

Chapitre I

Modèles abstraits de machines

Normalement, ce cours devrait commencer par l'étude du modèle des machines de Turing.

Nous allons repousser un peu l'étude de ce modèle formel afin de vous permettre d'en assimiler les prérequis dans le cours sur les langages.

Dans ce chapitre, nous allons donc expliquer ce qui doit être compris dans le cours sur les langages.

- tout objet informatique peut être codé comme un mot d'un langage formel,
- résoudre un problème informatique décisionnel est équivalent à reconnaître un langage.

1 Alphabets et mots

1.1 Prérequis

Définition I.1 (cardinal d'un ensemble)

Le cardinal d'un ensemble E est le nombre d'éléments qu'il contient.
On le note $\#E$.

Définition I.2 (ensemble fini)

Un ensemble E est fini s'il existe $k \in \mathbb{N}$ tel que $\#E = k$.

Autrement dit, un ensemble fini n'est pas infini.

EXEMPLES 1:

- L'ensemble des chiffres en base 10 est fini.
- L'ensemble des nombres en base 10 est infini.

Définition I.3 (Ensemble des parties d'un ensemble)

On appelle l'ensemble des parties d'un ensemble E l'ensemble des ensembles qu'il est possible de construire à partir de cet ensemble.

Notation : on note $\mathcal{P}(E)$ l'ensemble des parties de l'ensemble E .

EXEMPLE 2: Soit l'ensemble $E = \{a, b, c\}$.

Alors $\mathcal{P}(E) = \{\varepsilon, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a, b, c\}\}$

REMARQUE 0 : $\#\mathcal{P}(E) = 2^{\#E}$

En effet, pour tout sous-ensemble de E , chaque élément a 2 choix : y appartenir ou pas. Comme il y a $\#E$ élément dans E , il est donc possible de construire $2^{\#E}$ ensembles différents.

1.2 Alphabet

Définition I.4 (alphabet)

Un **alphabet** Σ est un ensemble **fini** de symboles.

EXEMPLES 3: d'alphabets

- $\Sigma = \{a, b, \dots, z\}$: alphabet des lettres (de l'alphabet).
- $\Sigma = \{0, 1\}$: alphabet binaire.
- $\Sigma = \{0, 1, 2, \dots, 9\}$: alphabet des chiffres de la base 10.
- ...

REMARQUES 1:

- Le symbole Σ dénotera toujours un alphabet.
- Si sa définition n'est pas précisée, on prendra $\Sigma = \{0, 1\}$.
- **(redite)** La taille d'un alphabet est toujours finie.

1.3 Mot

Définition I.5 (mot)

Un **mot** issu d'un alphabet Σ est une suite finie de symboles dont chacun appartient à Σ .

Remarque : Un mot peut être vide. On note ε le mot vide.

Exemples de mots :

- d est un mot issu de l'alphabet $\{a, b, \dots, z\}$.
- 01101 est un mot issu de l'alphabet $\{0, 1\}$.
- ε est un mot issu de tout alphabet.

Définition I.6

La longueur d'un mot est le nombre total de symboles dont il est composé.

Notation : si m est un mot, on notera $|m|$ sa longueur.

Exemples :

- $|\varepsilon| = 0$.
- si $m = 01101$, alors $|m| = 5$.

On note m^k représente le mot m répété k fois.

exemple : si $m = 01$ alors $m^5 = 0101010101$

Définition I.7 (opérateurs réguliers sur des mots)

L'ensemble des opérations régulières est :

- l'**union** \cup : si m_1 et m_2 sont deux mots, alors $m_1 \cup m_2$ est l'ensemble $\{m_1, m_2\}$.
- la **concaténation** \circ : si m_1 et m_2 sont deux mots, alors $m_1 \circ m_2 = m_1 m_2$.
- l'**opérateur** $*$ (étoile de Kleene) : si m est un mot, alors m^* est la répétition de m un nombre quelconque de fois, à savoir $m^* = \{\varepsilon\} \cup \bigcup_{k=1}^{\infty} \{m^k\}$.

EXEMPLES 4: si $m_1 = ab$ et $m_2 = c$

- $m_1 \cup m_2$ est l'ensemble de mots $\{ab, c\}$.
- $m_1 \circ m_2$ est le mot abc .
- m_1^* est l'ensemble de mots $\{\varepsilon, ab, abab, ababab, abababab, \dots\}$.

Par extension, on notera :

- Σ^k l'ensemble tous les mots de longueur k issus de l'alphabet Σ .
exemple : si $\Sigma = \{0, 1\}$, alors $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
- Σ^* l'ensemble des mots de toute taille qu'il est possible de construire avec l'alphabet Σ , y compris la chaîne vide ε .
 $\Sigma^* = \{\varepsilon\} \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots$
exemple : si $\Sigma = \{0, 1\}$, alors Σ^* contient toutes les chaînes binaires possibles de longueur finie.
Attention, Σ^* ne contient pas les chaînes binaires de longueur infinie.

2 Langages

2.1 Langage formel

Définition I.8 (langage formel)

Un **langage** L défini sur un alphabet Σ est un sous-ensemble de Σ^* .

Un langage L sur Σ est un ensemble de mots construit à partir de l'alphabet Σ .

Notes :

- un langage peut ne contenir aucun mot (i.e. $L = \emptyset$).
- le nombre de mots dans un langage n'est pas nécessairement fini.

Exemples de langages :

- pour $\Sigma = \{a, b, \dots, z\}$, l'anglais est un langage.
- pour $\Sigma = \{0, 1, \dots, 9\}$, les ensembles suivants sont des langages :
— les nombres entre 1 et 100

- les nombres pairs
- les nombres premiers
- pour $\Sigma = \{0, 1\}$, les ensembles suivants sont des langages :
 - $\{w \text{ tels que } w \text{ contient au moins 12 fois le symbole } 0\}$
 - $\{0^n 1^n \text{ tel que } n \geq 0\}$
 - $\{w \text{ tels que } w \text{ a le même nombre de symboles } 0 \text{ et } 1\}$

2.2 Opérateurs réguliers sur un langage

Définition I.9 (opérateurs réguliers sur des langages)

Soit L_1 et L_2 deux langages. L'ensemble des opérations régulières est :

- l'**union** $\cup : L_1 \cup L_2 = \{m \mid m \in L_1 \text{ ou } m \in L_2\}$.
- la **concaténation** $\circ : L_1 \circ L_2 = \{m_1 m_2 \mid m_1 \in L_1 \text{ et } m_2 \in L_2\}$.
- l'**opérateur** $*$: $L^* = \{\varepsilon\} \cup \bigcup_{k=1}^{\infty} \{m_1 m_2 \dots m_k \mid \forall i, m_i \in L\}$.

Note : le résultat d'un opérateur régulier sur un ou plusieurs langages est un langage.

EXEMPLE 5: Soit le langage $L_1 = \{0, 11\}$ construit sur l'alphabet $\Sigma_1 = \{0, 1\}$ et $L_2 = \{a, bb, ccc\}$ construit sur l'alphabet $\Sigma_2 = \{a, b, c\}$.

- $L_1 \cup L_2$ est le langage $\{0, 11, a, bb, ccc\}$.
- $L_1 \circ L_2$ est le langage $\{0a, 0bb, 0ccc, 11a, 11bb, 11ccc\}$.
- L_1^* est le langage $\{\varepsilon, 0, 11, 00, 011, 110, 1111, 000, 0011, 0110, 01111, 1100, 11011, 11110, 111111, 0000, 00011, 00110, 001111, 01100, 011011, 011110, 0111111, 11000, 110011, 110110, 1101111, 111100, 1111011, 1111110, 11111111, \dots\}$.

Cet ensemble n'est pas de taille finie.

2.3 Langages réguliers

Définition I.10 (Langage régulier)

Un langage sur un alphabet Σ est régulier s'il peut être défini comme un ensemble fini de mots de Σ^* ou de toute composition finie d'opérateurs réguliers sur Σ .

EXEMPLES 6: sur $\Sigma = \{0, 1\}$

- $\{010\} \cup \{010\}$ est un langage régulier.
- $\{01\}^*$ est un langage régulier.
- $(\{010\} \cup \{010\}) \circ \{01\}^*$ est un langage régulier.
- Σ^{10} est un langage régulier (l'ensemble des mots binaires de taille 10 peut être construit comme une union finie de mots de taille 10).
- $\{0^n 1^n \mid \forall n \leq 10\} = \varepsilon \cup \{01\} \cup \{0011\} \cup \dots \cup \{0^{10} 1^{10}\}$ est un langage régulier.
- $\{0^n 1^n \mid \forall n > 0\}$ n'est pas un langage régulier, car cet ensemble ne peut pas être écrit comme une composition finie d'opérateurs réguliers.

REMARQUES 2:

- On notera que tout langage fini est un langage régulier, car il peut être écrit comme l'union finie des mots qu'il contient.
- Le lemme de l'étoile permet de vérifier si un langage régulier (voir votre cours sur la théorie des langages).

2.4 Langages algébriques**Définition I.11** (Grammaire algébriques)

Une grammaire algébrique G est un ensemble de règles de construction de mots basés sur :

- un alphabet Σ .
- une variable de départ,
- des variables intermédiaires,
- des règles de transformations qui transforment une variable en toute concaténation de variables intermédiaires et de symboles de l'alphabet.

EXEMPLE 7: Soit la grammaire algébrique G définie par l'alphabet $\Sigma = \{0, 1\}$, la variable de départ S , la variable intermédiaire U , et les règles de transformation $S \rightarrow 1U0$, $U \rightarrow 0U1$ et $U \rightarrow \varepsilon$.

Définition I.12 (Mot engendré par une grammaire algébrique)

Un mot w est engendré par une grammaire algébrique G :

- en partant de la variable de départ,
- en appliquant des règles de transformation jusqu'à ce que le mot obtenu ne contiennent plus que des symboles de l'alphabet.

EXEMPLE 8: $S \xrightarrow{S \rightarrow 1U0} 1U0 \xrightarrow{U \rightarrow 0U1} 10U10 \xrightarrow{U \rightarrow 0U1} 100U110 \xrightarrow{U \rightarrow \varepsilon} 100110$.

Définition I.13 (Langages algébriques)

Un langage algébrique est un langage qui contient l'ensemble des mots engendrés par une grammaire algébrique.

EXEMPLE 9: L'ensemble des mots engendrés par la grammaire précédente est $\{10, 1010, 10^2 1^2 0, 10^3 1^3 0, \dots\}$.

3 Langages et modèles

Les langages réguliers et algébriques ont des applications directes en informatique, par exemple :

- avec les expressions régulières pour les langages réguliers,
- avec la syntaxe de tout objet informatique (langages programmations, représentation des données, ...) pour les langages algébriques.

Afin de reconnaître les mots issus d'un langage L , des modèles de machines élémentaires sont utilisés. On utilise :

- un automate fini permet de reconnaître si une chaîne appartient à un langage régulier.
- un automate à pile permet de vérifier si un code n'a pas d'erreur de syntaxe ou si des données lues respectent bien une structure prédéterminée.

Si A est l'automate qui reconnaît le langage L , alors $A(w)$ accepte si $w \in L$, et rejette sinon.

Chaque automate est donc spécialisé pour un langage particulier.

Essentiellement, ces automates sont des machines à état auxquels on donne un mot en entrée :

- dont les symboles sont lus l'un après l'autre (dans l'ordre),
- chaque symbole lu provoque un changement d'état (= une transition),
- et qui n'est accepté que si le dernier symbole lu conduit à un état acceptant, et rejeté sinon.

De manière un peu plus formelle,

- si L est un langage régulier, alors il existe un automate fini A tel que, pour tout mot w de L , $A(w)$ accepte si $w \in L$, et rejette sinon.
- si L est un langage algébrique, alors il existe un automate à pile A tel que, pour tout mot w de L , $A(w)$ accepte si $w \in L$, et rejette sinon.

L'inverse est aussi vrai : si A est un automate fini (resp. à pile), alors le langage L reconnu par A est un langage régulier (resp. algébrique).

redite : un automate A ne reconnaît donc qu'un seul langage L . On note alors $L = \mathcal{L}(A)$ afin d'indiquer que le langage reconnu par A est L .

3.1 Non déterminisme

Un automate fini est par essence **déterministe**.

On entend pas là que, pour chaque état dans lequel l'automate se trouve, il existe n'existe qu'un seul choix possible pour traiter le symbole courant du mot en cours d'analyse, et passer à l'état suivant (= faire une transition).

Définition I.14 (Transition non déterministe)

Une transition est non déterministe s'il peut exister plusieurs choix possibles pour traiter le symbole courant, voir même aucun.

Dans ce cas,

- s'il existe plusieurs choix, la machine se duplique et poursuit toutes les exécutions possibles (= plusieurs branches d'exécution)
- s'il n'existe aucun choix, la branche d'exécution s'arrête.
- le mot est accepté si au moins une branche accepte.

REMARQUES 3:

- Un automate fini peut s'écrire sous forme déterministe ou non déterministe.
- Un automate à pile est, par définition, non déterministe.

Une exécution non déterministe a les caractéristiques suivantes :

- l'exécution d'une machine peut se représenter sous forme d'un arbre d'exécution, où chaque nœud correspond à un choix non déterministe (son arité est le nombre de choix).
- le temps d'exécution d'une machine non déterministe est celui de la branche acceptante de plus petite profondeur.
- le nombre de branches non déterministe est potentiellement exponentiel.

Le **non-déterminisme n'est pas du parallélisme** au sens où :

- lancer des threads supplémentaires à chaque choix non déterministe augmente la charge de la machine,
- le temps d'exécution dépendra nécessairement du nombre de processeurs nécessaires, ou du nombre de qubits sur un ordinateur quantique.

En conséquence, une machine non déterministe n'est pas physiquement réalisable.

3.2 Limitations

Mais ces modèles souffrent de beaucoup de limitation :

- tout ce qui n'est pas de l'ordre de l'expression régulière ou relever de la syntaxe ne peut pas être reconnu.
- un automate ne reconnaît qu'un seul langage (un automate n'est pas programmable). Ils sont donc l'équivalent d'un processeur spécialisé qui ne serait câblé pour ne réaliser qu'une seule tâche.

Vous allez étudier essentiellement deux types de modèles en théorie des langages :

- les automates finis
- les automates à pile

Leurs capacités sont néanmoins bien trop limitées. Par exemple, on peut :

- savoir si un mot est un palindrome mais pas s'il est de la forme ww où w est un autre mot,
- savoir si un mot contient un entier en base 10, mais pas s'il est premier,
- savoir si un mot contient la description syntaxique d'un graphe, mais pas si le graphe est cohérent correctement,
- ...

3.3 Codage

Convenons tout d'abord qu'une chaîne de symboles aléatoires n'a pas de sens propre (tout ce à quoi on peut donner du sens dans une telle chaîne est le résultat d'un événement aléatoire).

Le codage d'une information avec les symboles d'un alphabet est obtenu :

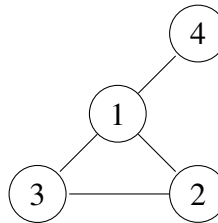
- en donnant un sens aux symboles,
- en organisant ceux-ci de manière à ce que leur interprétation ait un sens.

Autrement dit, un codage est la grammaire d'un langage.

EXEMPLE 10: Le codage de l'entier 212 en binaire est la chaîne "11010100" ne prend son sens qu'en prenant chacun de ses symboles $s_7 \dots s_1 s_0$, et en construisant sa valeur $v = \sum_{i=0}^7 s_i 2^i$. Un autre codage binaire serait "AAZAZAZZ" avec le sens A=1 et Z=0. Donc le codage d'un entier est déterminé par le choix de la base (ou toute autre méthode de codage) et par le choix de l'alphabet (sens des symboles).

On peut ainsi par exemple définir les grammaires suivantes :

- un **entier** avec comme une suite finie de chiffres par parmi $\{0, 1, 2, \dots, 9\}$.
- une **liste d'entiers** (12, 34, 56, 78) avec une "liste d'entiers séparés par des virgules et délimitée par des parenthèses."
- un **graphe** avec la "composition d'une liste d'entiers représentant les sommets, et d'une liste de couples d'entiers représentant les arêtes entre les sommets".



(1; 2; 3; 4)((1; 2); (2; 3); (3; 1); (1; 4))

- ...

Évidemment, différents types de codage auraient pu être utilisés pour ces mêmes objets. **Le codage utilisé n'a pas d'importance**, seul compte le fait que cela peut être codé.

Tout objet utilisable dans un traitement informatique peut ainsi être codé sous forme de grammaire.

REMARQUE 4:

On notera que tout mot accepté par la grammaire ci-dessus associée à un graphe ne représente pas nécessairement un graphe.

- la liste des arêtes ne doit utiliser des identifiants de sommet se trouvant dans la liste de sommets,
- chaque sommet dans la liste des sommets doit avoir un identifiant unique,
- ...

Il peut donc exister des mots **grammaticalement corrects** mais **sémantiquement faux**, et qui nécessitent des vérifications supplémentaires.

REMARQUE 5:

On utilisera les notations suivantes :

- si O est un objet, alors $\langle O \rangle$ représente l'encodage d'un objet,
- si O_1, O_2, \dots, O_k sont des objets, alors $\langle O_1, O_2, \dots, O_k \rangle$ représente l'encodage de tous ces objets,

et ceci sans se soucier **ni de l'alphabet, ni du codage utilisé à cet effet**.

EXEMPLES 11:

- **MATRIX** = $\{\langle M \rangle \mid M \text{ est une matrice}\}$
MATRIX est le langage constitué de l'ensemble des mots qui représentent une matrice.
- **INT_LIST** = $\{\langle L \rangle \mid L \text{ est une liste d'entiers}\}$
INT_LIST est le langage constitué de l'ensemble des mots qui représentent une liste d'entiers.
- **EXAMPLE** = $\{\langle M, L_1, L_2 \rangle \mid M \in \text{MATRIX et } L_1, L_2 \in \text{INT_LIST}\}$
EXAMPLE est le langage constitué d'un triplet constitué d'une matrice et de deux listes d'entiers.

4 Introduction aux machines de Turing

4.1 Modèle théorique

Une machine de Turing M est essentiellement un automate déterministe fini avec une bande de mémoire dans laquelle il peut, lire, écrire et se déplacer. Le mot à analyser est placé au début la bande.

Les concepts d'automate fini étant abordé en cours de langage, nous repoussons l'étude du fonctionnement exact de la machine de Turing pour un peu plus tard.

Comme pour les automates précédents, une MT M qui reconnaît un langage L , et fonctionne de la même manière suivantes :

- si $w \in L$, alors $M(w)$ accepte,
- si $w \notin L$, alors $M(w)$ rejette ou boucle.

De la même façon qu'un ordinateur peut boucler, la possibilité qu'une MT le puisse également n'est donc pas étonnante.

Au final, une MT a les mêmes capacités qu'un ordinateur pour reconnaître un langage.

Avant de décrire le modèle théorique, nous considérerons qu'une machine de Turing comme un ordinateur classique qui :

- prend en paramètre une chaîne de symboles,
- ne sait répondre que deux choses : oui (accepter) ou non (rejet) suivant que le paramètre appartienne ou non au langage reconnu par la machine.

et pour laquelle la description de ce qu'elle exécute est décrit sous forme d'un pseudo-code **effectuant uniquement des opérations élémentaires**.

EXEMPLE 12: de machine qui reconnaît le langage $L = \{xx \mid x \in \Sigma^*\}$ Ce langage contient tous les mots qui contiennent un mot qui se répète deux fois. Un code de MT M qui reconnaît L pourrait être :

```

M(w) =
  n = |w|
  SI n == 0 OU mod(n,2) == 1 ALORS REJETER
  p = n/2
  POUR i = 0 à p-1
    SI w[i] != w[p+i] ALORS REJETER
  ACCEPTER

```

4.2 Machine de Turing universelle

Pour les MTs classiques, nous avons :

- $\mathcal{L}(M)$ est le langage reconnu par la MT M .
- M_L indique que la MT qui reconnaît le langage L .
Évidemment, $\mathcal{L}(M_L) = L$.
- l'ensemble des mots reconnus par une MT M peut s'écrire :
 $\mathcal{L}(M) = \{w \mid M(w) \text{ accepte}\}.$

Une MT universelle U est une MT qui permet de simuler une autre MT M sur n'importe quelle entrée w , à savoir :

$$U(\langle M, w \rangle) = \begin{cases} \text{accepte} & \text{si } M(w) \text{ accepte} \\ \text{rejette} & \text{si } M(w) \text{ rejette ou boucle} \end{cases}$$

Le langage reconnu par U est :

$$\mathcal{L}(U) = \{\langle M, w \rangle \mid M(w) \text{ accepte}\}$$

Ce langage contient l'ensemble des mots couples de codes des machines M et d'entrées w tels que M accepte w .

Une MTU est donc un modèle de MT programmable.

L'existence d'une MTU implique la possibilité d'écrire son code équivalent :

$$U(\langle M, w \rangle) = \begin{array}{ll} \text{EXÉCUTER } M(w) & // \text{ simulation de } M \text{ sur l'entrée } w \\ \text{DÉCIDER comme } M & \end{array}$$

On peut ainsi écrire des codes permettant de contrôler d'appeler d'autres machines ou de contrôler finement l'exécution :

1. appel à plusieurs machines :

$$D(\langle M_1, M_2, w \rangle) = \begin{array}{ll} \text{EXÉCUTER } M_1(w) & // \text{ simulation de } M_1 \text{ sur l'entrée } w \\ \text{EXÉCUTER } M_2(w) & // \text{ simulation de } M_2 \text{ sur l'entrée } w \\ \text{SI } M_1 \text{ ET } M_2 \text{ acceptent ALORS ACCEPTER SINON REJETER} & \end{array}$$

2. exécution pas à pas plusieurs machines en parallèle :

$$D(\langle M_1, M_2, w \rangle) = \begin{array}{l} \text{RÉPÉTER} \\ \quad \text{EXÉCUTER } M_1(w) \text{ sur une transition} \\ \quad \text{EXÉCUTER } M_2(w) \text{ sur une transition} \\ \quad \text{SI } M_1 \text{ ACCEPTE ALORS ACCEPTER} \\ \quad \text{SI } M_2 \text{ ACCEPTE ALORS ACCEPTER} \end{array}$$

L'exécution sur une transition signifie que la machine s'exécute pas à pas.

Ceci est utile lorsqu'on exécute des machines qui peuvent boucler.

3. construction du code d'une machine à exécuter :

$$D(\langle M, w \rangle) = \begin{array}{l} \text{CONSTRUIRE le code:} \\ \quad M_1(x) = \text{SI } x \neq w \text{ ALORS REJETER} \\ \quad \quad \text{EXÉCUTER } M(\langle w \rangle) \\ \quad \quad \text{DÉCIDER } M \\ \text{EXÉCUTER } M_1(w) \\ \text{DÉCIDER comme } M_1 \end{array}$$

Exercice 1 (Langage et machine de Turing). Donner les langages reconnus par chacune des machines de Turing précédentes.

Ces langages L seront écrits sous la forme :

$$L = \{\langle w \rangle \mid \text{ensemble des mots } w \text{ tels que } \dots\}$$

où la forme de $\langle w \rangle$ sera adaptée en fonction du langage.

Pour la dernière machine, on commencera par décrire le langage reconnu par la machine M_1 .

4.3 Fonction calculable

Une fonction calculable est un type de MT qui retourne une valeur.

Définition I.15 (fonction (totalement) calculable)

Une fonction totalement calculable est une MT qui évalue une fonction $f : \Sigma^* \rightarrow \Sigma^*$. Elle commence avec l'entrée w sur sa bande, et s'arrête pour tout w avec la valeur $f(w)$ écrit sur sa bande.

Définition I.16 (fonction partiellement calculable)

Une fonction partiellement calculable est MT qui évalue une fonction f dont la valeur $f(w)$ ne peut pas être calculée pour certaines entrées w . Dans ce dernier cas, la MT retourne alors un caractère spécial \perp ou boucle.

REMARQUES 6:

- marche aussi pour les fonctions avec plus d'une variable.
Encoder les paramètres sur la chaîne d'entrée intercalée de séparateurs.
Chaque paramètre peut représenter n'importe quel objet.
- même remarque pour la sortie.
- une bande particulière peut être utilisée pour la sortie.
- Toutes les fonctions arithmétiques classiques sur les entiers sont totalement calculables : addition, soustraction, multiplication, division (quotient, reste), ...
- Toutes les fonctions non-arithmétiques sont également totalement calculables : trigonométrique, logarithmique, ... en utilisant les séries de Taylor (ou d'autres techniques d'approximation) à une précision spécifiée.

voir "Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables" d'Abramowitz et Stegun sur la façon pratique de réaliser ces approximations à une précision donnée.

5 Modèle de calcul

Un modèle de calcul peut s'interpréter comme une évaluation fonctionnelle de la forme $y = f(x)$ où f est la fonction qui résout le problème de trouver la sortie y à partir de l'entrée x .

L'adaptation d'un tel modèle pour correspondre à un calcul informatique suppose :

- le choix d'un alphabet Σ afin de coder des informations sous forme de chaînes de symboles,
- un codage de l'entrée x comme un mot de Σ^* ,
- un modèle abstrait de machine capable de traiter des chaînes de symboles et de produire une sortie.

EXEMPLE 13: Si on prend un ADF comme modèle abstrait de machine, alors il permettra de résoudre tous les problèmes devant déterminer si une chaîne de symboles appartient à un langage régulier particulier.

5.1 Problèmes

Nous allons en considérer deux types de problèmes résolubles par un traitement informatique :

- les problèmes de décision (ceux dont la réponse est oui ou non).
- les problèmes d'évaluation (ceux dont le but est de calculer une réponse).

5.1.1 Problèmes de décision

Un problème de décision est un problème dont la réponse ne peut être que oui ou non.

EXEMPLES 14: de problèmes de décision

- Décider si une chaîne binaire contient un nombre pair de 0.
- Décider pour un couple (x, y) si x divise y .
- Décider si un nombre entier x est premier.
- Décider si une liste d'entiers est triée.
- Décider si une équation à coefficients entiers a une solution entière.
- ...

Soit un alphabet Σ quelconque.

Définition I.17 (Problème de décision)

Un problème de décision est une fonction $f : \Sigma^* \rightarrow \{0, 1\}$ telle que, pour toute entrée $w \in \Sigma^*$, $f(w) = 1$ si la décision est oui pour w et $f(w) = 0$ si la décision est non.

REMARQUE 7:

Un problème de décision permet de définir un langage.

Reprenons l'exemple de test de parité, avec $\Sigma = \{0, \dots, 9\}$ et écrivons le problème de décision comme la fonction $f : \Sigma^* \rightarrow \{0, 1\}$ telle que $f(w) = 1$ si et seulement si w est un nombre premier.

Soit le langage $\text{PRIME} = \{w \in \Sigma^* \text{ tels que } w \text{ est nombre premier}\}$.

Alors il est défini de manière équivalente comme :

$\text{PRIME} = \{w \in \Sigma^* \text{ tels que } f(w) = 1\}$

En conséquence, tout problème de décision définit un langage.

REMARQUE 8:

La définition n'exclue pas que la fonction f puisse boucler, donc puisse ne jamais prendre de décision.

5.1.2 Problèmes d'évaluation

Un problème d'évaluation est un problème pour lequel on cherche à évaluer au moins une solution.

Pour cela, on note :

- soit x un problème.
- soit $R(x)$ l'ensemble des solutions au problème x .
- soit R l'ensemble des couples tels que x est un problème et $y \in R(x)$.

Définition I.18 (Problème d'évaluation)

Soit un problème d'évaluation $R \subseteq \Sigma^* \times \Sigma^*$.

Une fonction $f : \Sigma^* \rightarrow \Sigma^*$ résout un problème d'évaluation R si :

1. si l'ensemble des solutions au problème x n'est pas vide, alors $f(x)$ retourne l'une des solutions (i.e. si $R(x) \neq \emptyset$, $f(x) \in R(x)$).
2. s'il n'y a pas de solution, alors $f(x) = \perp$ (sortie spécifique pour signaler le problème)

EXEMPLES 15: de problème d'évaluation

- Évaluer le zéro d'une fonction.
si $x = \langle X \mapsto X^2 - 1 \rangle$, alors $R(x) = \{-1, +1\}$ et $f(x) = 1$ ou $f(x) = -1$.
- Trouver la décomposition en facteur premier d'un entier.
si $x = 18$, alors $R(x) = \{2 \cdot 3^2\}$ et $f(x) = 2 \cdot 3^2$
si $x = 0$, alors $R(x) = \emptyset$ et $f(x) = \perp$
- Trouver un diviseur strict $d > 1$ d'un entier.
si $x = 18$, alors $R(x) = \{2, 6, 9\}$ et $f(x) = 2, 3$ ou 9 .
- Trouver un chemin entre deux sommets d'un graphe.
si $x = \langle G, a, b \rangle$ alors $R(X) = \{\langle p_1 \rangle, \langle p_2 \rangle\}$ où on suppose que p_1 et p_2 sont deux listes de sommets décrivant un chemin dans G , commençant par a et terminant par b . Donc $f(x) = \langle p_1 \rangle$ ou $\langle p_2 \rangle$.
- Trouver le chemin le plus court entre deux sommets d'un graphe.
si $x = \langle G, a, b \rangle$ alors $R(X) = \{\langle p \rangle\}$ où p est le chemin le plus court dans G parmi les deux précédents. D'où $f(x) = \langle p \rangle$.
- ...

Il y a une fonction f particulière par problème d'évaluation à résoudre. Noter que x et $f(x)$ sont toujours codés dans Σ^* .

5.1.3 Considérations

Les deux modèles abstraits principaux de calcul que nous allons étudier sont :

- **les machines de Turing**
ce modèle est fait pour reconnaître les mots d'un langage.
il est adapté aux problèmes de décision.
- **les fonctions calculables**
ce modèle identiques aux machines de Turing mais capable de produire une sortie.
il est adapté aux problèmes d'évaluation.

On se restreindra dans un premier temps à des problèmes décisionnels :

- plus simple dans un premier temps
- ne pose pas de problèmes fondamentaux.

Noter que pour les problèmes d'évaluation, dire si un problème a oui ou non une solution implique souvent de trouver cette solution (par exemple, tel entier peut-il être factorisé = trouver cette factorisation si elle existe, et décider selon).

6 Hypothèse de Church-Turing

6.1 Algorithmes

Dans ce contexte plus général, la fonctionnement d'une MT devient le suivant :

- soit P un objet mathématique (resp. une structure de données) sur lequel on se pose une question mathématique (resp. algorithmique) à laquelle il est possible de répondre par oui ou par non,
- soit M une MT qui prend en entrée $\langle P \rangle$

Alors, le code de la MT M se structure de la manière suivante :

- analyse de la syntaxe de $\langle P \rangle$,
- vérification que $\langle P \rangle$ est un codage cohérent de l'objet représenté,
- application sur P de l'algorithme permettant de résoudre cette question (s'il en existe un exécutable sur une MT)

Ce qui amène aux questions :

- qu'est-ce-qu'un algorithme ?
- est-ce-que tout algorithme peut-être exécuté par une MT ?
- est-ce-que tout problème algorithmique peut être résolu par un algorithme ?

Qu'est-ce qu'un algorithme ?

- ☐ une recette ?
- ☐ une procédure ?
- ☐ un programme sur un ordinateur ?
- ☐ quelle importance ? Je le sais quand j'en vois un !

Historiquement :

- la notion intéresse les mathématiciens depuis longtemps
L'algorithme d'Euclide (300 av.J.-C.) pour le calcul du PGCD
- pas précisément définie avant le 20^{ème} siècle.

La notion était informelle, mais suffisante jusque là.

Avec l'étude mathématique des algorithmes, il devient maintenant nécessaire de définir rigoureusement la question.

6.2 Thèse de Church-Turing

D'où la **thèse de Church-Turing** :

"La notion intuitive d'un modèle raisonnable de calcul informatique est équivalente à un algorithme s'exécutant sur une machine de Turing."

Donc,

- tout programme que s'exécute sur une machine de Turing est algorithme.
- tout algorithme peut être exécuté sur une machine de Turing.

D'après cette hypothèse, tout algorithme exécuté sur n'importe quelle machine de calcul physiquement réalisable peut aussi être exécuté sur une machine de Turing.

y compris pour un ordinateur quantique (et une machine de Turing quantique)

Attention, ceci n'est qu'une hypothèse.

mais elle n'a pas été démentie jusqu'à présent.

Cette thèse restera probablement valide jusqu'à ce que l'on conçoive une machine physique qui ne puisse pas être simulée sur une MT.

Il existe aussi une forme forte de l'hypothèse de Church-Turing :

La notion intuitive d'un modèle raisonnable de calcul informatique est équivalente à un algorithme s'exécutant sur une machine de Turing **avec une pénalité au plus d'ordre polynomial**.

Il n'est pas encore vraiment clair que cette thèse ne soit pas vraie :

- les MTNDs n'ont actuellement pas d'implémentation physique,
- les ordinateurs quantiques sont de bons candidats pour invalider cette thèse,
 - Un modèle de machine de Turing quantique (MTQ) a été créé afin d'étudier les propriétés des ordinateurs quantiques tels qu'on les conçoit.
La thèse (faible) de Church-Turing reste vérifiée (= les MTQs ne reconnaissent pas d'autres langages que les MTs).
 - Aucune méthode n'a été trouvée pour simuler efficacement une MTQ sur une MT (= pénalité exponentielle).
 - Il n'est pas clair que les implémentations physiques des ordinateurs quantiques actuels utilisent bien les propriétés quantiques (la superposition d'état) qui rendent les ordinateurs quantiques intéressants.

7 Complétude de Turing

La MT est donc l'outil qui permet de simuler :

- tout modèle de machine physiquement réalisable,
- tout algorithme qu'il est possible d'écrire

En fait : Tous les modèles de langages de programmation "raisonnables" sont équivalents.

Java, C++, Lisp, Scheme, Prolog, Mathematica, Maple, Cobol, ...

Chacun de ces langages représente un formalisme en mesure de représenter plus facilement certains types d'algorithmes.

Définition I.19 (Complétude de Turing)

Le formalisme d'une machine est **Turing-complet** s'il permet de simuler une MT.

Donc, **Turing-complet** = peut exécuter tout algorithme.

8 Résumé

Nous avons vu dans ce cours que :

- une machine de Turing est modèle théorique qui permet de représenter l'exécution d'un algorithme sur un ordinateur,
- une fonction calculable est une machine de Turing qui s'arrête avec le résultat de la fonction écrit sur la bande.
- d'après la thèse de Church-Turing, un algorithme s'exécutant sur une machine de Turing est équivalent à un modèle raisonnable de calcul informatique,
- tout modèle raisonnable de calcul peut être simulé sur une machine de Turing,
- cette simulation peut être effectuée avec un facteur multiplicatif de pénalité au plus quadratique, exception faire des MTs quantiques.
- un modèle de machine est Turing-complet s'il permet de simuler tout algorithme (=simuler une machine de Turing),
- le modèle de la machine de Turing a servi de base pour concevoir le principe des ordinateurs modernes.

