Sciences orientation logicielle 2

Mathématiques Discrètes

Support du cours

Enseignant : Niklaus Eggenberg

Email : niklaus.eggenberg@hesge.ch

Support du cours : www.eswys.ch/hepia

Plan du cours

- Ch. 1 Automates et analyse de complexité
- Ch. 2 Théorie des graphes
- Ch. 3 ..

Evaluation: Moyenne arith. 3 travaux pratiques notés

Chapitre 1

Automates & Analyse de complexité

Définitions: Problème et Instance

- Un ***problème*** est une question générique
  - *Déterminer si un nombre n entier est pair ou impair.*
  - *Quel est le PDCD de deux entiers m et n?*

Le problème est indépendant de toute notion de solution, solvabilité et peut avoir plusieurs solutions.

- Une ***instance*** d'un problème est un cas particulier d'un problème
  - *Déterminer si* 10204091 *entier est pair ou impair.*
  - *Quel est le PDCD de deux entiers* 106 *et* 2076*?*

Algorithme

Un ***algorithme*** est une méthode effective de calcul ou un processus de résolution d'un problème par le calcul.

- Premières apparition en 2000 avant J.-C. à Babylone (Irak) pour le calcul des impôts.
- Premier algorithme «moderne» par Euclide (Grèce, 300 avant J.-C.): le calcul du PGCD.
- IX^{e} siècle après J.-C., Al Khuwarzimi, un mathématicien perse (Iran) publie un ouvrage dédié à ces «méthodes». Le mot «algorithme» est dérivé de son nom.
- Aux XVII^{e}, Pascal construit la pascaline, considéré comme l'une des première machine à calculer automatisée.

Décidabilité d'un problème

En 1928, Hilbert soulève la question de décidabilité (***Entscheidungsproblem***)

Etant donné un énoncé mathématique, existe-t-il un algorithme capable de décider si l'énoncé est vrai ?

Plusieurs formalismes (encodages) sont proposés dès 1930 pour tenter de démontrer le résultat, dont la ***machine de Turing*** grâce à la quelle Turing montre qu'il n'existe aucun algorithme pour l'*Entscheidungsproblem*.

Décidabilité d'un problème

- Un problème est dit ***décidable*** s'il existe un algorithme permettant de trouver une solution au problème quelle que soit son instance.

- A l'inverse, un problème est ***indécidable*** s'il n'existe aucun algorithme pour le résoudre.

Note:
Un problème de décision est une problème dont la réponse est soit «Oui» soit «Non». C'est une sous-classe de problèmes.

Exemples: Problème et Instance

- Exemples de problèmes décidables
  - Trouver le PGCD de deux nombres entiers m et n.
  - Déterminer si un nombre entier n est un nombre premier ou non.

- Exemples de problèmes indécidables
  - Soit un polynôme $p(x) = \sum_{i=1}^{n} \alpha_i x^i$, avec $\alpha_i \in \mathbb{Z}$, existe-t-il une valeur entière $x^* \in \mathbb{Z}$ tel que
$$p(x^*) = 0.$$
  - Soient $F_1, \ldots, F_n$ des formes polygonales, peut-on paver paver le plan, sans recouvrement ni espace vide, avec des exemplaires de $F_1, \ldots, F_n$

Thèse de Church-Turing

Pour tout problème décidable, il existe un machine de Turing pour le résoudre.

Autrement dit

Faisable par un algorithme = Faisable par une machine de Turing

Note:
Il existe d'autres formalismes équivalentes à la machine de Turing (p.ex. λ-calcul, machine à compteurs, foncions récursives, fonctions à pile, automates cellulaires).

Définitions

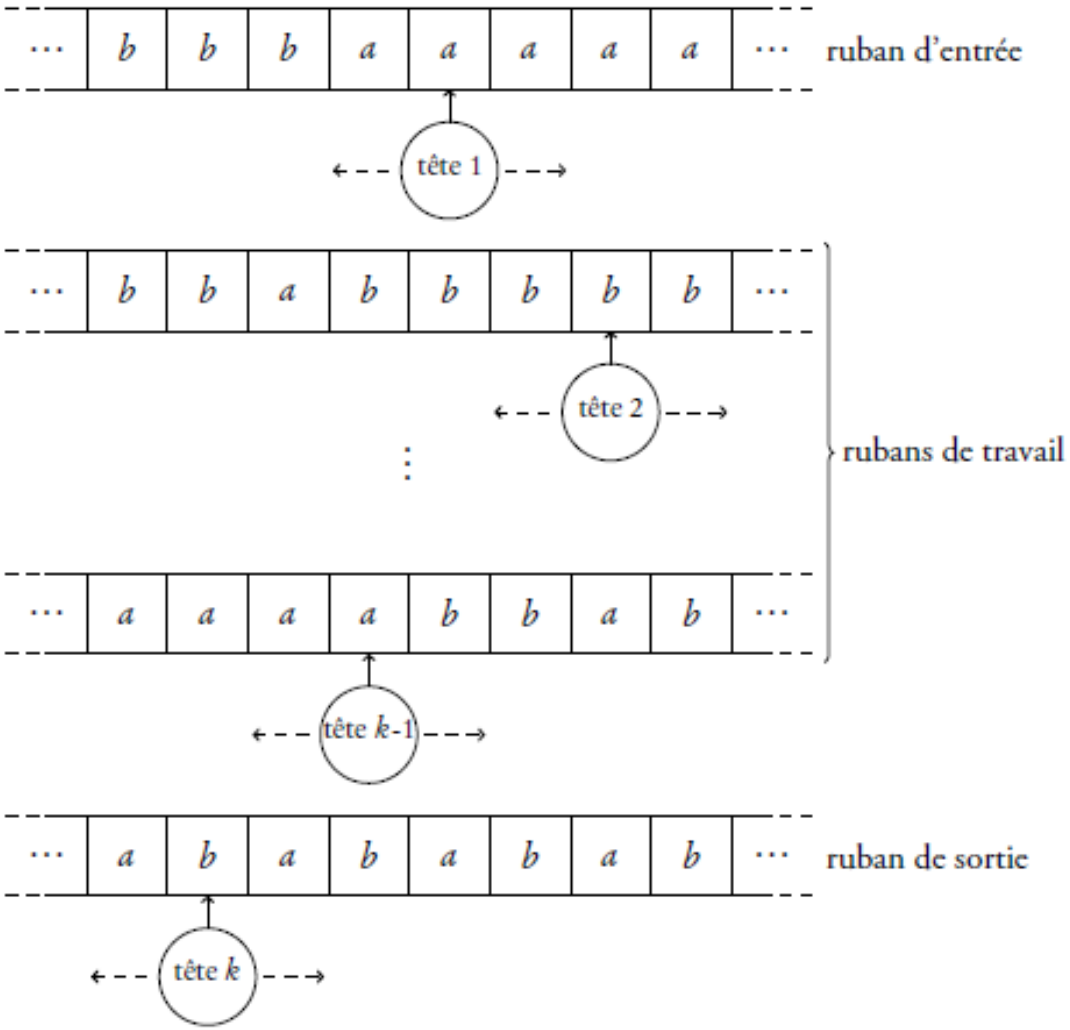
- Un ***alphabet*** est ensemble fini de ***symboles,***
- Le ***symbole vide***, représenté par ϵ,
- Un ***mot*** est une suite de symboles (ou ***concaténation***),
- La ***longueur*** d'un mot correspond au nombre de symboles dont le mot est composé,
- Un ***langage*** est un ensemble de mots,
- Un ***langage fini*** est un ensemble de mots de longueur finie,
- Une ***fonction d'encodage*** permet de traduire un problème dans un langage (fini ou non).

La machine de Turing

La machine de Turing est composée des éléments suivants :

- ***Ruban*** composé de cellules contenant des symboles,
- ***Tête de lecture*** et/ou écriture qui se trouve dans un certain état (position),
- Un **état** correspond au symbole dans la position du ruban,
- Il existe 2 états spéciaux correspondant à l'état initial et à l'état final,
- Une ou plusieurs ***fonctions de transition*** définissant les règles de transition entre les divers états.

La machine de Turing



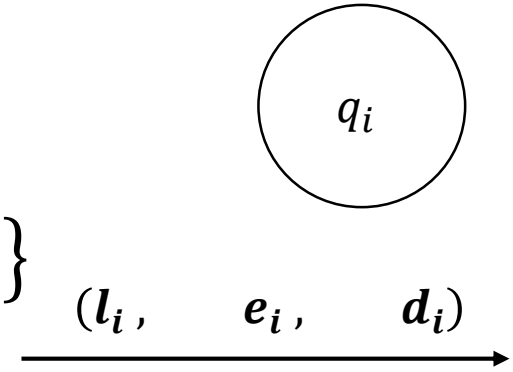
Exemple – Inverser un octet

- Alphabet$= \{null, 0, 1\}$,
- Mots $= \{(b_1, \dots, b_8), | \, b_i \in \{0,1\} \, \forall \, i = 1, \dots, 8\}$,
- **Etat** 1
  La tête se déplace à droite tant qu'elle lit $null$, et passe à l'Etat 2 dès qu'elle lit 0 ou 1,
- **Etat** 2
  Si la tête lit 0 ou 1, écrire l'inverse (1 ou 0) et se déplacer à droite
  Si la tête lit $null$, passage à l'état final (STOP)

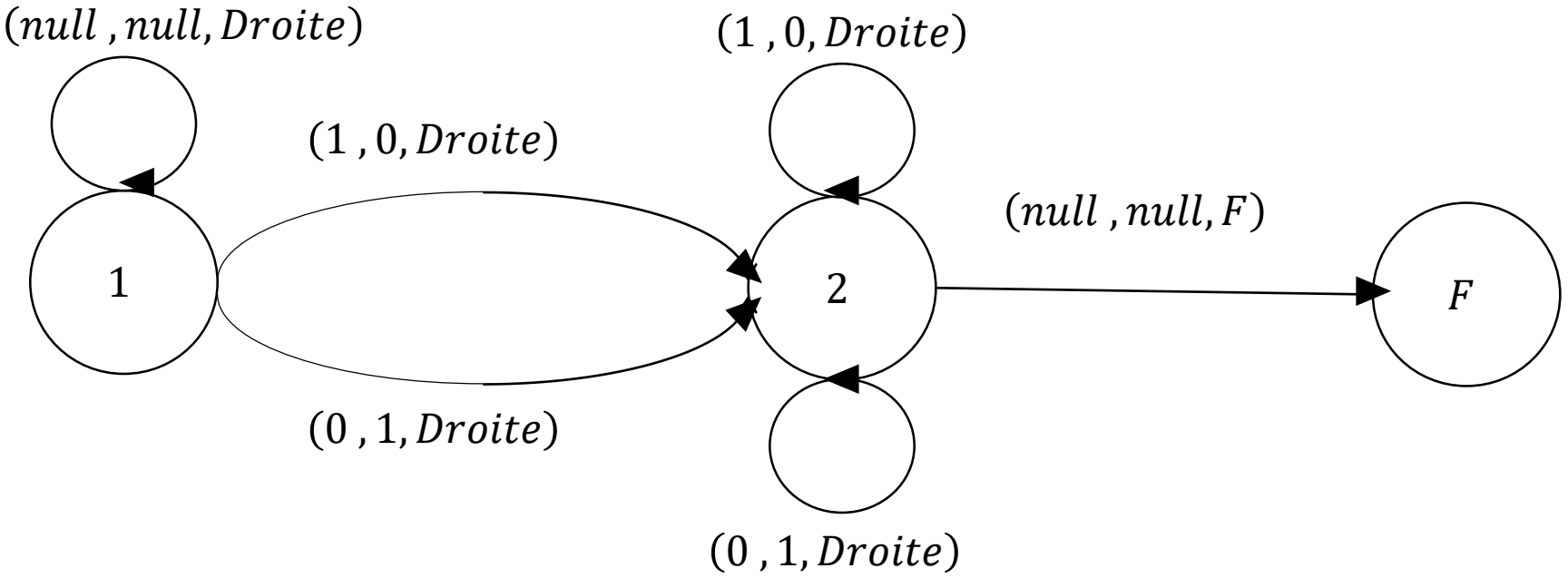
Formalisation

- Un état de la machine est défini par le triplet
  $q = (lect.,\ écr.,\ dir.)$

- La machine de Turing est définie par un ensemble d'états
  $$\mathrm{Q} = \{q_i \mid i = 0, \dots, F\}$$
  et de transitions
  $$\mathrm{T} = \{(q_i, q_j) \mid i, j = 0, \dots, F\}$$



Exemple – inversion du bit suite



Exemple à 3 bandes

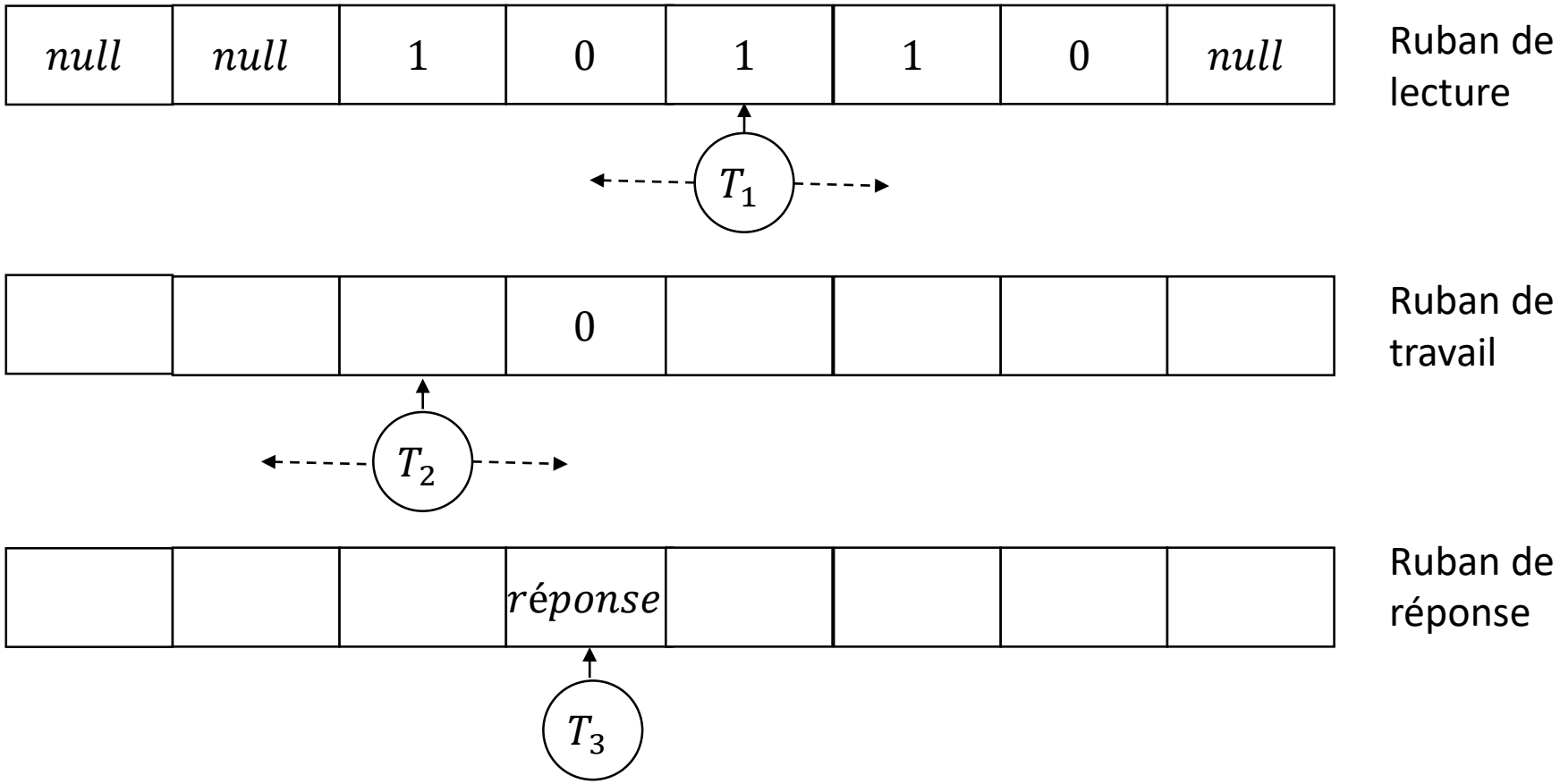
Objectif: tester si le mot en entrée a le même nombre de 0 que de 1.

Procédé:

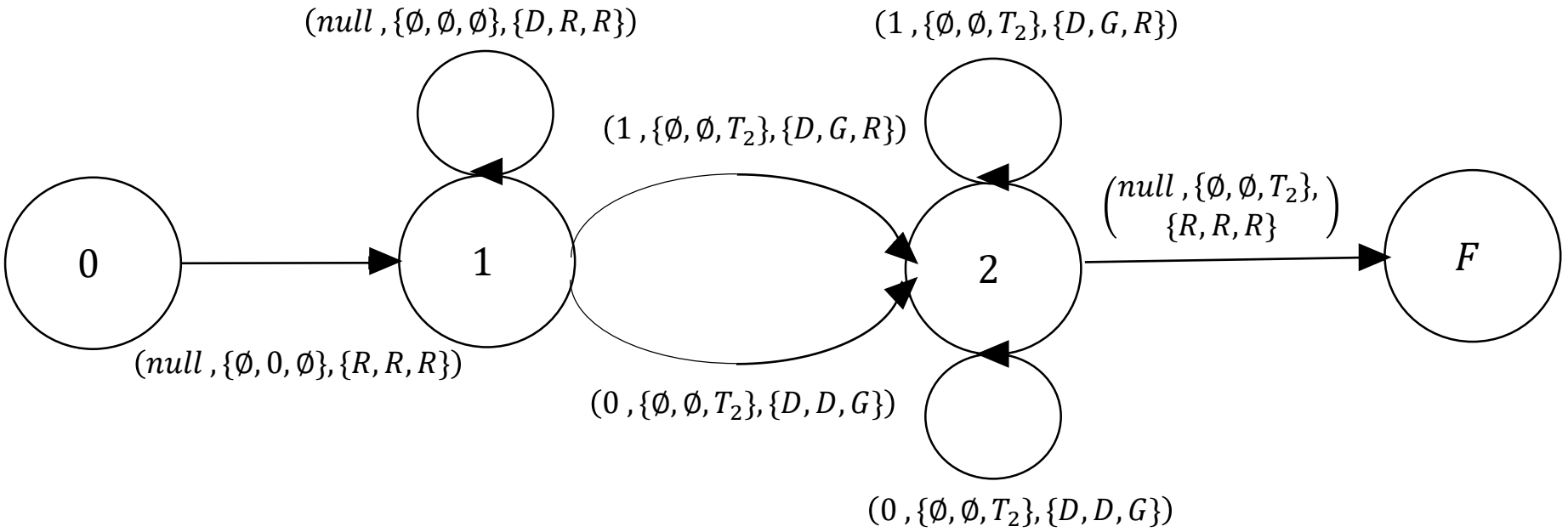
- Ecrire 0 sur le ruban d'écriture à la position 0,
- Lire le mot sur la bande de lecture de gauche à droite
- Si le symbole lu est 0, déplacer le ruban d'écriture à droite,
- Si le symbole lu est 1, déplacer le ruban d'écriture à gauche,
- Ecrire sur le ruban résultat l'état du ruban 2 (1 => «OUI», le nombre est égale, $null$ sinon).

Exemple à 3 bandes

Objectif: tester si le mot en entrée a le même nombre de 0 que de 1.



Exemple – Même nombre de 0 et 1



- R ***R***ester sur place
- D Déplacement à ***D***roite
- G Déplacement à ***G***auche
- $\emptyset$ Ne rien écrire

Exercice:

Construisez une machine de Turing permettant de lire une séquence de bits $\{0, 1\}$ et doublant le nombre de 1.

Exemple : (010100110) devient (011011001110).

Indice: utilisez 2 bandes !

Erratum: Problème et Instance

Le dixième problème de Hilbert suivant

Soit un polynôme $p(x) = \sum_{i=1}^{n} \alpha_i x^i$, avec $\alpha_i \in \mathbb{Z}$, existe-t-il une valeur entière $x^* \in \mathbb{Z}$ tel que

$$p(x^*) = 0.$$

N'est <u>PAS</u> décidable!

Erratum: Problème et Instance

Le dixième problème de Hilbert suivant

Soit un polynôme $p(x) = \sum_{i=1}^{n} \alpha_i x^i$, avec $\alpha_i \in \mathbb{Z}$, existe-t-il une valeur entière $x^* \in \mathbb{Z}$ tel que

$$p(x^*) = 0.$$

N'est PAS décidable!

En revanche, si on fixe $n < 3$, alors le problème est décidable.

Propriétés des machines de Turing

- Le ***résultat*** d'un mot correspond à tous les éléments non-vides du ruban résultat,
- Un mot est dit ***accepté*** par une machine de Turing si ladite machine se termine sur l'état final (la machine ne boucle pas et ne rejette pas le mot),
- Un langage est dit ***accepté*** par une machine de Turing si l'ensemble des mots du langage sont acceptés par ladite machine,
- Le ***temps de calcul*** d'un mot est le nombre d'étapes effectuées entre l'étape initiale et l'étape finale,
- ***L'espace*** utilisé est le nombre total de cases différentes visitées au cours du calcul par les *têtes du ruban de travail* (donc l'espace des rubans d'entrée et de sortie n'est pas compté).

Machines déterministes ou non

- Une machine est dite ***déterministe*** si, pour chaque état donné, l'action élémentaire effectuée est la même.
  - Toutes les machines encodées par une machine de Turing sont déterministes.

- Une machine ***non-déterministe*** est une machine qui peu effectuer un choix non-déterministe, ou aléatoire, de la transition.
  - Une machine non-déterministe ne peut pas être encodée par une machine de Turing standard.

Equivalences

- Une machine de Turing considère que les rubans sont infinis. Qu'ils soient infinis à gauche, à droite ou les deux est équivalent.

- Il existe un ***machine de Turing universelle*** permettant de simuler toutes les machines de Turing.
  - Cette machine a 5 rubans,
  - La description et la preuve nécessitent un formalisme qui dépasse le contenu de ce cours.

- Il est possible de construire un langage non-décidable à partir d'un langage décidable.

Preuve par la méthode diagonale

- La méthode de la preuve par le procédé diagonal fut introduite par Cantor en 1891 pour prouver que $\mathbb{R}$ n'est pas dénombrable.

- Idée par l'absurde : on suppose qu'un résultat soit vrai et, avec une liste infinie, on construit un élément qui ne fait pas partie de ladite liste.

Nous allons illustrer cette méthode avec la preuve que $\mathbb{R}$ n'est pas dénombrable.

$\mathbb{R}$ n'est pas dénombrable

- Si $[0,1)$ n'est pas dénombrable, $\mathbb{R}$ ne l'est pas non plus;
- Supposons $[0,1)$ dénombrable. Il existe donc une suite $(\mathrm{r}_1, \mathrm{r}_2, \dots)$ avec un nombre infini associant un $\mathrm{r_i}$ à tout élément dans $[0,1)$;
- $\mathrm{r_i}$ est composé d'une infinité de décimales (ou un nombre fini de décimales mais une infinité de 0);
- Ecrivons donc $\mathrm{r_i} = 0.\mathrm{r_{i1}}\ \mathrm{r_{i2}}\ r_{i3}\ \dots r_{in} \dots$ sous forme décimale;
- Construisons le réel suivant : $x = 0.\delta_{11}\ \delta_{22}\ \delta_{33} \dots \delta_{nn} \dots$ où

$$\delta_{ii} = \begin{cases} 1\ si\ r_{ii} \neq 1 \\ 2\ si\ r_{ii} = 1 \end{cases}$$

$\mathbb{R}$ n'est pas dénombrable

- Exercice : construisez $x \in [0,1)$ pour la suite suivante

$r_1 = 0, \mathbf{0}\ 1\ 0\ 5\ 1\ 1\ 0 \ldots$
$r_2 = 0, 4\ \mathbf{1}\ 3\ 2\ 0\ 4\ 3 \ldots$
$r_3 = 0, 8\ 2\ \mathbf{4}\ 5\ 0\ 2\ 6 \ldots$
$r_4 = 0, 2\ 3\ 3\ \mathbf{0}\ 1\ 2\ 6 \ldots$
$r_5 = 0, 4\ 1\ 0\ 7\ \mathbf{2}\ 4\ 6 \ldots$
$r_6 = 0, 9\ 9\ 3\ 7\ 8\ \mathbf{1}\ 8 \ldots$
$r_7 = 0, 0\ 1\ 0\ 5\ 1\ 3\ \mathbf{0} \ldots$

$\mathbb{R}$ n'est pas dénombrable

- Réponse

$$\delta_{11} = 2$$
$$\delta_{22} = 2$$
$$\delta_{33} = 1$$
$$\delta_{44} = 1$$
$$\delta_{55} = 1$$
$$\delta_{66} = 2$$
$$\delta_{77} = 1$$

$\mathbb{R}$ n'est pas dénombrable

- Posons $x = 0.\delta_{11}\delta_{22}\delta_{33}\ldots$
- Clairement, $x \in [0,1)$;
- Supposons $x \in \{r_i, i = 1, 2, 3, \ldots, n, \ldots\}$, il existe donc un élément i^* tel que $x = r_{i^*}$;
- Par définition de x, nous avons que la i^*-ème décimale de x est

$$r_{i^*i^*} = \begin{cases} 1 \; si \; r_{i^*i^*} \neq 1 \\ 2 \; si \; r_{i^*i^*} = 1 \end{cases}$$, ce qui conduit à une contradiction :

$$r_{i^*i^*} \neq r_{i^*i^*}.$$

- Nous avons donc construit un élément $x \notin \{r_i, i = 1, 2, 3, \ldots, n, \ldots\}$, qui est, par hypothèse, un ensemble dénombrable de $[0, 1)$.
- Nous concluons que l'hypothèse est fausse, et donc, que $[0, 1)$ et tous les espaces qui contiennent $[0, 1)$ (dont $\mathbb{R}$), ne sont pas dénombrables.

Langage non-décidable

- Soit $W = \{w_0, w_1, \dots, w_n, \dots\}$ l'ensemble infini de tous les mots finis (il en existe une infinité),
- Soit $M = \{ M_0, \dots, M_n, \dots\}$ l'ensemble infini de toutes les machines de Turing,
- Soit le tableau $A = \left\{ a_{ij} = \left[\begin{array}{c} 0 \; si \; M_i \; accepte \; w_j \\ 1 \; si \; M_i \; n'accepte \; pas \; w_j \end{array} \right\} \right.$,
- Alors le langage $L_0 = \{w \mid w = w_i \wedge A[i,i] == 1\}$ n'est pas décidable.

Note: $\wedge$ et l'opérateur binaire «*et*»

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

Preuve – par l'absurde

- Supposons que L_0 est accepté par une machine de Turing.
- Il existe alors une machine $M_{j^*} \in M = \{ M_0, \dots, M_n, \dots \}$ qui accepte L_0, car M est l'ensemble de ***toutes*** les machines de Turing.
- Si $w_{j^*} \in L_0$, cela implique, par définition de L_0, que $A[j^*, j^*] = 1$, donc que la machine M_{j^*} n'accepte pas w_{j^*}, ce qui contredit que L_0 est accepté par M_{j^*}.
- Si $w_{j^*} \notin L_0$, cela implique, par définition de L_0, que $A[j^*, j^*] = 0$, donc que la machine M_{j^*} accepte w_{j^*}, ce qui contredit que L_0 est accepté par M_{j^*}.
- Dans les deux cas, nous avons une contradiction, prouvant que l'hypothèse est fausse : donc il n'existe pas de machine de Turing acceptant L_0.

Langage non-décidable

- Le langage L_0 n'est pas accepté par une machine de Turing et est donc non-décidable.

- Question: existe-t-il des langages acceptés par une machine de Turing mais non-décidables ?

- Réponse – OUI:

$\bar{L}_0 = \{w \mid w = w_i \wedge A[i,i] == 0\}$ est un langage accepté par une machine de Turing, mais non-décidable.

Quelques problèmes indécidables

- Déterminer si une machine de Turing s'arrête lorsqu'elle est lancée sur le mot vide est indécidable.

- Déterminer si une machine de Turing s'arrête au moins sur un mot est indécidable (problème de l'arrêt existentiel).

- Déterminer si une machine de Turing s'arrête sur tous les mots d'un langage est indécidable (problème de l'arrêt universel).

<u>Conséquence:</u>
Il n'est pas possible de construire un programme de test permettant de tester si un programme existant se termine sur toutes les instances du problème.

Théorie de la complexité

- La ***complexité*** d'un problème s'exprime en nombre d'étapes requises par un algorithme pour résoudre ***toute instance*** dudit problème;

- Elle est exprimée en fonction de la ***taille*** de l'instance;

- L'étude de la complexité s'exprime donc souvent au «*pire des cas*» et pour une taille de problème suffisamment grande (non-triviale, p.ex. taille 1 ou 2);

- La complexité ne se mesure que pour les problèmes décidables en un nombre fini d'étapes pour une instance de taille finie.

Théorie de la complexité - Formalisation

Pour un problème donné,

- Désignons par n la taille d'une instance donnée et I_n un instance de taille n;
- Pour un algorithme A donné, le temps de résolution de I_n, dénoté $\tau_{\mathrm{A}}(I_n)$, est une fonction;
- Si A est un algorithme fini (il se termine en temps fini pour une instance de taille finie), alors il existe une constante c telle que et une fonction $f : \mathbb{N} \rightarrow \mathbb{N}$ tels que

$$\tau_A(I_n) \leq c \cdot f(n), \qquad \forall\, I_n$$

- De manière générale, nous ignorons la constante c pour ne nous intéresser qu'à la fonction de complexité, que nous écrirons

$$\tau_A(I_n) \leq O_A(f(n)), \qquad \forall\, I_n$$

Exemples de complexité

Pour un problème donné,

- La complexité est donc une mesure fortement liée à l'algorithme utilisé pour la résolution;
- La complexité d'un problème est définie comme la complexité du plus efficace des algorithmes pour résoudre le problème. Nous la noterons O.
- Il est souvent difficile, voir impossible, de prouver qu'un algorithme est «le meilleur» pour un problème donné;
- En revanche, s'il existe un algorithme avec une complexité $O_A(f(n))$ donnée, alors $O \leq O_A(f(n))$.

P vs NP

Question fondamentale

- La complexité d'un problème est-elle polynômiale en fonction de sa taille ?

- Si «Oui», quel est le degré du plus grand facteur ? Est-il constant (1,2,3…) ou dépendant de la taille ?

- P est l'ensemble des langages reconnus par une ***machine déterministe*** en temps polynomial.

- NP est l'ensemble des langages reconnus par une ***machine non-déterministe*** en temps polynomial.

- Bien que cela n'ait pas été prouvé, il est supposé que $P \neq NP$.

Définitions

- Réduction ($B \leq A$)

Un problème B peut être réduit à un problème A s'il existe fonction de transition permettant de résoudre B en utilisant un algorithme permettant de résoudre A.

- Problème NP-dur (ou NP-difficile):

Un problème A est dit NP-dur pour une réduction $\leq$ si pour tout problème B dans NP, on a $B \leq A$.

- Problème NP-complet:

Un problème A est dit NP-complet pour une réduction $\leq$ si $A \in$ NP et si A est NP-dur.

Remarques

Les affirmations suivantes sont équivalentes :

- $P = NP$;
- Tout problème NP-complet est dans P;
- Il existe un problème NP-complet dans P.

Exemple de problème NP-Complet

Problème SAT (problème de *satisfaisabilité*)

Soit une formule booléenne

$$\varphi(x_1, x_2, \dots, x_n) \text{ où } x_i \in \{0,1\}, \forall i = 1, \dots n$$

Le problème SAT consiste déterminer si la formule est ***satisfaisable***, autrement dit, s'il existe une *assignation* $a_1, \dots, a_n$ telle que $\varphi(a_1, a_2, \dots, a_n) = VRAI$.

Théorème (Cook 1971, Levin 1973):
Le problème SAT est NP-complet.

NP-complétude de SAT

Idée de la preuve:

Il est facile de montrer que SAT $\in$ NP (il suffit de deviner une assignation et la vérification de celle-ci se fait en temps polynomial) et donc SAT $\in$ NP $\supseteq$ P.

Il existe une formule polynomiale pour décrire la résolution du problème SAT sur une machine non-déterministe fonctionnant en temps polynomial.

Autres problèmes NP-complets

- Dans un graphe donné, déterminer s'il existe k sommets non-reliés deux-à-deux;

- NOTE: si k est fixé, alors le problème peut être résolu en temps polynômial, car la vérification se fait en temps $O(n^k)$!

- Pour une liste d'entiers et un entier données, existe-t-il un sous-ensemble de la liste tel que leur somme égale l'entier en question ?