

# Chapitre 16 : Décidabilité et complexité

## Table des matières

<b>1</b>	<b>Décidabilité</b>	<b>2</b>
1.1	Modèles de calcul et universalité . . . . .	2
1.1.1	Introduction . . . . .	2
1.1.2	Modèles de calcul historiques (H.P) . . . . .	2
1.1.3	Modèle de calcul au programme de la MPI . . . . .	4
1.1.4	Calculabilité . . . . .	5
1.1.5	Universalité . . . . .	6

# 1 Décidabilité

## 1.1 Modèles de calcul et universalité

### 1.1.1 Introduction

L'objet de ce chapitre est l'étude de ce qu'il est possible de calculer avec un algorithme, avec ou sans contrainte de complexité temporelle.

Afin de pouvoir énoncer précisément des propriétés, il faut une définition formelle de la notion d'algorithme.

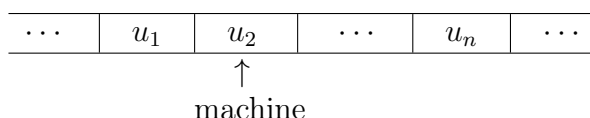
### 1.1.2 Modèles de calcul historiques (H.P)

- Nous avons vu dans le chapitre 14 plusieurs modèles de calcul, dont certains étaient équivalents (AFD, AFND,  $\varepsilon$ -AFND) et d'autres plus généraux (grammaires non contextuelle). On peut se demander s'il existe un modèle de calcul le plus général, capable de caractériser ce qu'il est possible de calculer mécaniquement.

- Plusieurs modèles de calcul ont été proposés dans les années 1930, et se sont révélés équivalents et plus généraux que les modèles précédents : les machines de TURING, et le  $\lambda$ -calcul de CHURCH.

Les modèles les plus généraux conçus ultérieurement, comme les fonctions récursives, sont équivalents à ces deux modèles et on admet l'hypothèse, nommée thèse de CHURCH-TURING, que ces modèles caractérisent vraiment la notion d'algorithme. On dit aujourd'hui qu'un langage de programmation est *Turing-complet* s'il est capable d'écrire les mêmes algorithmes que ceux implémentables par machine de TURING.

- machine de TURING : informellement, une machine de TURING est une machine finie travaillant sur un ruban infini qui lui sert de mémoire. La machine dispose d'une tête de lecture lui permettant d'accéder à une case du ruban et de règles de transition décrivant les opérations réalisées sur les cases et les déplacements de la tête de lecture.



À la manière des automates, les machines de TURING ont un ensemble fini d'états, d'où la définition suivante :

Une machine de TURING est un octuplet

$$(\Sigma, \Gamma, B, Q, q_0, q_a, q_r, \delta)$$

où

–  $\Sigma$  est l'alphabet d'entrée

.....

–  $\Gamma$  est l'alphabet de ruban, ou de travail, tel que  $\Sigma \subseteq \Gamma$  ;

–  $B \in \Gamma \setminus \Sigma$  est le symbole "blanc" représentant les cases vides ;

–  $Q$  est un ensemble fini non vide d'états ;

–  $q_0 \in Q$  est l'état initial ;



- $q_a, q_r \in Q$  sont les états finaux de la machine :  $q_a$  est appelé l'état acceptant,  $q_r$  l'état rejetant ;
- $\delta$  est la fonction de transition :

$$\delta : (Q \setminus \{q_a, q_r\}) \times \Gamma \longrightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$$

$$(q, a) \longmapsto (q', b, d)$$

Si  $\delta(q, a) = (q', b, d)$ , alors lorsque la machine lit le symbole  $a$  sur le ruban en étant dans l'état  $q$ , elle écrit le symbole  $b$  à la place, déplace sa tête de lecture selon le déplacement  $d$  et passe dans l'état  $q'$ .

La machine accepte un mot  $u \in \Sigma^*$  si et seulement si, en partant de l'état initial avec le ruban  $B^\infty u B^\infty$  et la tête de lecture sur la première de  $u$  (si elle existe), l'exécution mène à l'état  $q_a$ . Elle rejette  $u$  si et seulement si elle atteint l'état  $q_r$  et elle peut également ne pas terminer.

Une machine de TURING peut aussi calculer une fonction : l'argument est placé sur le ruban et le contenu du ruban à la fin de l'exécution est le résultat de la fonction.

• Exemple :

$$M = (\{0, 1\}, \{0, 1, B\}, B, \{q_0, q_1, q_2, q_a, q_r\}, q_0, q_a, q_r, \delta)$$

où  $\delta$  est définie par la table :

	0	1	B
$q_0$	$(q_0, 0, \rightarrow)$	$(q_1, 1, \rightarrow)$	$(q_a, B, \leftarrow)$
$q_1$	$(q_1, 0, \rightarrow)$	$(q_1, 1, \rightarrow)$	$(q_2, B, \leftarrow)$
$q_2$	$(q_a, B, \rightarrow)$	$(q_a, B, \rightarrow)$	/

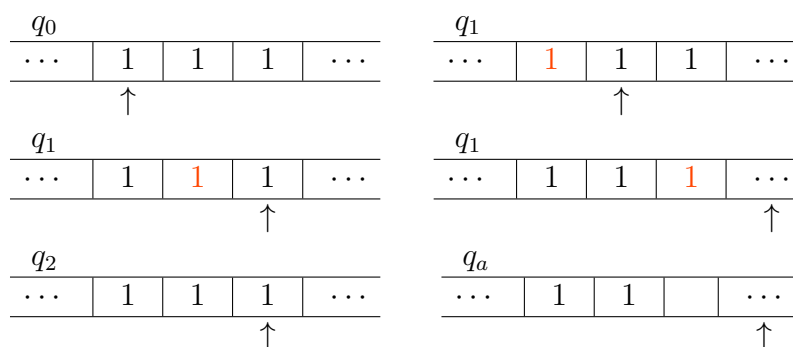
Avec  $\langle 000 \rangle_2$  en entrée :

$q_0$	$q_0$
...	...
0	0
0	0
0	0
...	...
↑	↑
$q_0$	$q_0$
...	...
0	0
0	0
0	0
...	...
↑	↑
$q_a$	$q_a$
...	...
0	0
0	0
0	0
...	...
↑	↑

Avec  $\langle 010 \rangle_2$  en entrée :

$q_0$	$q_0$
...	...
0	0
1	1
0	0
...	...
↑	↑
$q_1$	$q_1$
...	...
0	0
1	1
0	0
...	...
↑	↑
$q_2$	$q_a$
...	...
0	0
1	1
0	
...	...
↑	↑

Avec  $\langle 111 \rangle_2$  en entrée :



Cette machine de TURING calcule la fonction

$$\begin{array}{ccc} \{0, 1\}^* & \longrightarrow & \{0, 1\}^* \\ x & \longmapsto & y \end{array}$$

avec  $\langle y \rangle_2 = \left\lfloor \frac{\langle x \rangle_2}{2} \right\rfloor$

On dit plus généralement que la machine calcule

$$\begin{array}{ccc} \mathbb{N} & \longrightarrow & \mathbb{N} \\ n & \longmapsto & \left\lfloor \frac{n}{2} \right\rfloor \end{array}$$

De même, si on supprime  $q_2$  et on remplace  $\delta$  par la table

	0	1	$B$
$q_0$	$(q_0, 0, \rightarrow)$	$(q_0, 1, \rightarrow)$	$(q_1, B, \leftarrow)$
$q_1$	$(q_a, 0, \rightarrow)$	$(q_r, 1, \rightarrow)$	/

On obtient une machine qui reconnaît le langage  $\{u \in \{0, 1\}^* \mid \langle u \rangle_2 \equiv 0 [2]\}$ .

Autrement dit, la machine calcule  $1/2\mathbb{N}$

- Les modèles du  $\lambda$ -calcul et des machines de TURING sont pertinents pour l'étude théorique de la calculabilité grâce à leur "simplicité", mais ne sont pas pratiques pour l'écriture d'algorithmes concrets. C'est pourquoi on utilise en général un modèle différent pour l'écriture des algorithmes.

### 1.1.3 Modèle de calcul au programme de la MPI

Le modèle de calcul considéré est celui d'un programme C ou OCaml qui s'exécute sur une machine à mémoire infinie.

En particulier, il n'y a jamais de dépassement de capacité de la pile d'exécution et on peut toujours allouer de la mémoire sur le tas.

Il est aisé de simuler une machine de TURING avec un programme C ou OCaml. Le sens réciproque est beaucoup plus difficile.

Nous appelons *algorithme* tout objet qui est un programme C ou OCaml, une machine de TURING, ou un  $\lambda$ -calcul.

On s'autorisera l'usage du pseudo-code pour l'écriture d'algorithmes.



### 1.1.4 Calculabilité

Le terme *fonction* étant ambigu (fonction mathématique, fonctions dans un programme), on utilisera ce terme uniquement pour les fonctions mathématiques. Un algorithme peut être vu comme une réalisation d'une fonction mathématique partielle : celle qui à chaque entrée de l'algorithme sur la pile d'exécution se termine associe la valeur de retour de l'algorithme.

- Définition (*fonction calculable*) : une fonction  $f : A \rightarrow B$  est dite *calculable* s'il existe un algorithme  $M$  tel que  $\forall a \in A$ , l'exécution de  $M$  sur  $a$  termine en temps fini, et renvoie  $f(a)$ .

- Remarque :

- Il est "facile" de montrer qu'une fonction est calculable : il "suffit" d'exhiber un algorithme qui convient.

Il est en revanche beaucoup plus difficile de montrer qu'une fonction n'est pas calculable : il faut montrer qu'aucun algorithme ne peut convenir.

- On va se limiter aux fonctions de  $\mathbb{N} \rightarrow \mathbb{N}$ . En effet :

- + si  $A$  est fini, on peut se contenter de tabuler les valeurs de  $f$  et d'écrire un algorithme qui va chercher dans la table la bonne valeur.

- + Si  $A$  est indénombrable, on a un problème de représentation de l'entrée : on a besoin de représentation infinies, ce qui est peu pertinent dans l'optique d'étudier ce qu'une machine réelle peut calculer.

- + On utilise en général des encodages pour représenter les données manipulées et un encodage binaire peut être vu comme un entier non signé (donc un entier naturel).

- Proposition :

| Il existe une infinité de fonctions non calculables.

□

Il suffit de montrer que  $\mathbb{N}^{\mathbb{N}}$  est indénombrable car l'ensemble des algorithmes est dénombrable (l'ensemble des codes sources s'injecte dans l'ensemble des chaînes de caractères, dénombrable).

$\mathbb{N}^{\mathbb{N}}$  est indénombrable car  $\{0, 1\}^{\mathbb{N}}$  l'est déjà d'après le théorème de CANTOR (on voit une fonction  $\mathbb{N} \rightarrow \{0, 1\}$  comme la fonction indicatrice d'une partie de  $\mathbb{N}$ ).

Argument diagonal (pour montrer que  $\{0, 1\}^{\mathbb{N}}$  n'est pas dénombrable) : on suppose que  $\{0, 1\}^{\mathbb{N}}$  est dénombrable. Alors on peut numérotter les suites de  $\{0, 1\}$  :

$s_0$	0	0	0	...
$s_1$	1	0	0	...
$s_2$	1	1	1	...
$\vdots$				

On a  $(s_n)_{n \in \mathbb{N}} \in \left(\{0, 1\}^{\mathbb{N}}\right)^{\mathbb{N}}$ . Soit  $\forall n \in \mathbb{N}, u_n = 1 - s_n$ . Alors :

$$(u_n)_{n \in \mathbb{N}} \in \{0, 1\}^{\mathbb{N}}$$

Mais  $\forall n \in \mathbb{N}, s_n \neq (u_k)_{k \in \mathbb{N}}$  : absurde. ■

### 1.1.5 Universalité

- On utilise souvent un argument diagonal pour montrer qu'une fonction n'est pas calculable : on procède par l'absurde en supposant l'existence d'un algorithme  $M$  convenable et en construisant un algorithme qui utilise  $M$ , souvent en faisant référence à son propre code source, pour aboutir à une absurdité (cf ??, page ??).

Ce type de démonstration nécessite deux propriétés essentielles :

- L'autoréférence, *i.e* la possibilité de faire référence à son propre code source. C'est possible car l'ensemble des algorithmes est dénombrable : on peut faire référence à un algorithme par son numéro.
- La simulation : il faut pouvoir simuler l'exécution d'un algorithme afin d'exploiter le résultat.

- Théorème :

Il existe un algorithme, appelé machine universelle, d'entrée un algorithme  $M$  et une entrée  $x$  pour  $M$ , qui simule l'exécution de  $M$  sur  $x$ .

□

(démonstration informelle)

On passe par les machines de TURING.

Comme les machines à plusieurs rubans sont équivalentes aux machines à un ruban, on construit une machine qui a le code de  $M$  sur son ruban d'entrée, l'état courant de  $M$  dans l'exécution sur  $x$ , dans un deuxième ruban et le ruban de travail de  $M$  dans un troisième ruban. ■