Machine Turing

Septuplet $\mathcal{M} = (Q, X, \Gamma, \delta, q_0, F, \#)$ où :

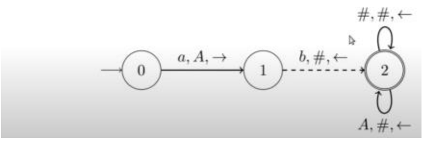
- Q : ensemble fini d'états
- X : alphabet (fini)
- Γ : alphabet de bande, tel que $X \subset \Gamma$, et $\# \in \Gamma \setminus X$ (le blanc)
- $q_0 \in Q$: l'état initial de l'automate
- $F \subseteq Q$: les états finals (ou terminaux)
- $\delta \in Q \times \Gamma \mapsto Q \times \Gamma \times \{\leftarrow, -, \rightarrow\}$: fonction de transition.

Une machine de Turing possède une structure de stockage qui est un ruban linéaire *non borné*.

Exemple

$(Q, \Sigma, \Gamma, \#, \delta, q_0, F)$

- $Q = \{0, 1, 2\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{A\}$
- $\# = \#$
- $F = \{2\}$
- $\delta(0, a) = (1, A, \rightarrow)$
- $\delta(1, b) = (2, \#, \leftarrow)$

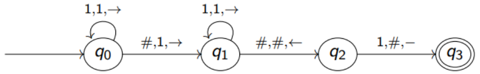


Etats possibles d'une machine de Turing

Quand une machine n'a pas de transition possible dans une configuration donnée, elle s'arrête. Pour une entrée donnée, une machine a donc trois comportements possibles : s'arrêter sur un état final, s'arrêter sur un état non final, boucler indéfiniment.

Tableau de transitions

	$\#$	1	
q_0	$q_1, 1, \rightarrow$	$q_0, 1, \rightarrow$	parcourt 1^n et met 1
q_1	$q_3, \#, \leftarrow$	$q_1, 1, \rightarrow$	parcourt 1^m
q_2		$q_3, \#, -$	enlève le 1 de trop
q_3	$\varnothing$		



Machine de Turing universelle

Une machine de Turing universelle est une machine de Turing spéciale capable de simuler le fonctionnement de n'importe quelle autre machine de Turing.

Indécidabilité

Décidabilité (algorithmique)

Un problème de décision est décidable s'il existe un algorithme (= une machine de Turing) qui termine en temps fini et répond oui / non selon si l'entrée est vraie (= est une instance positive).

Semi-décidabilité

Un problème de décision est semi-décidable s'il existe un algorithme (= une machine de Turing) qui, si l'entrée est vraie, termine en temps fini et répond oui.

Équations diophantiennes (dixième problème de Hilbert)

Soit $p(x_1, \ldots, x_n)$ un polynôme à coefficients entiers. Déterminer si l'équation $p(x_1, \ldots, x_n) = 0$ possède des solutions entières est un problème indécidable. (théorème de Matiyasevich, 1970)

Indécidabilité du rejet

Savoir si une machine n'accepte pas un mot donné est un problème ni décidable, ni semi-décidable.

Indécidabilité de la minimalité

Savoir si une machine de Turing est la plus petite qui résout un problème est indécidable.

Indécidabilité du test à zéro

Savoir si une machine n'accepte aucun mot est indécidable.

Indécidabilité de l'équivalence

Savoir si deux machines de Turing sont équivalentes (i.e. acceptent le même langage et calculent la même fonction) est indécidable.

Arithmétique

La validité d'une formule arithmétique (avec $+$ et $*$) est indécidable.

Algèbre linéaire

Étant donné un nombre fini de matrices 3×3 à coefficients entiers, déterminer si un produit multiple permet d'annuler la composante (i, j) est indécidable.

Décidabilité des langages rationnels

Globalement, tous les problèmes concernant les langages rationnels et les automates à états finis sont décidables :

- $m \in L(A)$
- $L(A) = \emptyset$
- $L(A) = L(B)$
- $L(A) \subseteq L(B)$
- ...

Décidabilité des grammaires algébriques

Soit une grammaire algébrique G, les problèmes suivants sont décidables :

- $m \in L(G)$
- $L(G) = \emptyset$

Preuve : analyseur d'Earley ou LR généralisé

Indécidabilité des grammaires algébriques

Soit deux grammaires algébriques G et G' sur un alphabet Σ, les problèmes suivants sont indécidables :

1. $L(G) \cap L(G') = \emptyset$
2. $L(G) = \Sigma^*$
3. $L(G) = L(G')$
4. $L(G) \subseteq L(G')$

Preuve par réduction de PCP.

Busy Beaver

Corollaire : Σ n'est pas calculable.

S n'est pas calculable.

Le nombre de transitions $S(n)$ est supérieur au nombre de 1 $\Sigma(n)$.

Il n'existe pas de fonction calculable qui donne une borne supérieure à $S(n)$ (un majorant).

Fonction récursive primitive

Toute fonction construite à partir des fonctions de base suivante :

Soit la classe des fonctions de $\mathbb{N}^k$ vers $\mathbb{N}^r$ construites à partir de :

- Identité $id : \mathbb{N}^k \to \mathbb{N}^k$ telle que $id(x_1, \ldots, x_n) = (x_1, \ldots, x_n)$
- Zéro $Z : \mathbb{N}^0 \to \mathbb{N}$ telle que $Z() = 0$
- Successeur $S : \mathbb{N} \to \mathbb{N}$ telle que $S(n) = n + 1$
- Projection $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$ telle que $\pi_i^k(x_1, \ldots, x_k) = x_i$
- Composition $Comp$ telle que $Comp(f, g_1, \ldots, g_n) = h$ où $h(x_1, \ldots, x_n) = f(g(x_1), \ldots, g(x_n))$
- Récursion Rec telle que $Rec(f, g) = u$ où
  $\begin{cases} u(m, 0) = f(m) \\ u(m, n+1) = g(n, u(m, n), m) \end{cases}$
  (La récursion termine nécessairement par décroissance à 0)

Ils sont calculables par une machine de Turing et donc décidable,

Complexité

Complexité en temps

- **P** = classe des problèmes solubles en temps polynomial de la taille de l'entrée
- **NP** = classe des problèmes vérifiables en temps polynomial
- **NP**-complet = problèmes vérifiables en temps polynomial, mais pas solubles en temps polynomial (sans doute)
- **EXPTIME** = problèmes solubles en temps exponentiel
- **BQP** = problèmes solubles en temps polynomial par un ordinateur quantique

Complexité en espace

- **LSPACE** = problèmes nécessitant moins que $\log n$ espace
- **PSPACE** = problèmes nécessitant un espace polynomial
- **EXPSPACE** = problèmes nécessitant un espace exponentiel

Exemple de problème en P

- Vérification de multiplication de matrices
- Test de primalité (2002)
  (naïf en $O(2^{\log n})$, algorithme probabiliste polynomial 1976)
- Existence d'un chemin entre deux nœuds d'un graphe
- Test de planarité d'un graphe
- Graphe eulérien : existence d'un cycle passant par chaque arc exactement une fois
- Programmation linéaire (1984)
- Factorisation de polynôme dans $\mathbb{Q}$ (1982)
- Un Rubik's cube arbitrairement coloré est-il soluble ? (2015)
- Évaluation d'un circuit logique à partir des entrées

Exemple de problème en EXPTIME

- Soit une machine de Turing M, $M(\epsilon)$ s'arrête-t-elle en $\leq k$ étapes ? La seule manière est d'exécuter M pendant k transitions. L'entrée k, codée en base 2, prend $\lceil \log_2 k \rceil$ bits, l'algorithme prend $O(2^{\log k})$ pas.
- Échecs, Go (généralisés) : une partie peut nécessiter un nombre de pas exponentiel de la taille du damier.
- Vérification d'une formule LTL : exponentiel en le nombre d'opérateurs temporels.

Exemple de problème en NP

En théorie de la complexité, la classe de complexité **NP** (non déterministe, temps polynomial) regroupe l'ensemble de tous les problèmes de décision pour lesquels une solution peut être vérifiée en un temps polynomial, par contre le calcul de la solution peut prendre un temps infiniment long.

- Satisfiabilité d'une formule propositionnelle
  certificat : valuation des symboles
- k-colorabilité d'un graphe
  certificat : l'affectation des couleurs
- Problème du sac à dos
  certificat : les objets retenus
- Factorisation : x a-t-il un diviseur premier dans l'intervalle $[a, b]$?
  certificat : le diviseur
- Programmation linéaire
  certificat : valuation des variables
- Existence d'un cycle hamiltonien (qui passe exactement une fois par tous les nœuds)
  certificat : un tel cycle

Hiérarchie

Hiérarchie

- **P** $\subseteq$ **NP** $\subseteq$ **EXPTIME** $\subseteq$ **NEXPTIME**
- **P** $\subsetneq$ **EXPTIME** **NP** $\subsetneq$ **NEXPTIME**
- **P** $\stackrel{?}{=}$ **NP**
- **NP** $\stackrel{?}{=}$ **EXPTIME**
- **EXPTIME** $\stackrel{?}{=}$ **NEXPTIME**

Complexité spatiale

Classe 1SPACE

Classe 1SPACE

1SPACE = problèmes décidables en espace constant :

$$\mathbf{1SPACE} \triangleq \mathrm{DSPACE}(1)$$

Exemples :

- Parité d'un entier
- Incrémenter un entier
- Nombre pair de 1 dans un mot de $\{0,1\}^*$

Classe LSPACE

Classe LSPACE

LSPACE = problèmes décidables en espace logarithmique :

$$\mathbf{LSPACE} \triangleq \mathrm{DSPACE}(\log(n))$$

- Compter jusqu'à n
- Calculer le n-ième nombre de Fibonacci
- Additionner, multiplier deux entiers
- Comparer deux entiers, trier un ensemble d'entier
- Connexité d'un graphe non orienté (2004)

Classe NLSPACE

Classe NLSPACE

NLSPACE = problèmes décidables non déterministiquement en espace logarithmique :

$$\mathbf{NLSPACE} \triangleq \mathrm{NSPACE}(\log(n))$$

Classe PSPACE et EXPSPACE

Classes PSPACE, EXPSPACE

PSPACE = problèmes décidables en espace polynomial :

$$\mathbf{PSPACE} \triangleq \bigcup_{c\in\mathbb{N}} \mathrm{DSPACE}(n^c)$$

EXPSPACE = problèmes décidables en espace exponentiel :

$$\mathbf{EXPSPACE} \triangleq \bigcup_{c\in\mathbb{N}} \mathrm{DSPACE}(2^{n^c})$$

Problème QBF (ou QSAT)

QBF = ensemble des formules booléennes quantifiées closes qui sont vraies

(closes = sans variable libre)
exemple : $\forall x_1, \exists x_2, x_1 \vee x_2$

QBF est **PSPACE**-complet.

Bilan

$$\mathbf{1SPACE} \subseteq \mathbf{LSPACE} \subseteq \mathbf{NLSPACE} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{EXPSPACE}$$

Calcul probabiliste

Tri rapide probabiliste

Tri rapide déterministe

$tridet(x) \triangleq$ si $|x| = 1$ alors retourner x
sinon $T_1 \leftarrow \{x_i \mid x_i < x_1\}$, $T_2 \leftarrow \{x_i \mid x_i > x_1\}$
retourner $[tridet(T_1), x_1, tridet(T_2)]$

Complexité pire cas $= O(n^2)$ (si x est déjà trié)

Tri rapide probabiliste

$triproba(x) \triangleq$ si $|x| = 1$ alors retourner x
sinon $p \leftarrow$ au hasard parmi $1, ..|x|$
$T_1 \leftarrow \{x_i \mid x_i < x_p\}$, $T_2 \leftarrow \{x_i \mid x_i > x_p\}$
retourner $[triproba(T_1), x_p, triproba(T_2)]$

- Temps moyen d'exécution $= O(n \log n)$ sur toute entrée
- Encore mieux : tirer trois pivots et utiliser le x_p médian

Sortir d'un labyrinthe

Problème st-UCONN

Soit un graphe non orienté G et deux nœuds s et t, trouver un chemin de s à t.

- Parcours en profondeur de G : temps et espace polynomiaux
- Parcours aléatoire : choisir aléatoirement la prochaine arête, sans mémoire : temps polynomial, espace logarithmique (numéro du nœud courant). En $O(n^3)$ pas, il y a une forte probabilité d'avoir parcouru toutes les arêtes (1979).
- En fait, il existe un algorithme déterministe polynomial en espace logarithmique (2005).

Si le graphe est orienté, un parcours aléatoire prend un temps exponentiel

La classe BPP

Classe BPP

Un langage $L \subseteq \{0,1\}^*$ appartient à la classe de complexité **BPP** $\triangleq \exists M$ une machine de Turing polynomiale, $\exists p$ un polynôme :

- $\forall x \in L : Pr_{y\in\{0,1\}^{p(|x|)}}[M(\langle x,y\rangle) \text{ accepte}] \geq 2/3$.
- $\forall x \notin L : Pr_{y\in\{0,1\}^{p(|x|)}}[M(\langle x,y\rangle) \text{ rejette}] \geq 2/3$.

Est-ce que la fonction de Collatz termine pour n>0 ?

la réponse correcte est : "on ne sait pas". Bien que la conjecture de Collatz suggère que la séquence de Collatz se termine pour toute valeur de n≥0, cette conjecture n'a pas encore été prouvée, et il existe des valeurs de *n* pour lesquelles la terminaison de la séquence n'est pas démontrée.

La question de savoir si une machine de Turing n'accepte qu'un unique mot est décidable.

Principe de l'algorithme :

1. Exécuter la machine de Turing sur tous les mots possibles de longueur croissante, en commençant par les mots de longueur 0, puis 1, puis 2, et ainsi de suite.
2. Pour chaque longueur de mot, vérifier si la machine de Turing accepte un mot. Si oui, sauvegarder ce mot comme étant accepté.
3. Si la machine de Turing accepte un mot à une longueur donnée, mais aucun mot de longueur plus courte n'a été accepté, alors arrêter et conclure que la machine accepte un unique mot.
4. Si la machine de Turing accepte un mot à plusieurs longueurs différentes, alors conclure qu'elle n'accepte pas qu'un unique mot.

- Ce principe repose sur le fait que les machines de Turing sont des modèles de calculs déterministes. Si une machine de Turing accepte plus d'un mot, alors elle acceptera un mot de longueur minimale avant d'accepter d'autres mots de longueur supérieure. Par conséquent, si aucun mot de longueur plus courte n'est accepté, alors la machine accepte un unique mot.

Pourquoi complexité + calculabilité ?

En bref, il est important d'étudier la complexité en plus de la calculabilité pour plusieurs raisons :

1. Évaluation des performances des algorithmes.
2. Comprendre les limites pratiques de la calculabilité.
3. Optimisation des programmes.
4. Conception de systèmes informatiques.

En combinant l'étude de la calculabilité et de la complexité, les informaticiens peuvent mieux comprendre les capacités et les limites des systèmes informatiques, et concevoir des solutions efficaces et optimales pour résoudre divers problèmes.

Complexité en temps et espace

Oui, c'est vrai. En général, la complexité en temps d'un algorithme ne dépend pas du codage des données, tandis que la complexité en espace peut en dépendre.

- **Complexité en temps** : Elle mesure le nombre d'opérations effectuées par un algorithme et reste constante quelle que soit la manière dont les données sont codées.
- **Complexité en espace** : Elle mesure la quantité de mémoire utilisée par un algorithme. Elle peut varier en fonction du codage des données. Par exemple, si les données sont compressées, l'algorithme peut utiliser moins de mémoire que s'il traite des données non compressées

Un problème complet pour PSPACE est le problème du jeu de la vie quantique (Quantum Game of Life). Dans ce jeu, on est donné une configuration initiale de cellules quantiques sur une grille bidimensionnelle. Les cellules peuvent être dans un état superposé, ce qui signifie qu'elles peuvent représenter à la fois l'état de vie et de mort en même temps.

À chaque étape, l'évolution du jeu est déterminée par des règles quantiques spécifiques qui prennent en compte les interactions entre les cellules et leur superposition quantique. Le problème consiste à décider si une certaine configuration initiale mène à un état final où toutes les cellules sont dans un état de vie ou si au moins une cellule est dans un état de mort, en suivant les règles du jeu.

Formellement, le problème du jeu de la vie quantique peut être décrit comme suit : Entrée : Une configuration initiale de cellules quantiques sur une grille bidimensionnelle et un nombre d'étapes *k*. Question : Est-ce que toutes les cellules sont vivantes après *k* étapes du jeu, en suivant les règles du jeu de la vie quantique ?