des examens est supérieure ou égale à 12/20
- Déterminer si un polynôme à coefficients entiers a des racines entières

## Caractéristique d'un problème

- Un problème est générique: il s'applique à un ensemble de données
- Pour chaque donnée particulière, il existe une réponse
- Problème  $\neq$  programme

## Représentation d'un problème

- Représentation (ici) d'un problème par une *fonction* 
- Description du problème = description de la fonction

## 4. La notion de programme

Par exemple: programme Java

Un programme (ou algorithme) est une “procédure effective”,  
c’est à dire exécutable par une machine

Compréhension de concept de programme basée sur la compréhension de “programme écrit dans un langage de programmation”

Diversité de formalismes permettant la description de “procédure effective”

# 5. Calculabilité et complexité

## Calculabilité

Recherche et analyse des limites de l'informatique

- indépendamment des réalisations physiques des machines
- indépendamment des langages et méthodes de programmation existant aujourd'hui

Résultats fondamentaux d'un point de vue

- théorique
- pratique
- philosophique (les limites de l'informatique sont-elles applicables aux êtres humains ?)

## Questions

- Existe-t-il une tâche qu'un ordinateur ne soit pas capable d'accomplir ?
- Existe-t-il un problème qui ne puisse être résolu par un algorithme ?

# 5. Calculabilité et complexité

## Complexité

Etant donné un problème, on souhaite un programme permettant de résoudre le problème

- on doit exiger qu'un programme soit *correct*
- on peut souhaiter qu'un programme correct soit *efficace*

L'efficacité est une mesure des *ressources* nécessaires à mettre en oeuvre pour que le programme produise les résultats attendus

- temps CPU
- espace mémoire nécessaire
- ...

# 5. Calculabilité et complexité

## Complexité

Définition de la notion de complexité indépendamment

- du formalisme utilisé
- de la machine utilisée
- du contexte technologique

Frontière entre le “**faisable en pratique**” et le “**infaisable en pratique**”

## Questions

- Existe-t-il un problème qu'un ordinateur puisse accomplir théoriquement, mais qui, en pratique, ne peut être résolu (incompatible avec les ressources disponibles : espace ou temps) ?



## 6. Résultats principaux

1. Equivalence des langages de programmation
2. Un problème non calculable
3. Un problème intrinsèquement complexe

# 6.1. Equivalence des langages de programmation

Existe-t-il des langages de programmation plus puissants que d'autres ?

Les langages tels que

Ada, Algol, APL, Basic, C, C++, C#, Cobol, Delphi, Erlang, Forth, Fortran, Groovy, Haskell, Java, Javascript, Lisp, Logo, Miranda, ML, Modula, ObjectiveC, Oz, Pascal, Perl, PHP, Prolog, Python, Ruby, Scala, Scheme, Smaltalk, Visual Basic, ...

sont tous ***équivalents***

Java est un langage de programmation **complet** : Il permet de résoudre les mêmes problèmes que ceux résolus par les autres langages de programmation.

En pratique, certains langages de programmation sont mieux adaptés à certaines *classes de problèmes*



## 6.2. Problème non calculable

Il existe des problèmes qui *ne peuvent être résolus* par un programme !

**Problème non calculable** : il n'existe pas de programme qui résoud ce problème

### Exemples

- Equivalence de programmes

*Précondition:* Un programme P

*Postcondition:* « 1 » si P est équivalent à un programme de tri  
« 0 » sinon

- Déterminer si un polynôme à coefficients entiers a des racines entières
- Détection de virus
- Analyse de programmes (terminaison, exactitude, ...)
- Programmation automatique

## 6.2. Problème non calculable

### Exemple: Détection de virus informatique

#### Ver (worm)

Un *ver* est un programme dont l'exécution a pour effet de se recopier (au sein d'un ordinateur ou d'un ordinateur à l'autre)

→ multitude de vers

#### Virus

Un *virus* est un fragment de programme, attaché à un programme "légitime"

Ce programme est alors *contaminé* ou *infecté*

Un programme est dit *nuisible* si son exécution a pour effet de contaminer d'autres programmes

→ multitude de programmes nuisibles

## 6.2. Problème non calculable

### Exemple: Détection de virus informatique

#### Cheval de Troie

Un *cheval de Troie* est un programme dont l'exécution produit des effets (nuisibles) en plus de ceux qui sont attendus

Un cheval de Troie est souvent combiné avec une *bombe logique*

Danger si le virus (ver) contient un cheval de Troie !

## 6.2. Problème non calculable

### Détection de virus

Pour pouvoir détecter si un programme est nuisible, il faut au moins être capable de détecter si  $P(D)$  est nuisible, c'est-à-dire ;

étant donné un programme  $P$

et une donnée  $D$  pour ce programme,

l'exécution de  $P$  avec la donnée  $D$  va-t-elle contaminer un autre programme

### Spécification : détecteur $(P,D)$

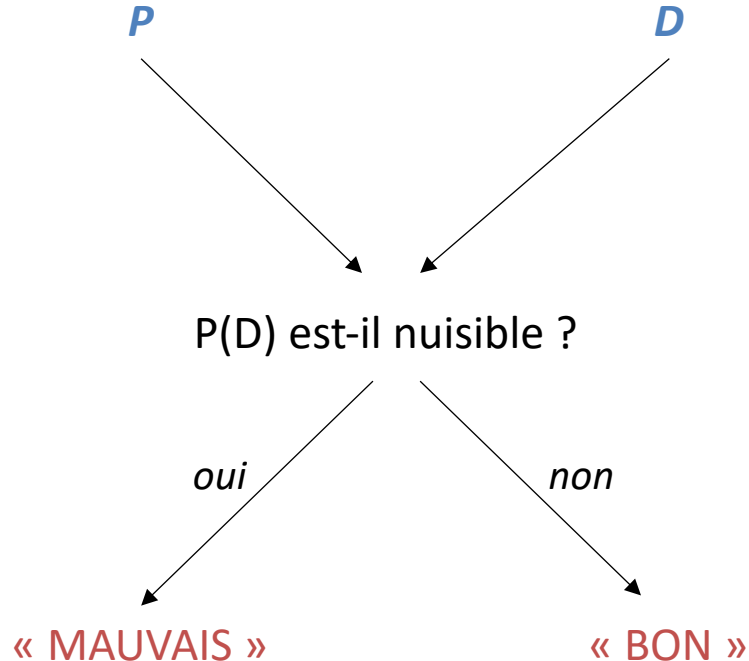
*Préconditions :*    *Un programme  $P$ , une donnée  $D$*

*Postconditions:*    « MAUVAIS » si  $P(D)$  est nuisible  
                          « BON » sinon

Existe-t-il un programme détecteur qui ne soit pas nuisible ?

## 6.2. Problème non calculable

Effet du programme détecteur (P,D)



## 6.2. Problème non calculable

### Construction d'un programme drôle( $P$ )

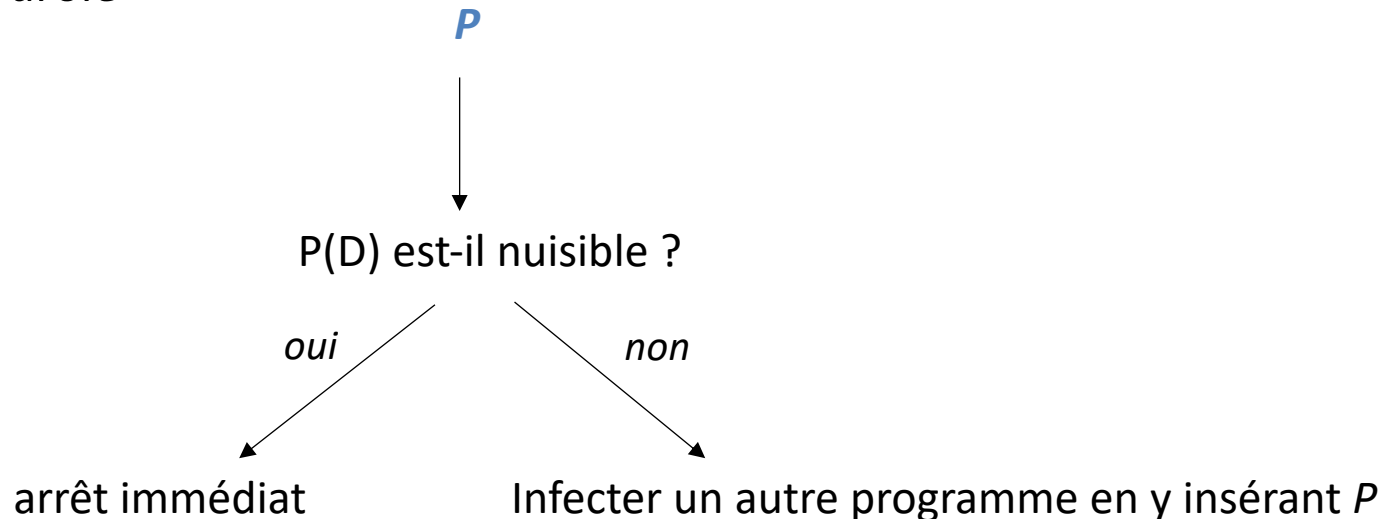
*drôle( $P$ )*

**if** *décteur*( $P, P$ ) = « MAUVAIS »

**then** arrêt immédiat

**else** infecter un autre programme en y insérant  $P$

Si le programme détecteur (non nuisible) existe, alors on peut réaliser le programme *drôle*



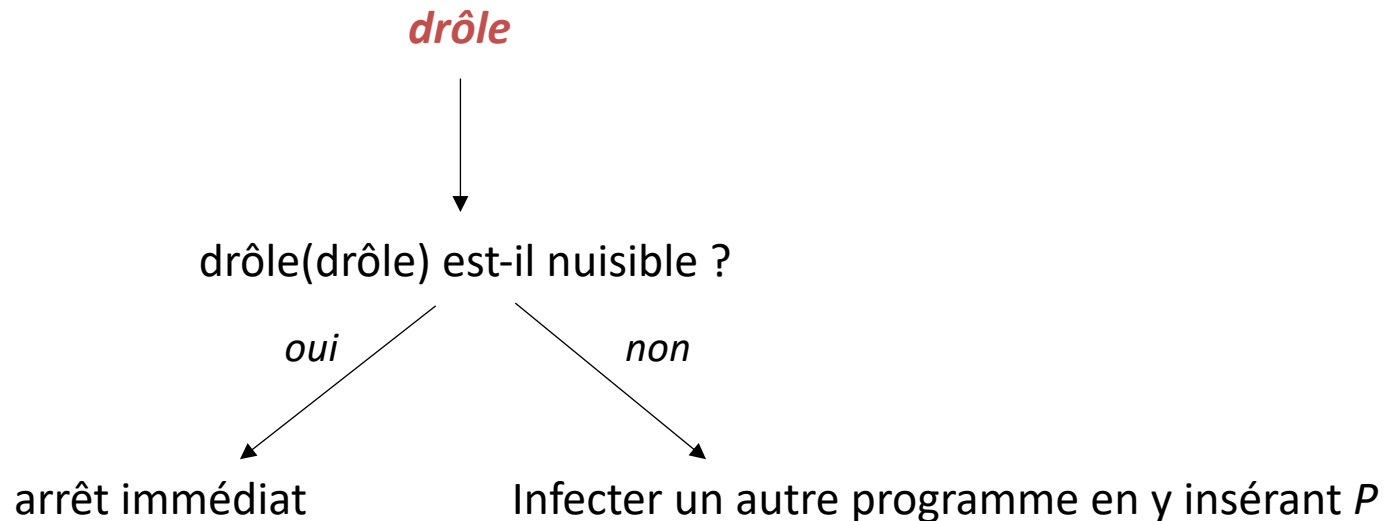
## 6.2. Problème non calculable

**Drôle(drôle) est-il nuisible ?**

**if** *décteur*(*drôle*,*drôle*)= « MAUVAIS »

**then** arrêt immédiat

**else** infecter un autre programme en y insérant *drôle*



Le programme *drôle* ne peut exister

Le programme *décteur* non nuisible ne peut exister

## 6.3. Problèmes intrinsèquement complexes

Certains problèmes, quoique théoriquement solubles par programmes, sont en pratique *infaisables*, car nécessitant des ressources incompatibles avec les réalités physiques

Problèmes *intrinsèquement complexes*



## 6.3. Problèmes intrinsèquement complexes

### Exemple : L'impossible voyage du Secrétaire Général de l'ONU

Supposons que le Secrétaire Général de l'ONU doivent (rapidement) faire le tour des capitales de  $n$  pays (par exemple  $n = 50$ )

#### Problème

Etant donnés  $n$  villes, déterminer le circuit *le plus court* reliant toutes ces villes

#### Algorithmes existants

- **Enumération**

Enumérer tous les circuits possibles  $(n-1)!$

Choisir le plus court

⇒ Complexité:  $O(2^n)$

- **Programmation dynamique**

Extension de résultats partiels

⇒ Complexité:  $O(2^{(n-1)\log n}) = O(2^n)$

Il n'existe pas de meilleurs algorithmes !

## 6.3. Problèmes intrinsèquement complexes

### Complexité : temps d'exécution

- Hypothèses:
  - ordinateur 100.000 Mips
  - traitement de 1 élément = 1.000 instructions machine

|       | 10               | 50                         | 100                           | 500             | 1000           | 10000           |
|-------|------------------|----------------------------|-------------------------------|-----------------|----------------|-----------------|
| $n$   | .0000001<br>sec. | .0000005<br>sec.           | .000001<br>sec.               | .000005<br>sec. | .00001<br>sec. | .0001<br>sec.   |
| $n^2$ | .000001<br>sec.  | .000025<br>sec.            | .0001<br>sec.                 | .0025<br>sec.   | 0.01<br>sec.   | 1<br>sec.       |
| $n^3$ | .00001<br>sec.   | .00125<br>sec.             | 0.01<br>sec.                  | 1.25<br>sec.    | 10<br>sec.     | 2.8<br>heures   |
| $n^5$ | .001<br>sec.     | 3.13<br>sec.               | 1.67<br>min.                  | 3.6<br>jours    | 115.7<br>jours | 31.7<br>siècles |
| $2^n$ | .0000102<br>sec. | 130<br>jours               | $4 \times 10^{12}$<br>siècles | ...             | ...            | ...             |
| $3^n$ | .00059<br>sec.   | $2 \times 10^5$<br>siècles | ...                           | ...             | ...            | ...             |

## 6.3. Problèmes intrinsèquement complexes

### Influence de la technologie

- Hypothèses:
  - ordinateur 100.000 Mips
  - traitement de 1 élément = 1.000 instructions machine
- $N_i$  = taille du “plus grand” exemple dont la solution peut être calculée en 1 heure de temps calcul

| Complexité | Ordinateur d'aujourd'hui   | Ordinateur 100x | Ordinateur 1000x |
|------------|----------------------------|-----------------|------------------|
| $n$        | $N_1 = 3.6 \times 10^{11}$ | $100 N_1$       | $1000 N_1$       |
| $n^2$      | $N_2 = 60000$              | $10 N_2$        | $31.6 N_2$       |
| $n^3$      | $N_3 = 7110$               | $4.64 N_3$      | $10 N_3$         |
| $n^5$      | $N_4 = 205$                | $2.5 N_4$       | $3.98 N_4$       |
| $2^n$      | $N_5 = 38$                 | $N_5 + 6$       | $N_5 + 10$       |
| $3^n$      | $N_6 = 24$                 | $N_6 + 4$       | $N_6 + 6$        |

Complexité exponentielle est intrinsèquement complexe



## 6.3. Problèmes intrinsèquement complexes

Peu importe les évolutions technologiques, un problème exponentiel **ne peut** et **ne pourra** être résolu que pour de petits exemples

Beaucoup de problèmes intéressants sont intrinsèquement complexes

### Exemples

- Problème du voyageur de commerce
- Coloration de graphes
- Beaucoup de problèmes d'optimisation
- Allocation de ressources

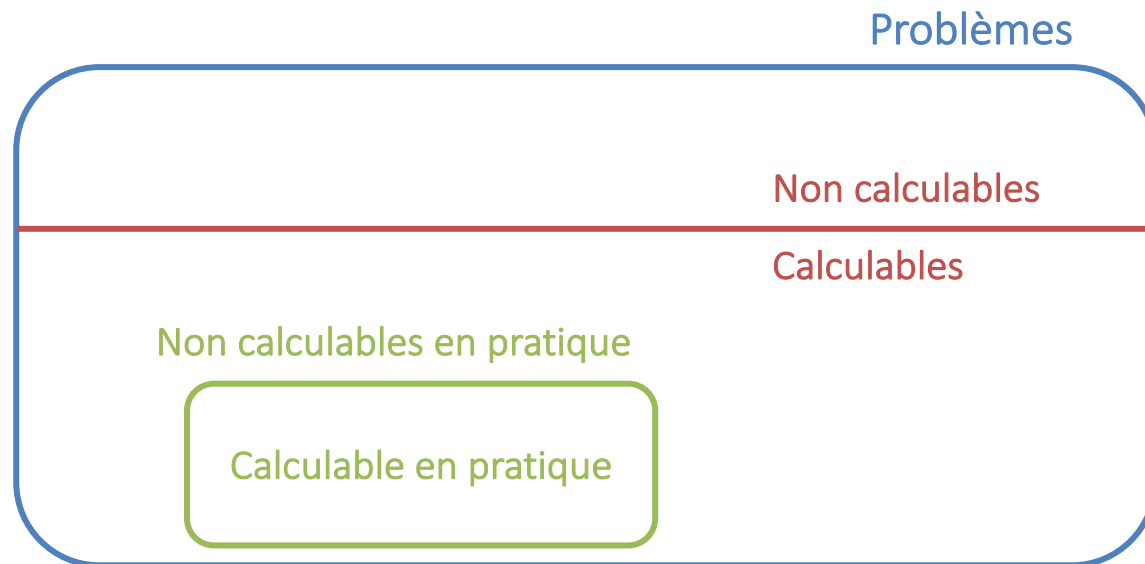
### En pratique

- Il existe des algorithmes (efficaces) permettant de calculer des **approximations** de la solution exacte
- Certains algorithmes résolvent efficacement les problèmes *pratiques*

# 7. Objectifs et Intérêts

## Objectifs calculabilité et complexité

Tracer des frontières dans l'univers des problèmes



- Ces frontières existent-elles ?
- Sont-elles précises ? Comment les définir ?
- Que faire si un problème est en dehors de ces frontières?
- Quelles sont les conséquences pratiques pour un informaticien ?

# Objectifs calculabilité et complexité

## Un peu d'histoire...

Conjecture ancienne: si un problème peut être défini avec précision, alors il existe une solution algorithmique à ce problème

- *Conjecture de Hilbert (1928) (Entscheidungsproblem)*

Il existe un algorithme pour décider si une formule (propriété) d'un système formel (e.g. arithmétique) est vraie ou fausse

- Gödel (1931), *Théorème d'incomplétude*

Réponse négative à la conjecture de Hilbert

Il existe une frontière...

Comment décrire cette frontière ?

- Nécessité de définir avec précision la notion de “calcul” ou d'algorithme

La calculabilité et ses principaux résultats sont antérieures à l'informatique

L'informatique et les ordinateurs donnent un nouvel éclairage à la calculabilité

# Intérêts calculabilité et complexité

- Intérêt pratique
  - ne pas essayer de résoudre un problème non calculable
  - être conscient de la complexité intrinsèque d'un problème
- Formation de base en informatique
  - modèles et formalismes informatiques
  - limites des algorithmes et des ordinateurs
  - langages de programmation
  - complexité
- Formation générale de l'informaticien
  - mécanismes de raisonnements originaux
  - suscite la réflexion sur les limites de l'informatique et sur les rapports hommes/machines