Langages formels, calculabilité, complexité et analyse d'algorithmes

Paul Gastin
Paul.Gastin@liafa.jussieu.fr

http://liafa.jussieu.fr/~gastin/MMFAI/MMFAI.html

LIAFA, Université Paris 7, UMR CNRS 7089

3 Calculabilité : plan

1. Introduction
2. Machines de Turing
3. Complexité en temps et en espace
4. Machines de Turing et énumération
5. Machines de Turing non déterministes
6. Fonctions récursives
7. Décidabilité

3.1 Introduction

Calculabilité : Quels problèmes peut-on résoudre avec une machine (indépendamment de la technologie actuelle) ?

Nécessité de formaliser la notion de **problème** et la notion de **machine**.

Exemples

– Déterminer si un entier est pair.
– Déterminer si un entier est premier.

Un problème est une question générique (être pair, être premier) qui porte sur un ensemble de données (les entiers naturels) dont une description est fixée (binaire, décimal).

Une instance du problème est la question posée sur une donnée particulière (est-ce que 18 est pair ?).

Un problème est indépendant d'un éventuel algorithme qui le résout.

3.1 Problèmes de décision

Définition 3.1 *Un* problème binaire *est constitué par*
– *L'ensemble (dénombrable) de ses données*,
– *la description de la représentation de ces données*.
– *Un énoncé portant sur ces données pouvant être vrai ou faux*.

Exemples :

Donnée : Un entier écrit en base 10 ;
Question : cet entier est-il premier ?

Donnée : Un entier écrit en base 10 ;
Question : cet entier est-il la somme de 4 carrés ?

Problème de Post
Donnée : des couples de mots $(u_1, v_1), \ldots, (u_k, v_k)$;
Question : Existe-t-il $i_1, \ldots, i_n$ tels que $u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n}$?

3.1 Problèmes de décision

Accessibilité

Donnée : Un graphe fini représenté par le nombre de sommets, suivie de la matrice d'adjacences et deux sommets u et v du graphe ;
Question : Le sommet v est-il accessible à partir du sommet u ?

Syntaxe du langage C

Donnée : un texte ASCII ;
Question : Est-ce un programme C syntaxiquement correct ?

Arrêt universel

Donnée : un programme C ;
Question : Est-ce que ce programme s'arrête sur toutes ses données ?

Arrêt existentiel

Donnée : un programme C ;
Question : Est-ce que ce programme s'arrête sur au moins une donnée ?

3.1 Fonctions

Il y a des problèmes qui ne sont pas binaires.

Exemples

– Calculer $n!$.
– Trier un tableau.
– Calculer le flot maximal dans un graphe.

Un tel problème est une fonction de l'ensemble des données du problème dans l'ensemble des résultats.

Un problème binaire est une fonction à valeurs dans $\{\text{oui}, \text{non}\}$.

3.1 Formalisation

Problème = Langage

Une donnée ou un résultat est un **mot** écrit sur un alphabet A.
L'ensemble des données d'un problème est un **langage** sur A^*.

Un problème partitionne A^* en trois ensembles :
– *Instances positives*
– *Instances négatives*
– *non instances*
Souvent, on identifie un problème à l'ensemble de ses instances positives.

Lorsque le problème n'est pas binaire, il se formalise par une fonction $f : A^* \longrightarrow B^*$ définie sur l'ensemble $D \subseteq A^*$ des données du problème.

Théorème 3.2 *Il y a des problèmes qu'on ne peut pas résoudre par un programme C.*

3.1 Procédures effectives

Procédures effectives entrée $\longrightarrow$ Machine $\longrightarrow$ sortie

Il faut formaliser la notion de machine.
On veut un modèle très simple pour capturer tout ce qu'on peut calculer.

– Automates

Problème : mémoire bornée (états de l'automate).
Remarque : Si on autorise un nombre infini (dénombrable) d'états, on peut reconnaître n'importe quel langage avec un automate.
Le problème est que l'automate n'admet plus une description finie.

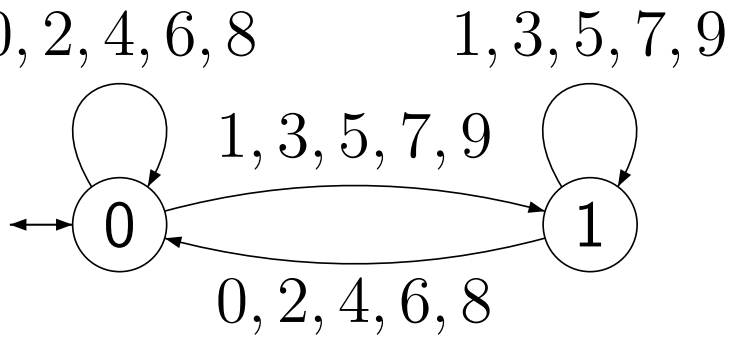
– Automates à pile

Mémoire : états + pile : non bornée mais toujours finie durant un calcul.
Mais on ne sait pas reconnaître $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.

Ce sont des classes de machines obéissant aux mêmes règles.
Des machines d'une même classe différent par leur programme.

3.1 Exemples d'automates

Nombres pairs :

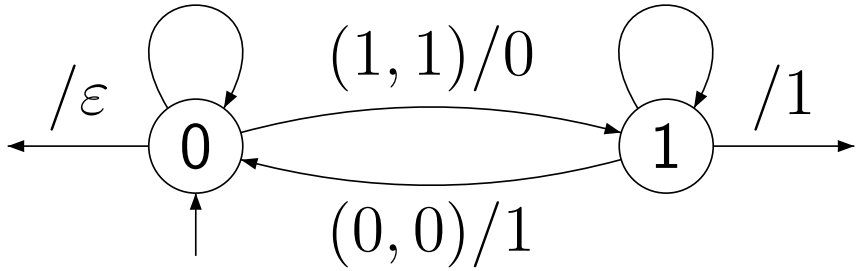


$(0,0)/0$ $(1,1)/1$

$(0,1)/1$ $(0,1)/0$

$(1,0)/1$ $(1,0)/0$

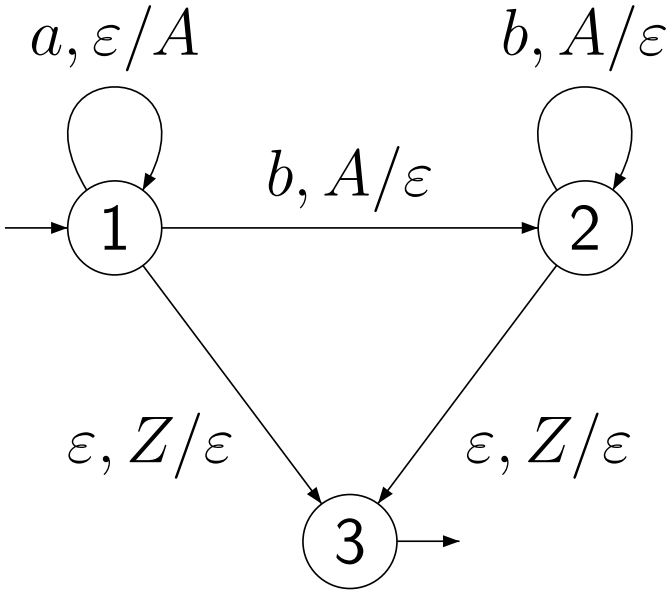
Addition en binaire :



On suppose que les nombres sont lus de droite à gauche.

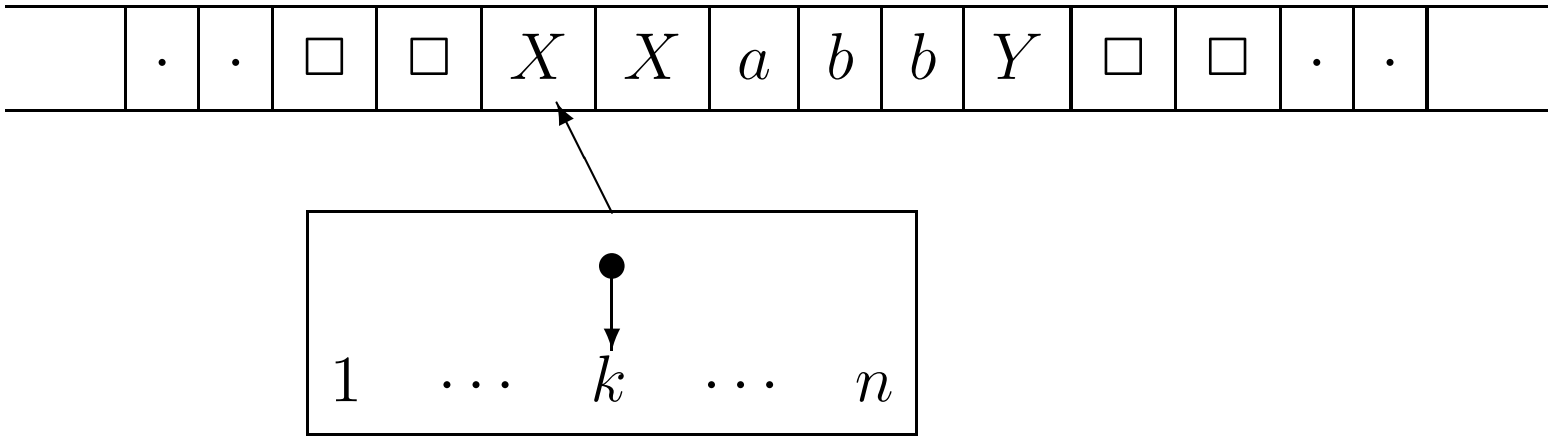
3.1 Exemple d'automate à pile

Le langage $L = \{a^n b^n \mid n \geq 0\}$ est reconnu par l'automate à pile :



Au cours de chaque calcul, on utilise une pile de taille finie, mais cette taille n'est pas bornée a priori par la définition de l'automate.

3.2 Machines de Turing (Alan, 1912 – 1954)



Nombre fini d'états

Mémoire auxiliaire : bande infinie avec une tête de lecture/écriture.

On peut
– lire,
– écrire,
– déplacer la tête d'une case vers la gauche ou vers la droite.

3.2 Machines de Turing : Définition

Définition 3.3 *MT déterministe à 1 bande :* $\mathcal{M} = (Q, H, s, \Sigma, \Gamma, \delta)$

– Q *: ensemble fini d'états,*

– $s \in Q$ *: état initial,*

– H *: ensemble des états d'arrêt (Halt states),*

– Γ *: alphabet (fini) de la bande (*$\square \in \Gamma$*),*

– $\Sigma \subseteq \Gamma \setminus \{\square\}$ *: l'alphabet (fini) d'entrée,*

– *fonction de transitions :* $\delta : Q \times \Gamma \longrightarrow (Q \cup H) \times \Gamma \times \{\leftarrow, -, \rightarrow\}$

– *Configuration :* $(q, u, a, v) \in (Q \cup H) \times \Gamma^* \times \Gamma \times \Gamma^*$

En général, $u \notin \square\Gamma^*$ *et* $v \notin \Gamma^*\square$.

– *Config. initiale pour* u *:* $CI(u) = \begin{cases} (s, \varepsilon, a, u') & \text{si } u = au' \in \Sigma^+ \\ (s, \varepsilon, \square, \varepsilon) & \text{sinon} \end{cases}$

– *Exécution d'une transition :* $(p, u, a, v) \vdash (q, u', a', v')$

3.2 Machines de Turing : Définition

Définition 3.4 *MT d'acceptation* $\mathcal{M}$ *:* $\mathrm{OK} \in H$.

– $u \in \Sigma^*$ *est accepté (reconnu) par* $\mathcal{M}$ *si* $CI(u) \vdash^* (\mathrm{OK}, x, y, z)$

– $L(\mathcal{M})$ *: ensemble des mots reconnus par* $\mathcal{M}$.

– L *est récursivement énumérable (semi-décidable) s'il existe une MT* $\mathcal{M}$ *telle que* $L = L(\mathcal{M})$.

– L *est* décidable *(*calculable, récursif*) s'il existe une MT* sans exécution infinie *qui l'accepte.*

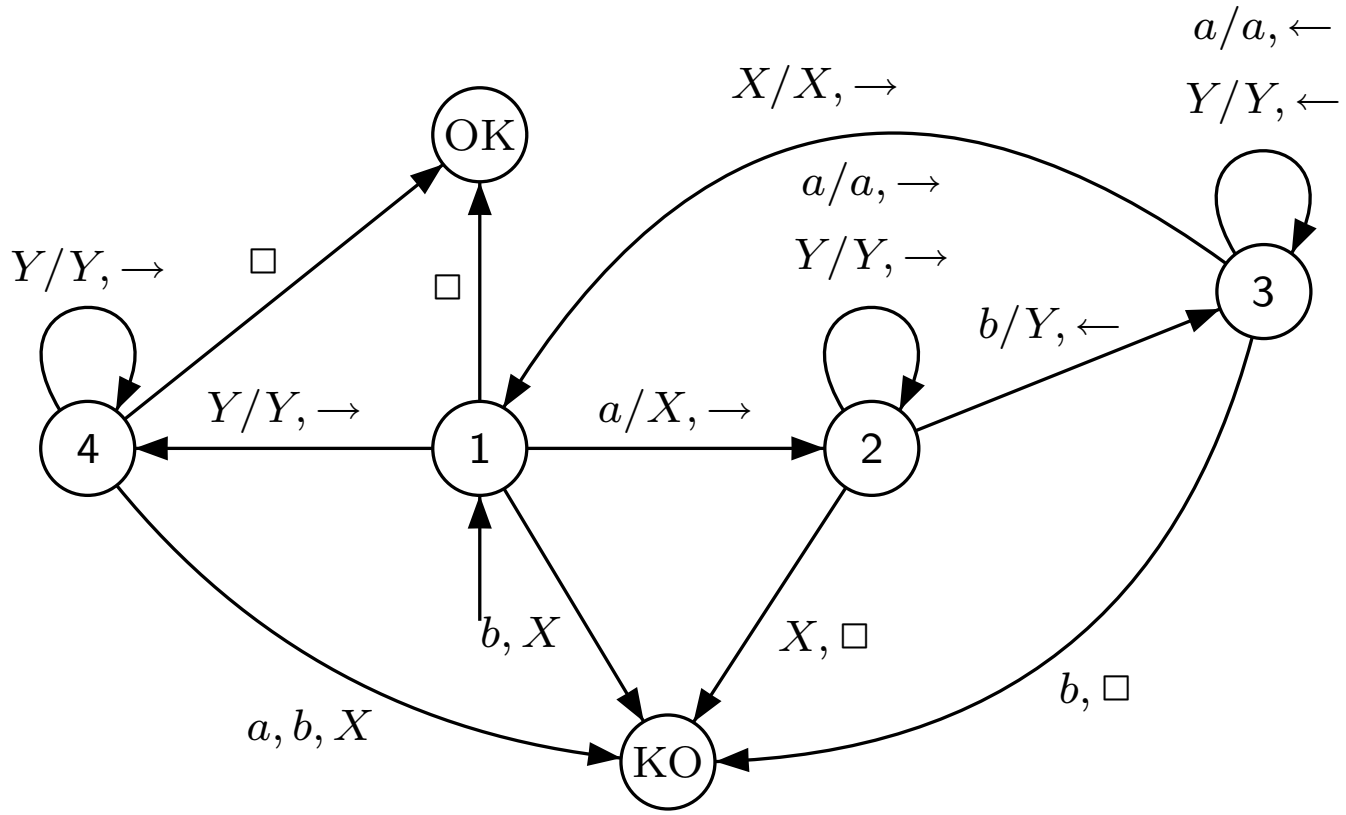
Remarque : On peut transformer une machine d'acceptation pour qu'elle efface sa bande avant de s'arrêter, ou bien pour qu'elle repositionne sa tête au "début" de la bande.

Remarque : Une MT $\mathcal{M}$ définit une procédure effective pour décider si un mot $u \in L(\mathcal{M})$ seulement si $\mathcal{M}$ s'arrête toujours.

3.2 MT : Exemple

Exemple : MT pour le langage $\{a^n b^n \mid n \geq 0\}$.

$\mathcal{M} = (Q, H, s, \Sigma, \Gamma, \delta)$ avec $Q = \{1, 2, 3, 4\}$, $H = \{\mathrm{OK}, \mathrm{KO}\}$, $s = 1$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, X, Y, \square\}$



3.2 MT : Exemple

Avec un langage de plus haut niveau

TQ $R = a$ faire
    $W(X)$; D; TQ $R \in \{a, Y\}$ faire D FTQ;
    si $R \neq b$ alors KO
    $W(Y)$; TQ $R \in \{a, Y\}$ faire G FTQ; D
FTQ;
TQ $R = Y$ faire D FTQ;
si $R = \square$ alors OK sinon KO fsi

3.2 MT : Temps de calcul

Chaque transition de la MT prend 1 unité de temps.

Temps sur $u \in \Sigma^*$: nombre de transitions jusqu'à un état d'arrêt.

Définition 3.5 *Soit $f : \mathbb{N} \longrightarrow \mathbb{N}$, une MT $\mathcal{M}$ travaille en temps*
– f si $\forall u \in \Sigma^*$, $\mathcal{M}$ sur u s'arrête en au plus $f(|u|)$ transitions.
– $\mathcal{O}(f)$ si elle travaille en temps g pour $g \in \mathcal{O}(f)$.

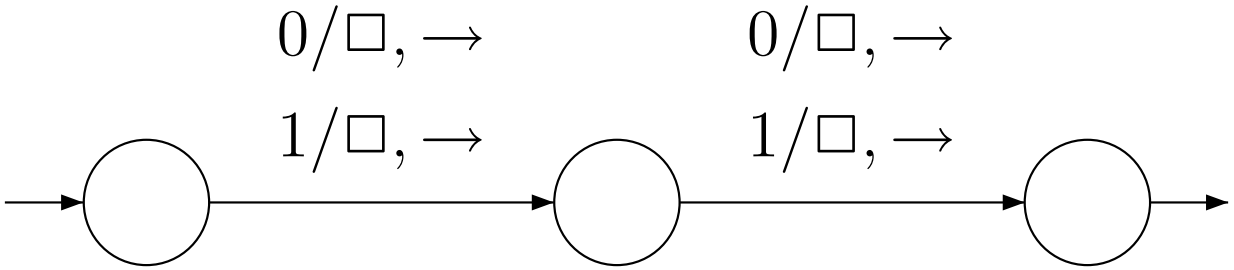
Exemple : La MT pour $\{a^n b^n \mid n \geq 0\}$ travaille en temps $\mathcal{O}(n^2)$.

En général, on suppose que $f(n) \geq n$: il faut au moins lire la donnée.
(Ce n'est pas toujours le cas : être pair en binaire).

3.2 MT : Fonction calculable

Définition 3.6 *Une machine de Turing peut être utilisée pour calculer une fonction f.*
La donnée $u \in \Sigma^*$ est écrite sur la bande comme précédemment.
La fonction est définie sur u si la machine atteint l'état OK. Le résultat $f(u)$ est alors le mot écrit sur la bande entre les symboles $\square$.

Exemple : Division par 4 d'un entier en binaire.
Le bit de poids faible est à gauche.



Exercice : Fonction miroir en temps quadratique.

3.2 MT : Composition

Soient $\mathcal{M}_1$ et $\mathcal{M}_2$ des MT qui calculent les fonctions f et g.

On suppose qu'avant de s'arrêter les MT positionnent leur tête de lecture/écriture sur le premier caractère du résultat.

La MT $\mathcal{M}_2 \circ \mathcal{M}_1$ définie ci-dessous calcule la fonction $g \circ f$.
– $Q = Q_1 \uplus Q_2$, $s = s_1$, $H = H_2$,
– $\delta = \delta_1 \uplus \delta_2$ à la différence près qu'on remplace les transitions $p, x \longrightarrow \mathrm{OK}, y, z$ de δ_1 par $p, x \longrightarrow s_2, y, z$.

3.2 MT : Programmation

But : Langage de plus haut niveau permettant de définir facilement et de façon compréhensible des MT. Une MT sera décrite par un programme. Un programme se compilera en une MT.

On supposera toujours que $\mathrm{OK}, \mathrm{KO} \in H$.

Instructions élémentaires

– Écriture : $W(x)$,
– Lecture : R,
– Déplacements : G et D,
– Arrêts : STOP(h) où $h \in H$.

3.2 MT : Programmation

Composition séquentielle

$$\mathcal{M}_1; \mathcal{M}_2$$

Alternative

Machine de test : deux états d'arrêt spéciaux $V, F \in H \setminus \{\mathrm{OK}, \mathrm{KO}\}$.
Soient M_0 une machine de test et M_1, M_2 deux MT.
On définit la MT

si M_0 alors M_1 sinon M_2 finsi

Exemple : si $R = \square$ alors M_1 sinon M_2 finsi

3.2 MT : Programmation

Cas parmi

Une machine de choix a n états d'arrêt spéciaux :

$$C_1, \ldots, C_n \in H \setminus \{\mathrm{OK}, \mathrm{KO}\}$$

Soient M_0 une machine de choix et $M_1, \ldots, M_n$ des MT.
On définit la MT

Cas M_0 parmi
$C_1 : M_1$
...
$C_n : M_n$
Fincas

3.2 MT : Programmation

Tant que

Soit M_0 une machine de test et soit M_1 une MT ayant un état d'arrêt spécial

$$F_1 \in H_1 \setminus \{\mathrm{OK}, \mathrm{KO}\}$$

On définit la MT

Tant que M_0 faire M_1 finTQ

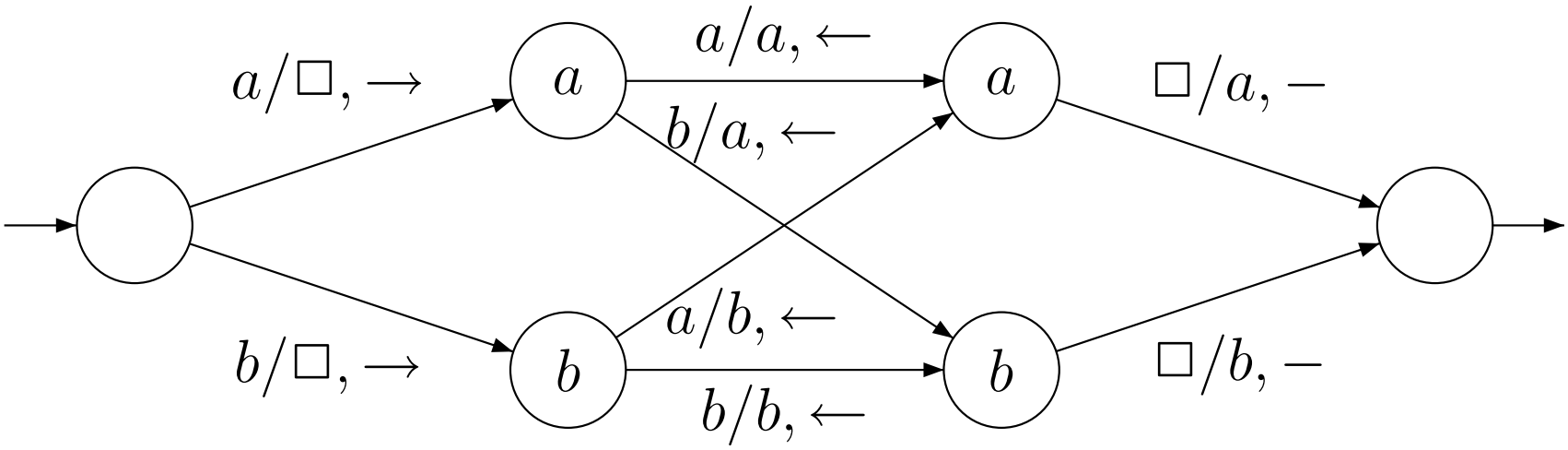
Reset : TQ $R \neq \square$ faire G FTQ

3.2 MT : Programmation

Variables à domaines finis

Exemple : Échange des deux premiers caractères.

var $x, y : \Gamma$;
$x := R$; D; $y := R$; $W(x)$; G; W(y); STOP



3.2 MT : Bande infinie à droite

Le mot d'entrée se trouve au début de la bande.
La machine s'arrête dans un état d'erreur si on essaie d'aller à gauche de la première case du ruban.

Proposition 3.7 *Soit $\mathcal{M} = (Q, H, s, \Sigma, \Gamma, \delta)$ une MT à bande infinie à droite uniquement. On peut effectivement construire une MT $\mathcal{M}'$ équivalente à bande bi-infinie.*

Proposition 3.8 *Soit $\mathcal{M} = (Q, H, s, \Sigma, \Gamma, \delta)$ une MT à bande bi-infinie. On peut effectivement construire une MT $\mathcal{M}'$ équivalente à bande à droite uniquement.*

Remarque : les simulations sont linéaires.

3.2 MT à plusieurs bandes

Définition 3.9 $\mathcal{M} = (Q, H, s, \Sigma_1, \ldots, \Sigma_k, \Gamma_1, \ldots, \Gamma_k, \delta)$.

k bandes. k têtes indépendantes.

Alphabets : $\Sigma_1, \ldots, \Sigma_k$ *et* $\Gamma_1, \ldots, \Gamma_k$.

Donnée : mot de $\Sigma_1^* \times \cdots \times \Sigma_k^*$.

Transitions :

$$\delta : Q \times \Gamma_1 \times \cdots \times \Gamma_k \longrightarrow Q \times \Gamma_1 \times \cdots \times \Gamma_k \times \{\leftarrow, -, \rightarrow\}^k$$

Exemple : Addition en binaire.

Théorème 3.10 *On peut simuler en temps quadratique une MT à plusieurs bandes avec une MT à une seule bande.*

3.3 Complexité en temps et en espace

Définition 3.11 (Complexité en temps)
Soit $f : \mathbb{N} \longrightarrow \mathbb{N}$.
Un langage L *est décidable en temps* f *s'il existe une MT* $\mathcal{M}$ *à plusieurs bandes qui décide* L *et travaille en temps* f.

TIME(f) *: classe des langages décidables en temps* f

Théorème 3.12 (Accélération linéaire)
Soient $f : \mathbb{N} \longrightarrow \mathbb{N}$ *et* $\varepsilon > 0$. *Alors*

$$\textbf{TIME}(f(n)) \subseteq \textbf{TIME}(\varepsilon f(n) + \varepsilon n + n + 5)$$

On choisit m tel que $\frac{3}{m} < \varepsilon$.
On construit $\mathcal{M}'$ qui simule m pas de calcul de $\mathcal{M}$ en 3 pas de calcul.

3.3 Complexité en temps

Conséquences :

– $\mathbf{TIME}(n\ln(n)) = \mathbf{TIME}(8n\ln(n)+3n+6) = \mathbf{TIME}(10^{-6}n\ln(n))$
– Si P est un polynôme de degré $k>1$, $\mathbf{TIME}(P) = \mathbf{TIME}(n^k)$
– Si f est linéaire, on peut rendre la constante aussi proche de 1 que l'on veut.

En complexité, on ne s'intéresse qu'aux ordres de grandeur :

si f est super-linéaire, $\mathbf{TIME}(f) = \bigcup_{g\in\mathcal{O}(f)} \mathbf{TIME}(g)$.

Pour un ordinateur ceci correspond à :
– Accélérer l'horloge.
– Grouper plusieurs instructions en une seule.
– Augmenter le nombre de processeurs.
Tout ceci agit sur les constantes, pas sur l'ordre de grandeur.

3.3 MT avec entrées/sorties

– **bande d'entrée :** la MT n'écrit jamais.

– **MT off-line :** la tête ne se déplace que vers la droite sur les bandes d'entrée.

– **bande de sortie :** bande initialement vide sur laquelle la tête ne se déplace que vers la droite.

– **MT d'E/S :** p bandes d'entrée, q bandes de sortie et k bandes de travail.

La complexité en espace se compte à partir d'une MT à plusieurs bandes et sans compter les bandes d'entrée et les bandes de sortie.
On ne veut compter que la mémoire auxiliaire utilisée.

3.3 Complexité en espace

Définition 3.13 (Complexité en espace)
Soit $f : \mathbb{N} \longrightarrow \mathbb{N}$, $\mathcal{M}$ *une MT d'E/S et* $\mathcal{C}$ *un calcul de* $\mathcal{M}$.

– $L_i(\mathcal{C})$ *taille maximale de la bande de travail* B_i *au cours du calcul* $\mathcal{C}$.

– $L(\mathcal{C}) = \sum_{i=1}^{k} L_i(\mathcal{C})$.

Une MT $\mathcal{M}$ *calcule en espace (au plus)* f *si* $\forall w = (w_1, \ldots, w_p)$,

$$L(\mathcal{C}(w)) \leq f(|w|)$$

(où $|w| = |w_1| + \cdots + |w_p|$ *et* $\mathcal{C}(w)$ *est le calcul de* $\mathcal{M}$ *sur* w*).*

SPACE(f) *: classe des langages décidables par une MT en espace* f.

3.3 Complexité en espace

Théorème 3.14 *Soit* $f : \mathbb{N} \longrightarrow \mathbb{N}$ *et soit* $\varepsilon > 0$. *Alors*

$$\textbf{SPACE}(f) \subseteq \textbf{SPACE}(2 + \varepsilon f).$$

Conséquence : les constantes ne sont pas importantes.

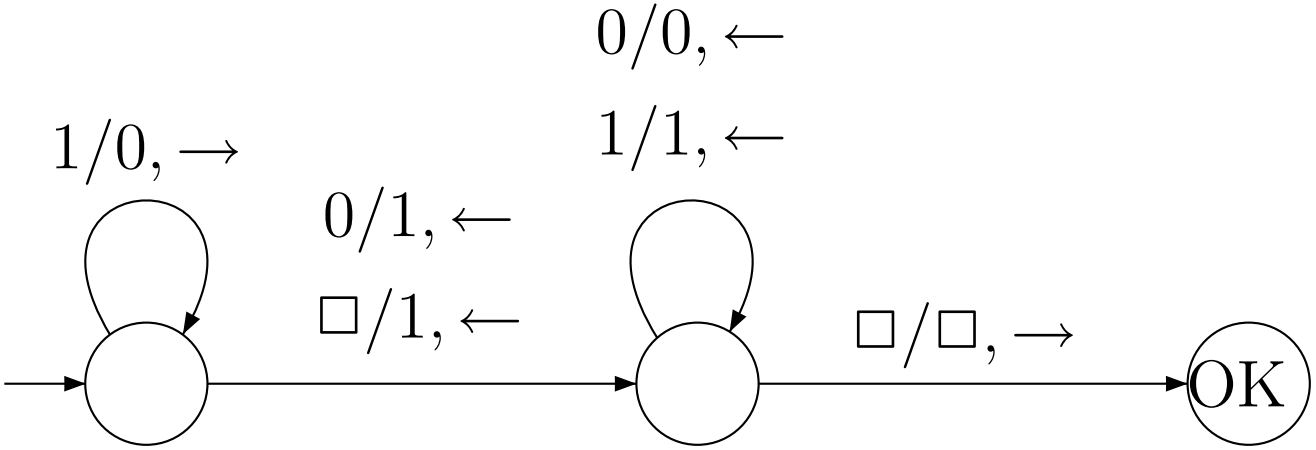
$$\textbf{SPACE}(f) = \bigcup_{g \in \mathcal{O}(f)} \textbf{SPACE}(g).$$

3.3 Opérations sur les entiers en binaire

Les entiers sont codés en binaire sur les bandes.
Le bit de poids faible est à gauche.
– Affectation : $B := n$, $B := B'$
– Comparaisons : $B = B'$, $B \neq B'$, $B < B'$, ...
– Opérations : $B := B' + B''$, $B := B' - B''$, $B := B' * B''$, $B := B'/B''$, B++, B--, ...
Ces machines replacent toujours la tête au début de la bande.

– La multiplication et la division travaillent en espace linéaire et en temps quadratique.
– Les autres machines travaillent en temps linéaire et en espace constant ou linéaire (i.e. logarithmique par rapport aux entiers manipulés).

3.3 exemple : incrémentation



Soit x le nombre de 1 à gauche. Le temps de calcul est $2(x+1)$.
L'incrémentation, la décrémentation, la comparaison à une constante sont en coût amorti $\mathcal{O}(1)$:

$$B := 0;\ \text{TQ}\ B \neq N\ \text{faire}\ B{+}{+}\ \text{FTQ}$$

Majoration du temps cumulé des incrémentations ($2^{k-1} \leq N < 2^k$) :

$$\sum_{i=1}^{k} 2^{k-i} \cdot 2 \cdot i = 2^k \sum_{i=1}^{k} i \cdot (1/2)^{i-1} \leq 2^k \cdot 4 \leq 8N$$

3.3 Palindromes en SPACE$(\log n) \cap$ TIME(n^2)

– Bande d'entrée : B_1 contient la donnée w.
– Bandes 2 et 3 : compteurs de valeur $\leq n+1$ ($n = |w|$).

var $x, y : \Sigma$;	
$B_2 := 1$;	$\mathcal{O}(1)$
TQ Vrai faire	n fois
$x :=$ caractère en position B_2 en partant de la gauche;	$\mathcal{O}(n)$
si $x = \square$ alors STOP(OK) fsi;	$\mathcal{O}(1)$
$y :=$ caractère en position B_2 en partant de la droite;	$\mathcal{O}(n)$
si $x \neq y$ alors STOP(KO) fsi;	$\mathcal{O}(1)$
$B_2{+}{+}$	$\mathcal{O}(1)$
FTQ	

Complexité en temps $\mathcal{O}(n^2)$ et en espace $\mathcal{O}(\log n)$.

Exercice : Tester les palindromes en **SPACE**$(n) \cap$ **TIME**(n).

3.3 Classes de complexité déterministes

$$\mathbf{P} = \bigcup_{k \geq 0} \mathbf{TIME}(n^k)$$

$$\mathbf{EXP} = \bigcup_{k \geq 0} \mathbf{TIME}(2^{n^k})$$

$$\mathbf{L} = \mathbf{SPACE}(\log n)$$

$$\mathbf{PSPACE} = \bigcup_{k \geq 0} \mathbf{SPACE}(n^k)$$

3.4 Machines de Turing et énumération

Ordres sur Σ^*

$\Sigma = \{a_1, \ldots, a_n\}$ alphabet ordonné $a_1 < \cdots < a_n$.

Ordre lexicographique :

$$u <_{\text{lex}} v \iff \begin{cases} v = uw & \text{avec } w \in \Sigma^+ \\ \text{ou} & \\ u = wau', v = wbv' & \text{avec } a < b \end{cases}$$

L'ordre lexicographique est un ordre total qui n'est pas bien fondé :

$$b >_{\text{lex}} ab >_{\text{lex}} aab >_{\text{lex}} aaab >_{\text{lex}} \cdots$$

Il ne permet pas d'énumérer les mots de Σ^*.

3.4 Ordre hiérarchique

$$u \prec v \iff \begin{cases} |u| < |v| \\ \text{ou} \\ |u| = |v| \text{ et } u <_{\text{lex}} v \end{cases}$$

L'ordre hiérarchique est un ordre total isomorphe à $(\mathbb{N}, <)$.
Il permet d'énumérer les mots de Σ^* :

$$\varepsilon \prec a \prec b \prec aa \prec ab \prec ba \prec bb \prec \cdots$$

Remarque : Si on considère l'alphabet $\Sigma = \{0, 1\}$ avec l'ordre $0 < 1$ et qu'on se restreint aux mots de $1\Sigma^* \cup \{\varepsilon\}$ alors l'ordre hiérarchique correspond à l'ordre *naturel* sur les entiers.

3.4 Successeur dans l'ordre hiérarchique

$$\begin{aligned} \mathrm{succ}(ua_k a_n^p) &= ua_{k+1}a_1^p \qquad \text{si } k < n \\ \mathrm{succ}(a_n^p) &= a_1^{p+1} \end{aligned}$$

MT : la tête de lecture est à la droite du mot.

TQ $R = a_n$ faire $W(a_1)$; G FTQ;

Cas R parmi

$\square$: $W(a_1)$

a_1 : $W(a_2)$

$\vdots$

a_{n-1}: $W(a_n)$

FinCas

TQ $R \neq \square$ faire D FTQ; G;

Complexité en temps : linéaire $\approx 2|u|$ et en espace : $\mathcal{O}(1)$.

Coût amorti en temps : $\mathcal{O}(1)$.

3.4 Exercices

Ecrire une MT qui travaille en temps linéaire et en espace constant pour comparer dans l'ordre hiérarchique : COMP(B_1, B_2)
Données u et v sur les bandes B_1 et B_2.
États d'arrêt : $\{\prec, =, \succ\}$.

Montrer que la relation sur $\mathbb{N} \times \Sigma^*$ définie par $(n, u) < (m, v)$ ssi
– $n + |u| < m + |v|$ ou bien
– $n + |u| = m + |v|$ et $n < m$ ou bien
– $n = m$ et $|u| = |v|$ et $u <_{\text{lex}} v$
est un ordre total isomorphe à $\mathbb{N}$.

Écrire la fonction SUCC correspondante.

3.4 Énumération

Définition 3.15 *Soit $\mathcal{M}$ MT avec un état spécial* $\mathrm{ENUMERER} \in Q$ *et dont une bande (initialement vide) est appelée bande d'énumération. Le langage énuméré par $\mathcal{M}$ est l'ensemble des mots écrits sur la bande d'énumération de $\mathcal{M}$ à chaque passage dans l'état* $\mathrm{ENUMERER}$.

Un langage énuméré peut être fini ou infini.

Exemple : énumération de Σ^* dans l'ordre hiérarchique.

$\mathrm{ENUMERER}$; /* mot vide */
TQ VRAI faire SUCC; $\mathrm{ENUMERER}$ FTQ;

3.4 Énumération

Proposition 3.16 *Un langage est décidable ssi il peut être énuméré dans l'ordre hiérarchique par une MT.*

Proposition 3.17 *Un langage est récursivement énumérable ssi il peut être énuméré par une MT.*

3.5 Machines de Turing non déterministes

Définition 3.18 *$\mathcal{M}$ machine de Turing non déterministe à k bandes.*

- *Il suffit de considérer une fonction de transition non déterministe :*

$$\delta : Q \times \Gamma^k \longrightarrow \mathcal{P}((Q \cup H) \times \Gamma^k \times \{\leftarrow, -, \rightarrow\}^k)$$

- *Un mot w est accepté par $\mathcal{M}$ s'il existe un calcul acceptant pour w, même si d'autres calculs rejettent w ou ne s'arrêtent pas.*
- *$L(\mathcal{M})$ est l'ensemble des mots acceptés par $\mathcal{M}$*
- *$\mathcal{M}$ décide un langage L si $L = \mathcal{L}(\mathcal{M})$ et tous les calculs de $\mathcal{M}$ s'arrêtent.*

3.5 Exemple : Accessibilité dans un graphe

Données : Un graphe $G = (S, A)$ défini par le nombre n de sommets, la matrice d'adjacence $A \in \{0,1\}^{n^2}$ et deux sommets $u, v \in \{0, \ldots, n-1\}$.
Les entiers n, u, v sont représentés en binaire.

Taille des données : $|G|$ avec $|G| \approx n^2 + 3 \log n$.

Problème : Existe-t-il un chemin de u à v dans le graphe G ?

On utilise les bandes d'entrées N, A, U, V.

3.5 Solution déterministe

$B_3 := U\sharp$; (liste des sommets déjà vus)	$\mathcal{O}(\log n)$
$B_4 := U\sharp$; (liste des sommets à traiter)	$\mathcal{O}(\log n)$
TQ $R_4 \neq \square$ faire	n fois
Copier en l'effaçant le premier sommet de B_4 sur B_1;	$\mathcal{O}(\log n)$
si $B_1 = V$ alors STOP(OK) fsi;	$\mathcal{O}(\log n)$
Déplacer la tête de A en position $N * B_1$;	$\mathcal{O}(n^2)$
POUR $B_2 := 0$ à $N - 1$ faire	n fois
si $R_A = 1$ et $B_2 \notin B_3$ alors	$\mathcal{O}(n \log n)$
Ajouter $B_2\sharp$ à B_3 et à B_4;	$\mathcal{O}(n \log n)$
fsi;	
D_A;	$\mathcal{O}(1)$
FPOUR;	
FTQ	
STOP(KO)	$\mathcal{O}(1)$

Complexité en temps : $n^3 \log n \leq |G|^2$ et en espace : $n \log n \leq |G|$

3.5 Solution non déterministe

$B_1 := U$;	$\mathcal{O}(\log n)$
POUR $B_3 := 1$ à $N - 1$ faire	n fois
si $B_1 = V$ alors STOP(OK) fsi;	$\mathcal{O}(\log n)$
$B_2 := N * B_1$;	$\mathcal{O}(\log^2 n)$
Écrire sur B_1 aléatoirement $\lceil \log_2 n \rceil$ bits	$\mathcal{O}(\log n)$
si $B_1 \geq N$ alors STOP(KO) fsi;	$\mathcal{O}(\log n)$
Déplacer la tête de A en position $B_2 + B_1$;	$\mathcal{O}(n^2)$
si $R_A = 0$ alors STOP(KO) fsi;	$\mathcal{O}(1)$
FPOUR	
Si $B_1 = V$ alors STOP(OK) sinon STOP(KO) fsi	$\mathcal{O}(\log n)$

Complexité en temps : $n^3 \leq |G|^2$ et en espace : $\log n \leq \log |G|$

3.5 Complexité des MT non déterministes

Définition 3.19 *Soit* $f, g : \mathbb{N} \longrightarrow \mathbb{N}$.

- *Une MT non déterministe* $\mathcal{M}$ *décide un langage* L *en temps* f *et en espace* g *si*
  - $L(\mathcal{M}) = L$
  - $\forall w \in \Sigma^*$ *et pour tout calcul* $\mathcal{C}$ *de* $\mathcal{M}$ *sur* w*, le calcul* $\mathcal{C}$ *s'arrête après au plus* $f(|w|)$ *transitions et utilise un espace de travail inférieur ou égal à* $g(|w|)$.
- *On note* **NTIME**(f) *la classe des langages décidables en temps* f *par une MT non déterministe.*
- *On note* **NSPACE**(g) *la classe des langages décidables en espace* g *par une MT non déterministe.*

3.5 Classes de complexité non déterministes

$$\mathbf{NP} = \bigcup_{k \geq 0} \mathbf{NTIME}(n^k)$$

$$\mathbf{NEXP} = \bigcup_{k \geq 0} \mathbf{NTIME}(2^{n^k})$$

$$\mathbf{NL} = \mathbf{NSPACE}(\log n)$$

$$\mathbf{NPSPACE} = \bigcup_{k \geq 0} \mathbf{NSPACE}(n^k)$$

3.5 MT non déterministes

Proposition 3.20 *On peut simuler une MT non déterministe à k bandes par une MT non déterministe à 1 bande avec une perte de temps quadratique.*

On verra dans la suite du cours les résultats suivants :

Théorème 3.21 (Savitch)

$$\textit{Accessibilité} \in \mathbf{SPACE}(\log^2 n).$$

Corollaire 3.22

- $\mathbf{NSPACE}(f) \subseteq \mathbf{SPACE}(f^2)$.
- $\mathbf{PSPACE} = \mathbf{NPSPACE}$.

3.5 Problème du voyageur de commerce

Données : Le nombre n de villes en binaire. Les distances $(d_{i,j})_{1 \leq i,j \leq n}$ entre les villes.
Problème : Trouver la tournée qui minimise le trajet parcouru, i.e. trouver une permutation σ de $1, \ldots, n$ telle que le coût $C(\sigma)$ est minimal. Avec la convention $\sigma(n+1) = \sigma(1)$, le coût de σ est :

$$C(\sigma) = \sum_{i=1}^{n} d_{\sigma(i),\sigma(i+1)}$$

Variante avec un budget fixé :
Données : Le nombre n de villes en binaire. Les distances $(d_{i,j})_{1 \leq i,j \leq n}$ entre les villes. Le budget B à ne pas dépasser pour réaliser la tournée.
Problème : Existe-t-il une tournée dont le coût est inférieur à B, i.e. Existe-t-il une permutation σ de $1, \ldots, n$ telle que $C(\sigma) \leq B$?

3.5 Voyageur de commerce

Bandes d'entrées :

N : le nombre de villes en binaire

B : le budget en binaire

Δ : $d_{1,1}\sharp\cdots\sharp d_{1,n}\sharp d_{2,1}\sharp\cdots\sharp d_{n,n}$ tous en binaire.

Taille des données : $|w|$ avec $|w| \geq n^2$. En fait $|w| = \mathcal{O}(n^2)$ si les distances et le budget sont bornés indépendamment de n.

Solution déterministe :

– Énumérer toutes les permutations.

– Pour chacune d'elles, calculer le coût.

– Si un coût est inférieur à B on accepte, sinon on rejette.

Pour cet algorithme, la complexité en espace est linéaire,
mais la complexité en temps est non polynomiale ($n!$ permutations).
On ne connaît pas de solution dans **P** à ce problème.

3.5 Voyageur de commerce

Solution non déterministe :

– Deviner une permutation.

– Calculer son coût.

– Si ce coût est inférieur à B on accepte, sinon on rejette.

Pour cet algorithme, la complexité en espace est linéaire,
la complexité en temps est quadratique.

3.5 Génération aléatoire d'une permutation

POUR $B_1 := 1$ à N faire	n fois
Écrire sur B_2 aléatoirement $\lceil \log_2 n \rceil$ bits	$\mathcal{O}(\log n)$
si $B_2 = 0$ ou $B_2 > N$ alors STOP(KO) fsi;	$\mathcal{O}(\log n)$
si B_2 est déjà sur X alors STOP(KO) fsi;	$\mathcal{O}(n \log n)$
Écrire $B_2\sharp$ à la fin de X; reset(X)	$\mathcal{O}(n \log n)$
Effacer B_2;	$\mathcal{O}(\log n)$
FPOUR	

Complexité en temps : $n^2 \log n \leq |w|^2$ et en espace : $n \log n \leq |w|$.

3.5 Coût de la permutation

Copier le premier nombre de X sur B_1	$\mathcal{O}(\log n)$
Copier $B_1\sharp 0\sharp$ à la fin de X; reset X	$\mathcal{O}(n \log n)$
Copier le deuxième nombre de X sur B_2	$\mathcal{O}(\log n)$
$C := 0$;	$\mathcal{O}(1)$
TQ $B_2 \neq 0$ faire	n fois
$B_3 := (B_1 - 1) * N + B_2 - 1$;	$\mathcal{O}(\log^2 n)$
Chercher la valeur en position B_3 sur Δ;	$\mathcal{O}(\|w\|)$
Ajouter cette valeur à C;	$\mathcal{O}(\log \|w\|)$
$B_1 := B_2$;	$\mathcal{O}(\log n)$
Copier le nombre suivant de X sur B_2	$\mathcal{O}(\log n)$
FTQ	
Si $C \leq B$ alors STOP(OK)	
sinon STOP(KO) fsi	$\mathcal{O}(\log n)$

Complexité en temps : $n|w| \leq |w|^2$ et en espace : $|w|$ (à cause de C).

3.5 Simulation d'une MT ND par une MT D

Théorème 3.23 *Soit $\mathcal{M}$ une MT non déterministe acceptant un langage L. Il existe une MT déterministe $\mathcal{M}'$ acceptant le même langage L.*

Si de plus, $\mathcal{M}$ décide L en temps f alors $\mathcal{M}'$ décide L en temps $2^{\mathcal{O}(f)}$

Corollaire 3.24 $\mathbf{NTIME}(f) \subseteq \mathbf{TIME}(2^{\mathcal{O}(f)})$

Corollaire 3.25 $\mathbf{NP} \subseteq \mathbf{EXP}$

Preuve: $\mathbf{NTIME}(n^k) \subseteq \mathbf{TIME}(2^{\mathcal{O}(n^k)}) \subseteq \mathbf{TIME}(2^{n^{k+1}})$ □

3.5 Preuve

On va simuler toutes les exécutions de $\mathcal{M}$.

Puisqu'il peut y avoir une infinité d'exécutions pour une même donnée et que certaines exécutions peuvent ne pas s'arrêter, on va simuler ces exécutions par longueurs croissantes.

Soit K le degré de non déterminisme de $\mathcal{M}$:

$$K = \max\{|\delta(p,x)| \mid (p,x) \in Q \times \Gamma\}$$

Soit $\Sigma_1 = \{1, \ldots, K\}$. Un mot de Σ_1^* représente un calcul de $\mathcal{M}$ en donnant la suite des choix effectués à chaque étape du calcul.

3.5 Simulation

On construit une MT $\mathcal{M}'$ qui énumère sur une bande B_1 les mots de Σ_1^* dans l'ordre hiérarchique et simule les calculs décrits par ces mots.

$\mathcal{M}'$ accepte le même langage que $\mathcal{M}$.
On trouve le plus court calcul qui accepte le mot.

B' bande d'entrée de $\mathcal{M}'$.
B simulation de la bande de $\mathcal{M}$.
B_1 bande d'énumération de Σ_1^*.

3.5 Simulation

var etat : $Q \cup H$
TQ VRAI faire
$B := B'$; $etat := s$;
TQ $R_1 \neq \square$ faire (Simulation d'une transition de $\mathcal{M}$)
Cas (etat,R, R_1) parmi
$\vdots$
/* $\delta(p,x) = \{(q_1,y_1,z_1),\ldots,(q_\ell,y_\ell,z_\ell)\}$ */
p, x, k $(k \leq \ell)$: $etat := q_k$; $W(y_k)$; Move(z_k)
p, x, k $(k > \ell)$: EXIT(TQ)
$\vdots$
FinCas
D_1
FTQ
si etat=OK alors STOP(OK) fsi; effacer(B); reset(B_1); SUCC(B_1)
FTQ

3.5 Décision

Remarque :
$\mathcal{M}'$ ne s'arrête que si $\mathcal{M}$ accepte, pas si $\mathcal{M}$ rejette.
Si $\mathcal{M}$ décide L, il faut modifier $\mathcal{M}'$ pour qu'elle s'arrête toujours.

Lemme : La machine $\mathcal{M}$ a un calcul infini sur le mot u si et seulement si $\forall n > 0$, il existe un calcul de longueur n sur u.

i.e., La machine $\mathcal{M}$ n'a pas de calcul infini sur le mot u si et seulement si il existe $n > 0$ tel que $\mathcal{M}$ n'a pas de calcul sur u de longueur n.

On modifie $\mathcal{M}'$ pour qu'elle teste la condition du lemme.
Si $\mathcal{M}$ décide L alors $\mathcal{M}'$ décide L.

3.5 Décision

var état : $Q \cup H$; var fini, arrêt : booleen; fini := vrai
TQ VRAI faire
    $B := B'$; état := s; arrêt := faux; $\mathcal{O}(|u|)$
    TQ $R_1 \neq \square$ faire $|B_1|$ fois
        Cas (état,R_1, R) parmi
            p, x, k $(k \leq \ell)$: $etat := q_k$; $W(y_k)$; Move(z_k)
            p, x, k $(k > \ell)$: arrêt := vrai; EXIT(TQ)
        FinCas
        D_1
    FTQ
    si état $= \mathrm{OK}$ alors STOP(OK) fsi
    fini := fini $\wedge$ arrêt
    si $(B_1 \in \{K\}^*)$ et fini alors STOP(KO) fsi $\mathcal{O}(|B_1|)$
    fini := fini $\vee$ $(B_1 \in \{K\}^*)$
    effacer(B); reset(B_1); SUCC(B_1) $\mathcal{O}(|u| + |B_1|)$
FTQ

3.5 Complexité

Temps de la simulation du calcul indiqué par B_1 : $\mathcal{O}(|u| + |B_1|)$.

De plus, si $\mathcal{M}$ décide L en temps f alors tous les calculs de $\mathcal{M}$ sur u s'arrêtent après au plus $N \leq f(|u|)$ transitions (ops $N \geq |u|$).

Donc le calcul de $\mathcal{M}'$ est (en ordre de grandeur)

$$|u| + (|u| + 1)K + (|u| + 2)K^2 + \cdots + (|u| + N)K^N$$

$$\leq (|u| + N)\frac{K^{N+1} - 1}{K - 1} \in \mathcal{O}(NK^N)$$

Comme $NK^N \in \mathcal{O}((K+1)^N)$ et que $(K+1)^N = 2^{N \log_2(K+1)}$, le temps de calcul de $\mathcal{M}'$ est en $2^{\mathcal{O}(N)}$.

La MT $\mathcal{M}'$ travaille donc en temps $2^{\mathcal{O}(f)}$.

3.6 Fonctions récursives

Une autre formalisation de la notion de procédure effective :
fonctions calculables sur les entiers naturels.

$$f : \mathbb{N}^k \longrightarrow \mathbb{N}$$

Exemples : somme, produit, puissance, factorielle, Fibonacci, ...

Les fonctions ci-dessus sont totales. On considérera aussi des fonctions partielles comme DIV ou MOINS.

3.6 Fonctions primitives récursives

Fonctions de base :

– Constante 0.

– Successeur $\sigma : \mathbb{N} \longrightarrow \mathbb{N}$

$$n \longmapsto n+1$$

– Projections $\pi_i^k : \mathbb{N}^k \longrightarrow \mathbb{N}$

$$(n_1, \ldots, n_k) \longmapsto n_i$$

Composition de fonctions : $f : \mathbb{N}^k \longrightarrow \mathbb{N}$

$$\bar{n} \longmapsto h(g_1(\bar{n}), \ldots, g_\ell(\bar{n}))$$

avec h d'arité ℓ, $g_1, \ldots, g_\ell$ d'arité k.

3.6 Fonctions primitives récursives

Récursion primitive :

Soient g d'arité k et h d'arité $k+2$. On définit f d'arité $k+1$ par

$$\begin{aligned} f(\bar{n}, 0) &= g(\bar{n}) \\ f(\bar{n}, m+1) &= h(\bar{n}, m, f(\bar{n}, m)) \end{aligned}$$

Définition 3.26 *Les fonctions primitives récursives sont celles obtenues à partir des fonctions de base par compositions et récursions primitives.*

Les fonctions PR sont intuitivement calculables.

Les fonctions PR sont totales.

3.6 Exemples

Constantes $j = \overbrace{\sigma(\sigma(\cdots\sigma(}^{j}0)\cdots))$

Somme

$$\begin{aligned} \mathrm{plus}(n,0) &= n & g &= \pi_1^1 \\ \mathrm{plus}(n,m+1) &= \sigma(\mathrm{plus}(n,m)) & h &= \sigma \circ \pi_3^3 \end{aligned}$$

Prédécesseur (totale)

$$\begin{aligned} \mathrm{Pred}(0) &= 0 & g &= 0 \\ \mathrm{Pred}(m+1) &= m & h &= \pi_1^2 \end{aligned}$$

Différence (totale)

$$\begin{aligned} n \ominus 0 &= n & g &= \pi_1^1 \\ n \ominus (m+1) &= \mathrm{Pred}(n \ominus m) & h &= \mathrm{Pred} \circ \pi_3^3 \end{aligned}$$

Produit

$$\begin{aligned} \mathrm{prod}(n,0) &= 0 & g &= 0 \\ \mathrm{prod}(n,m+1) &= \mathrm{plus}(n,\mathrm{prod}(n,m)) & h &= \mathrm{plus}(\pi_1^3,\pi_3^3) \end{aligned}$$

3.6 Exemples

Factorielle

$$0! = 1 \qquad g = 1$$
$$(m+1)! = (m+1) \times m! \qquad h = \mathrm{prod}(\sigma \circ \pi_1^2, \pi_2^2)$$

Puissance

$$n^0 = 1 \qquad g = 1$$
$$n^{m+1} = n \times n^m \qquad h = \mathrm{prod}(\pi_1^3, \pi_3^3)$$

Double puissance

$$n \uparrow\uparrow 0 = 1 \qquad g = 1$$
$$n \uparrow\uparrow (m+1) = n^{n \uparrow\uparrow m} \qquad h = \mathrm{puiss}(\pi_1^3, \pi_3^3)$$

On a donc

$$n \uparrow\uparrow 1 = n$$
$$n \uparrow\uparrow 2 = n^n$$
$$n \uparrow\uparrow 3 = n^{n^n}$$
$$n \uparrow\uparrow m = \left. n^{n^{\cdot^{\cdot^{\cdot^n}}}} \right\} m$$

3.6 Exemples

k-puissance

$$\begin{aligned} n \uparrow^k 0 &= 1 & g = 1 \\ n \uparrow^k (m+1) &= n \uparrow^{k-1} (n \uparrow^k m) & h = \uparrow^{k-1} (\pi_1^3, \pi_3^3) \end{aligned}$$

Remarque : si on considère k comme un argument, on obtient la formule de récurrence suivante qui n'est pas une récursion primitive :

$$f(k+1, n, m+1) = f(k, n, f(k+1, n, m))$$

De façon similaire, la fonction d'Ackermann est calculable mais non PR.

$$\begin{aligned} \mathrm{Ack}(0, m) &= m+1 \\ \mathrm{Ack}(k+1, 0) &= \mathrm{Ack}(k, 1) \\ \mathrm{Ack}(k+1, m+1) &= \mathrm{Ack}(k, \mathrm{Ack}(k+1, m)) \end{aligned}$$

3.6 Exemples

Somme bornée : Soit f une fonction d'arité $k+1$.

On définit la fonction $g(\bar{n}, m) = \sum_{i=0}^{m} f(\bar{n}, i)$ par

$$\begin{aligned} g(\bar{n}, 0) &= f(\bar{n}, 0) \\ g(\bar{n}, m+1) &= g(\bar{n}, m) + f(\bar{n}, m+1) \end{aligned}$$

Produit borné : Soit f une fonction d'arité $k+1$.

On définit la fonction $g(\bar{n}, m) = \prod_{i=0}^{m} f(\bar{n}, i)$ par

$$\begin{aligned} g(\bar{n}, 0) &= f(\bar{n}, 0) \\ g(\bar{n}, m+1) &= g(\bar{n}, m) \times f(\bar{n}, m+1) \end{aligned}$$

3.6 Fibonacci

Le schéma ci-dessous n'est pas primitif récursif, c'est une récurrence d'ordre 2.

$$\begin{cases} \mathrm{Fib}(0) = \mathrm{Fib}(1) = 1 \\ \mathrm{Fib}(n+1) = \mathrm{Fib}(n) + \mathrm{Fib}(n-1) \end{cases}$$

On peut le modifier en une récurrence d'ordre 1 en définissant simultanément deux fonctions

$$\begin{cases} f(0) = 1 & g(0) = 0 \\ f(n+1) = f(n) + g(n) & g(n+1) = f(n) \end{cases}$$

Mais ce n'est toujours pas un schéma de récursion primitive.
Il s'agit d'une récursion multiple.

Nous verrons en utilisant un codage de $\mathbb{N}^2$ dans $\mathbb{N}$ que la fonction de Fibonacci est bien PR.

3.6 Prédicats primitifs récursifs

Un prédicat p d'arité k est PR si sa fonction caractéristique $\mathbb{N}^k \longrightarrow \{0,1\}$ est PR.

Zéro $(=0)$

$$\begin{aligned} \mathrm{zero}(0) &= 1 \\ \mathrm{zero}(m+1) &= 0 \end{aligned}$$

Positif (>0)

$$\begin{aligned} \mathrm{pos}(0) &= 0 \\ \mathrm{pos}(m+1) &= 1 \end{aligned}$$

Comparaisons

$$\begin{aligned} n < m &= \mathrm{pos}(m \ominus n) \\ n \leq m &= \mathrm{zero}(n \ominus m) \\ n = m &= \mathrm{zero}(n \ominus m) \times \mathrm{zero}(m \ominus n) \end{aligned}$$

3.6 Prédicats primitifs récursifs

Opérateurs booléens

$$\begin{aligned} p(\bar{n}) \wedge q(\bar{n}) &= p(\bar{n}) \times q(\bar{n}) \\ p(\bar{n}) \vee q(\bar{n}) &= \mathrm{pos}(p(\bar{n}) + q(\bar{n})) \\ \neg p(\bar{n}) &= 1 \ominus p(\bar{n}) \end{aligned}$$

Quantifications bornées :

Soit p un prédicat d'arité $k+1$. On définit les prédicats

$$q(\bar{n}, m) = \forall i \leq m, p(\bar{n}, i) = \prod_{i=0}^{m} p(\bar{n}, i)$$

$$e(\bar{n}, m) = \exists i \leq m, p(\bar{n}, i) = \mathrm{pos}\left(\sum_{i=0}^{m} p(\bar{n}, i) \right)$$

3.6 Définition par cas

Soient $g_1, \ldots, g_\ell, g_{\ell+1}$ des fonctions d'arité k et $p_1, \ldots, p_\ell$ des prédicats d'arité k.

La fonction $f(\bar{n}) = \begin{cases} g_1(\bar{n}) & \text{si } p_1(\bar{n}) \\ \vdots & \\ g_\ell(\bar{n}) & \text{si } p_\ell(\bar{n}) \\ g_{\ell+1}(\bar{n}) & \text{sinon} \end{cases}$

peut se définir par $f(\bar{n}) = \sum_{i=1}^{\ell+1} q_i(\bar{n}) \times g_i(\bar{n})$

avec
$$\begin{aligned} q_1 &= p_1 \\ q_i &= p_i \wedge \neg(p_1 \vee \cdots \vee p_{i-1}) \\ q_{\ell+1} &= \neg(p_1 \vee \cdots \vee p_\ell) \end{aligned}$$

Si les prédicats ne sont pas disjoints, le résultat est celui correspondant au premier prédicat satisfait.

3.6 Exemple : mod et div

La fonction $\mathrm{mod} : \mathbb{N}^2 \longrightarrow \mathbb{N}$ est PR.

$$\begin{aligned} 0 \bmod n &= 0 \\ (m+1) \bmod n &= \begin{cases} 0 \text{ si } (m \bmod n) + 1 = n \\ (m \bmod n) + 1 \text{ sinon} \end{cases} \end{aligned}$$

Il s'agit d'un schéma PR utilisant la fonction h définie par cas :

$$h(n, m, k) = \begin{cases} 0 & \text{si } k + 1 = n \\ k + 1 & \text{sinon} \end{cases}$$

La fonction $\mathrm{div} : \mathbb{N}^2 \longrightarrow \mathbb{N}$ est PR.

$$\begin{aligned} 0 \operatorname{div} n &= 0 \\ (m+1) \operatorname{div} n &= (m \operatorname{div} n) + \mathrm{zero}((m+1) \bmod n) \end{aligned}$$

3.6 Exemple : mod et div

Remarque : Avec ces définitions, on a $\forall m \in \mathbb{N}$,

$$\begin{aligned} m \bmod 0 &= m \\ m \operatorname{div} 0 &= 0 \end{aligned}$$

Exercice :

1) Montrer que $\max : \mathbb{N}^2 \longrightarrow \mathbb{N}$ est PR.

2) Soit $f : \mathbb{N}^2 \longrightarrow \mathbb{N}$ une fonction PR.

Montrer que $g : \mathbb{N} \longrightarrow \mathbb{N}$ définie par $g(k) = \max_{m+n=k} f(m,n)$ est PR.

3.6 Minimisation bornée

Soit p un prédicat d'arité $k+1$. On veut définir la fonction

$$\begin{array}{rcl} \mathbb{N}^k & \longrightarrow & \mathbb{N} \\ \bar{n} & \longmapsto & \min\{i \in \mathbb{N} \mid p(\bar{n}, i)\} \end{array}$$

La fonction est partielle, donc elle n'est pas PR.
On va donc restreindre cette minimisation.

3.6 Minimisation bornée

On veut calculer $f(\bar{n}, m) = \min\{i < m \mid p(\bar{n}, i)\}$.
Le résultat sera m si un tel entier n'existe pas.

On a

$$f(\bar{n}, 0) = 0$$

$$f(\bar{n}, m+1) = \begin{cases} f(\bar{n}, m) & \text{si } \exists i \leq m, p(\bar{n}, i) \\ m+1 & \text{sinon} \end{cases}$$

ou

$$f(\bar{n}, 0) = 0$$

$$f(\bar{n}, m+1) = \begin{cases} f(\bar{n}, m) & \text{si } p(\bar{n}, f(\bar{n}, m)) \\ m+1 & \text{sinon} \end{cases}$$

Dans la suite, on utilisera la notation $\mu i < m, p(\bar{n}, i)$.

3.6 Représentation binaire d'un entier

$\mathrm{bin}(n)$: représentation binaire normalisée de n.
$|n| = |\mathrm{bin}(n)|$: longueur de la représentation de n.

n	0	1	2	3	4	5	6	7	8
$\mathrm{bin}(n)$	ε	1	10	11	100	101	110	111	1000
$\|n\|$	0	1	2	2	3	3	3	3	4

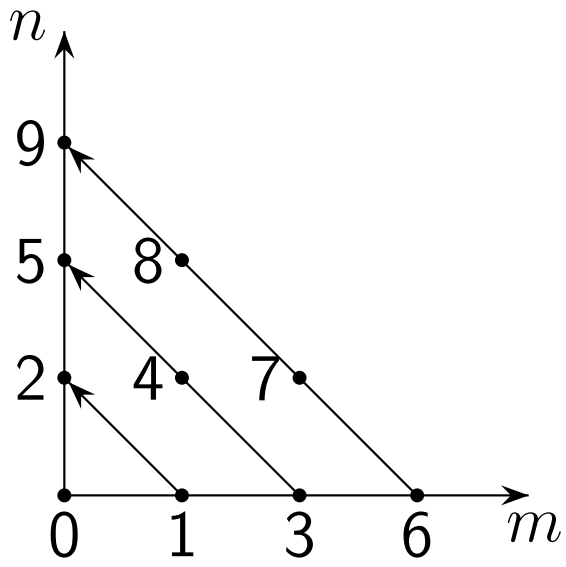
La fonction $n \longmapsto |n|$ est PR.

$$|n| = \lceil \log_2(n+1) \rceil = \mu k \leq n, (n+1) \leq 2^k$$

La fonction $(n, i) \longmapsto c(n, i)$ qui associe le i-ème chiffre de $\mathrm{bin}(n)$ est PR (retourne 0 si $i > |\mathrm{bin}(n)|$).

$$c(n, i) = (n \operatorname{div} 2^i) \bmod 2$$

3.6 codage de $\mathbb{N}^2$ dans $\mathbb{N}$



La bijection couple : $\mathbb{N}^2 \longrightarrow \mathbb{N}$ est PR.

$$(m,n) \longmapsto \frac{(m+n)(m+n+1)}{2} + n$$

Les fonctions c_1 et c_2 qui définissent la bijection inverse sont PR

$$(c_1, c_2) : \mathbb{N} \longrightarrow \mathbb{N}^2$$

$$k \longmapsto (c_1(k), c_2(k))$$

Exercice : Généraliser au codage de $\mathbb{N}^k$ dans $\mathbb{N}$.

3.6 Fibonacci

Application : La fonction de Fibonacci est PR.

On considère (avec la convention $\mathrm{Fib}(-1) = 0$) :

$$\begin{array}{rccl} h: & \mathbb{N} & \longrightarrow & \mathbb{N} \\ & n & \longmapsto & \mathrm{couple}(\mathrm{Fib}(n), \mathrm{Fib}(n-1)) \end{array}$$

La fonction h est PR :

$$\begin{array}{rcl} h(0) & = & \mathrm{couple}(1, 0) \\ h(n+1) & = & \mathrm{couple}(c_1(h(n)) + c_2(h(n)), c_1(h(n))) \end{array}$$

Donc, la fonction de Fibonacci est bien PR :

$$\mathrm{Fib}(n) = c_1(h(n))$$

3.6 Récursion multiple

Exercice : Montrer que les fonctions définies par un schéma de récursion multiple sont PR.

$$\begin{cases} f_1(\bar{n}, 0) &= g_1(\bar{n}) \\ \quad\vdots & \\ f_\ell(\bar{n}, 0) &= g_\ell(\bar{n}) \end{cases}$$

$$\begin{cases} f_1(\bar{n}, m+1) &= h_1(\bar{n}, m, f_1(\bar{n}, m), \ldots, f_\ell(\bar{n}, m)) \\ \quad\vdots & \\ f_\ell(\bar{n}, m+1) &= h_\ell(\bar{n}, m, f_1(\bar{n}, m), \ldots, f_\ell(\bar{n}, m)) \end{cases}$$

3.6 Récursion d'ordre ℓ

Exercice : Montrer qu'une fonction définie par une relation de récurrence d'ordre ℓ est PR.

$$\begin{cases} f(\bar{n}, 0) & = & g_0(\bar{n}) \\ \vdots & & \\ f(\bar{n}, \ell - 1) & = & g_{\ell-1}(\bar{n}) \\ f(\bar{n}, m + \ell) & = & h(\bar{n}, m, f(\bar{n}, m), \ldots, f(\bar{n}, m + \ell - 1)) \end{cases}$$

3.6 MT et fonctions PR

Théorème 3.27 *Toute fonction PR est calculable par une MT en temps PR, i.e. si* $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ *est une fonction PR alors il existe une MT* $\mathcal{M}$ *avec* k *bandes d'entrée et 1 bande de sortie telle que*

$$\mathcal{M}(\mathrm{bin}(\bar{n})) = \mathrm{bin}(f(\bar{n}))$$

De plus, $\mathcal{M}$ *travaille en temps* C *qui est une fonction PR.*

preuve : par induction sur la structure des fonctions PR.

3.6 Codage de Gödel

Soit $\Gamma = \{0, \ldots, k-1\}$ un alphabet de taille k.
On considère le codage

$$\begin{array}{rccc} \text{Gödel} : & \Gamma^* & \longrightarrow & \mathbb{N} \\ & u_0 \cdots u_n & \longmapsto & \sum_{i=0}^{n} u_i \cdot k^i \end{array}$$

Remarque : le mot vide est codé 0.

Cette application est surjective mais pas injective :

$$\text{Gödel}(12) = \text{Gödel}(120) = \text{Gödel}(1200)$$

La fonction Gödel devient une bijection si on se restreint aux mots ne se terminant pas par 0 ($\square$).

3.6 MT et fonctions PR

Théorème 3.28 *Toute fonction calculable par une MT en temps PR est une fonction PR.*

si $\mathcal{M}$ est une MT qui travaille en temps C (C étant une fonction PR), alors il existe une fonction PR $f : \mathbb{N} \longrightarrow \mathbb{N}$ telle que pour tout mot $w \in \Sigma^*$,

$$f(\text{Gödel}(w)) = \text{Gödel}(\mathcal{M}(w))$$

Tout langage décidable par une MT en temps PR est un prédicat PR.

Toutes les fonctions calculables en pratique sont des fonctions PR.

3.6 Fonctions calculables mais non PR

Proposition 3.29 *Il y a des fonctions calculables qui ne sont pas PR.*

Preuve: On note $f_0, f_1, \ldots$ la suite des fonctions PR.

On considère la fonction $g : \mathbb{N} \longrightarrow \mathbb{N}$

$$n \longmapsto f_n(n, 0, \ldots, 0) + 1$$

g est calculable mais pas PR. □

La fonction d'Ackermann est calculable mais pas PR.

Elle croît plus vite que n'importe quelle fonction PR.

$$\begin{aligned} \mathrm{Ack}(0, m) &= m + 1 \\ \mathrm{Ack}(k+1, 0) &= \mathrm{Ack}(k, 1) \\ \mathrm{Ack}(k+1, m+1) &= \mathrm{Ack}(k, \mathrm{Ack}(k+1, m)) \end{aligned}$$

3.6 Minimisation non bornée

Soit p un prédicat d'arité $k+1$, on définit la fonction partielle :

$$\begin{array}{rccl} f: & \mathbb{N}^k & \longrightarrow & \mathbb{N} \\ & \bar{n} & \longmapsto & \min\{i \in \mathbb{N} \mid p(\bar{n}, i)\} \end{array}$$

Notation $f(\bar{n}) = \mu i.p(\bar{n}, i)$.

– Un prédicat est sûr si $\forall \bar{n}, \exists i, p(\bar{n}, i)$.
– Si p est sûr, alors $\mu i.p(\bar{n}, i)$ est totale.

On ne sait pas décider si un prédicat est sûr.

3.6 Fonctions récursives générales

Définition 3.30
Une fonction est récursive si elle peut s'obtenir à partir des fonctions de base par composition, récursion primitive et minimisation de prédicats sûrs.

Une fonction est récursive partielle si elle peut s'obtenir à partir des fonctions de base par composition, récursion primitive et minimisation de prédicats arbitraires.

– Les fonctions récursives sont totales.
– La récursion primitive est inutile.

On ne sait pas décider si un mot représente une fonction récursive.

Exercice : Montrer que la fonction d'Ackermann est récursive.

3.6 Fonctions récursives générales

Théorème 3.31 *Une fonction est récursive si et seulement si elle est calculable par une MT.*

Théorème 3.32 *Une fonction f est récursive partielle ssi elle est (semi-)calculable par une MT $\mathcal{M}$ qui peut ne pas s'arrêter.*
$\mathcal{M}$ s'arrête sur une donnée $\bar{n}$ si et seulement si f est définie en $\bar{n}$.

La preuve d'existence de fonctions calculables non PR ne s'applique plus.
Soit $f_0, f_1, \ldots$ la suite des fonctions récursives.
la fonction $g(n) = f_n(n, 0, \ldots, 0) + 1$ n'est pas calculable.

Thèse de Church

Un problème est effectivement décidable (i.e. par une procédure effective) si et seulement si il est décidable par une machine de Turing.

Une fonction est effectivement calculable (i.e. par une procédure effective) si et seulement si elle est calculable par une machine de Turing.

À l'appui de cette thèse :

- Machine de Turing = Machine RAM = fonction récursive
- On n'a jamais pu faire mieux.

3.7 Langages R et RE

$\mathbf{R}$ classe des langages récursifs (décidable, calculable)

$\mathbf{RE}$ classe des langages récursivement énumérables (semi-décidables)

$\mathbf{co-RE}$ complémentaires des langages $\mathbf{RE}$

Proposition 3.33

1. $L \in \mathbf{R} \Longrightarrow \overline{L} \in \mathbf{R}$,
2. $L \in \mathbf{R} \Longleftrightarrow L \in \mathbf{RE} \cap \mathbf{co-RE}$,
3. *Il y a des langages non* $\mathbf{R}$, *non* $\mathbf{RE}$, *non* $\mathbf{co-RE}$.

3.7 Codage d'une MT

Soit $M = (Q, H, s, \Sigma, \Gamma, \delta)$ une MT déterministe telle que
– $Q \cup H = \{0, 1, \ldots, n-1\}$ avec $s = 0$, $\mathrm{OK} = 1$ et $\mathrm{KO} = 2$.
– $\Sigma = \{0, \ldots, \ell - 1\}$ et $\Gamma = \{0, \ldots, m-1\}$ $(\ell < m)$.

On note $c(q) \in \{0,1\}^{\lceil \log_2 n \rceil}$ le codage binaire d'un état $q \in Q \cup H$.
On note $c(a) \in \{0,1\}^{\lceil \log_2 m \rceil}$ le codage binaire d'une lettre $a \in \Gamma$.
On note $b(k) \in \{\varepsilon\} \cup 1 \cdot \{0,1\}^*$ le codage binaire d'un entier $k \in \mathbb{N}$.

Le codage d'une direction est $c(\leftarrow) = 0$ et $c(\rightarrow) = 1$.
Le codage de $(p, x, q, y, z) \in \delta$ est $c(p)\#c(x)\#c(q)\#c(y)\#c(z)$.
Le codage de $\delta = \{t_1, t_2, \ldots\}$ est $c(\delta) = c(t_1)\#c(t_2)\#\cdots$.
Le codage de la MT M est $c(M) = b(|Q \cup H|)\#b(|\Sigma|)\#b(|\Gamma|)\#c(\delta)$.

Le codage de $w = a_1 a_2 \cdots \in \Sigma^*$ est $c(w) = c(a_1)c(a_2)\cdots$

3.7 Machine universelle

Proposition 3.34

On peut construire une MT universelle $\mathcal{U}$ qui accepte

$$\mathcal{LU} = \{c(M)\#\#c(w) \mid M \text{ accepte } w\}$$

La MT $\mathcal{U}$ a pour alphabets $\Sigma_{\mathcal{U}} = \{0, 1, \#\}$ et $\Gamma_{\mathcal{U}} = \{0, 1, \#, \square\}$.
Elle vérifie que sa donnée est syntaxiquement correcte,
i.e. de la forme $c(M)\#\#c(w)$,
puis simule le calcul de M sur w et accepte si M accepte w.

3.7 Problèmes non R, non RE

On considère le langage sur l'alphabet $\{0, 1, \#\}$

$$L_0 = \{c(M) \mid M \text{ MT avec } |\Sigma_M| \geq 3 \text{ et } M \text{ n'accepte pas } c(M)\}.$$

Proposition 3.35

1. $L_0 \in \mathbf{co-RE} \setminus \mathbf{RE}$.
2. $\mathcal{LU} \in \mathbf{RE} \setminus \mathbf{R}$.

Exercice : Montrer que $L_1 \in \mathbf{co-RE} \setminus \mathbf{RE}$.

$$L_1 = \{c(M) \mid M \text{ MT avec } |\Sigma_M| \geq 3 \text{ et } M \text{ ne s'arrête pas sur } c(M)\}.$$

3.7 Réduction

Définition 3.36

Un problème $K_1 \in \Sigma_1^*$ se réduit à un problème $K_2 \subseteq \Sigma_2^*$ s'il existe une fonction calculable (par une MT) $f : \Sigma_1^* \to \Sigma_2^*$ telle que

$$\forall x \in \Sigma_1^*, \quad x \in K_1 \Longleftrightarrow f(x) \in K_2.$$

Applications : Si K_1 se réduit à K_2, alors

$$\begin{aligned} K_2 \in \mathbf{R} &\implies K_1 \in \mathbf{R} \\ K_2 \in \mathbf{RE} &\implies K_1 \in \mathbf{RE} \\ K_1 \notin \mathbf{R} &\implies K_2 \notin \mathbf{R} \\ K_1 \notin \mathbf{RE} &\implies K_2 \notin \mathbf{RE} \end{aligned}$$

3.7 Problème de l'arrêt

On considère les langages

$$\begin{aligned} H &= \{c(M)\#\#c(w) \mid M \text{ s'arrête sur } w\} \\ H_\varepsilon &= \{c(M) \mid M \text{ s'arrête sur la bande vide}\} \\ H_\exists &= \{c(M) \mid M \text{ s'arrête sur au moins une donnée}\} \\ H_\forall &= \{c(M) \mid M \text{ s'arrête sur toutes ses données}\} \end{aligned}$$

Proposition 3.37

1. $H, H_\varepsilon, H_\exists \in \mathbf{RE} \setminus \mathbf{R}$.
2. $H_\forall \notin \mathbf{RE} \cup \mathbf{co{-}RE}$.
3. *Tout langage* $\mathbf{RE}$ *se réduit à* H.

3.7 Théorème de RICE

Théorème 3.38 (RICE)
Toute propriété non triviale des langages $\mathbf{RE}$ *est indécidable.*

Une propriété des langages $\mathbf{RE}$ est définie par une partie $\mathcal{C}$ de $\mathbf{RE}$.

La propriété est non triviale si $\emptyset \neq \mathcal{C} \neq \mathbf{RE}$.

On note $\mathrm{RICE}(\mathcal{C}) = \{c(M) \mid \mathcal{L}(M) \in \mathcal{C}\}$.

Théorème 3.39 (RICE)
Si $\mathcal{C}$ *est non triviale alors* $\mathrm{RICE}(\mathcal{C})$ *est indécidable.*

Exemple : $\mathrm{PAIR} = \{c(M) \mid \mathcal{L}(M) = (\Sigma_M^2)^*\} \notin \mathbf{RE} \cup \mathbf{co{-}RE}$.

3.7 Pavage

Soit C un ensemble de couleurs contenant la couleur blanche.

Une pièce est un quadruplet $(c(g), c(h), c(d), c(b)) \in C^4$.

Un pavage du quart de plan utilisant un ensemble de couleurs $P \subseteq C^4$ est une application $f : \mathbb{N}^2 \to P$ telle que $\forall m, n \in \mathbb{N}$,
– $f(0, n)(g) = f(m, 0)(b) = \text{blanc}$,
– $f(m, n)(d) = f(m + 1, n)(g)$ et $f(m, n)(h) = f(m, n + 1)(b)$.

Problème du pavage :
Données : Un ensemble fini P de pièces.
Question : Existe-t-il un pavage du quart de plan avec les pièces de P ?

Théorème 3.40 *Le problème du pavage est* $\mathbf{co-RE}$ *mais pas* $\mathbf{RE}$.