Modéles de Calcul - partie MT

Michaël Périn

Verimag – Polytech' Grenoble – Université de Grenoble-Alpes

Préambule

Pour bâtir ce cours je me suis principalement inspiré des ouvrages

— Pierre Wolper. *Introduction à la calculabilité*. Dunod, 2006. 3ème édition

— H. Common-Lundh, P. Schnoebelen, S. Haddad, P-A. Reynier, F-R. Sinot, and C. Sirangelo. *Note de cours de Calculabilité et Logique*. Polycopié en ligne de l'ENS Cachan, 2009

Le premier prend le temps d'expliquer pourquoi en informatique on s'intéresse à la question de décidabilité. Le second démontre les résutlats du cours et apporte des compléments.

Je me suis efforcé de réduire au maximum les notations, de détailler les preuves et de les mener de la manière aussi simple que possible en mettant en jeu le minimum de concepts.

Ce cours contient les principaux résultats sur les machines de Turing, les réductions, les problèmes indécidables classiques et le théorême de Rice. Bref, ce qu'il faut de culture pour **transformer un programmeur en informaticien**.

Pour les néophytes, le meilleur ouvrage pour aborder les questions de décidabilité est sans doute la BD

— Apóstolos K. Doxiàdis, Christos Papadimitriou, Alecos Papadatos, and Annie Di Donna. *Logicomix*. Vuibert, BD 2010. 352 pages.

Pour se distraire après avoir relu votre cours, je vous conseille deux films :

— Morten Tyldum. «The Imitation Game», DVD 2014. Adaptation cinématographique de la biographie «Alan Turing ou l'énigme de l'intelligence» d'Andrew Hodges

— Gabe Ibáñez. «Autómata», DVD 2014. Une reflexion sur les capacités des machines

Chapitre 1

Machine de Turing

1.1 Rappels du module INF232 «Langages & Automates»

1.1.1 Langage

— Un langage L est un ensemble de mots écrits sur un alphabet donné, Σ.

— Σ^* représente l'ensemble de tous les mots possibles écrits avec des symboles de Σ.

— Tout langage L sur Σ est nécessairement inclus dans Σ^*

Langage = Ensemble de mots écrits sur un alphabet Σ

1.1.2 Automates à nombre d'états finis et langages réguliers

1. langage = ensemble de mots sur un alphabet Σ
2. langages réguliers = $\mathscr{L}$(AEF déterministe) = $\mathscr{L}$(AEF non-déterministe)
   clos par union, intersection, complémentaire, concaténation $L_1 \cdot L_2$, l'étoile (L^*)
3. $\{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas régulier se prouve grâce au lemme de l'itération

1.2 Automates à une pile et langages algébriques

Langage algébrique = Langage reconnu par un automate à une pile (AUP).

Exercice 1

Donnez un AUP déterministe qui reconnaît $\{a^n b^n \mid n \in \mathbb{N}\}$

Exercice 2

Donnez un AUP déterministe qui reconnaît les palindromes sur $\Sigma = \{a, b\}$ dont le milieu est marqué par un $\bullet$.

Exercice 3

Donnez un AUP non-déterministe qui reconnaît les palindromes sur $\Sigma = \{a, b\}$

Remarque Il n'existe pas d'AUP déterministe qui reconnaît les palindromes.
Donc $\mathscr{L}$(AUP *déterministe*) $\subset$ $\mathscr{L}$(AUP *non-déterministe*)
Ils existent des langages non-algébriques, *ie.* pas reconnaissables par un AUP. Par exemple $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.
On le montre au moyen d'un équivalent du lemme de l'itération pour les AUP.

1.3 Machine de Turing

Les machines de Turing sont une extension des automates à une pile. Voici les deux différences :

1. Alors qu'un aup a le droit d'écrire et de lire **uniquement** le sommet de la pile, une mt peut **déplacer à sa guise sa tête de lecture/écriture** dans la pile – qu'on appelle donc plus une pile mais **un ruban**.
2. La seconde différence est que **le mot à reconnaître est inscrit sur le ruban**.

Les différences entre aup et mt semblent mineures, pourtant ...

Ces extensions suffisent à donner aux mt la puissance du calculable.

Encore plus surprenant ...

L'ajout d'une seconde pile aux aup suffit à leur donner la puissance du calculable.

Preuve : On le montre expliquant comment simuler le fonctionnement d'une mt par un aup à deux piles, cf. exercice de td. □

1.3.1 Représentation d'une machine de Turing

On considére des machines de Turing opérant sur **l'alphabet** $\Sigma = \{\$, \square, 0, 1\}$. Les symboles 0,1 servent à coder les données, \$ le **marqueur de début de la donnée d'entrée** et $\square$ dénote une **case blanche** du ruban. Un **ruban vierge** est une succession infinie de cases blanches : $\overline{\infty\square \mid \square \mid \square \mid \square \mid \square\infty}$

Les **transitions des** mt sont de la forme

- $\mathbf{q} \xrightarrow{\ell/e} \mathbf{q}'$: dans l'état $\mathbf{q}$, si on lit ℓ, on écrit e sur le ruban et on passe dans l'état $\mathbf{q}'$
- $\mathbf{q} \xrightarrow{\ell:d} \mathbf{q}'$: dans l'état $\mathbf{q}$, si on lit ℓ, on effectue le déplacement d de la tête de lecture/écriture et on passe dans l'état $\mathbf{q}'$. Les déplacements possibles sont $\{L, H, R\}$ pour *Left,Here,Right*
- $\mathbf{q} \xrightarrow{\ell/e:d} \mathbf{q}'$: dans l'état $\mathbf{q}$, si on lit ℓ, on écrit e sur le ruban avant d'effectuer le déplacement d de la tête, puis on passe dans l'état l'état $\mathbf{q}'$

Parmi les **états d'une** mt on distingue un état initial et possiblement un état accepteur, un état rejet et des états terminaux : on utilise les mêmes notations que pour les automates à nombre d'états finis.

- $\searrow(\mathbf{q})$ désigne $\mathbf{q}$ comme **état initial**
- $\textcircled{\not\to}$ est **l'état accepteur**
- $\otimes\not\to$ est **l'état rejet**
- un état $\mathbf{q}$ **est terminal** $\iff$ $\mathbf{q}$ n'a **pas de transition sortante**, ce que l'on note $\mathbf{q}\not\to$

1.3.2 Notations mathématiques d'une mt

Pour définir précisement une mt M il faut indiquer

- Σ, l'alphabet sur lequel elle opère
- $\mathscr{Q}$, l'ensemble des états utilisés dans les transitions. Par exemple, $\mathscr{Q} = \{\mathbf{q}_0, \mathbf{q}_1, \dots\}$
- $\searrow(\mathbf{q})$, son état initial
- $\delta : \mathscr{Q} \times \Sigma \to \Sigma \times \{L, H, R\} \times \mathscr{Q}$, sa fonction de transition.
  δ associe à un état $\mathbf{q}$ et un symbole lu ℓ, un triplet $(e, d, \mathbf{q}')$ qui représente le symbole à écrire, le déplacement de la tête et le nouvel état.
- $\mathcal{A} \subseteq \mathscr{Q}$, l'ensemble des états accepteurs (qui peut être vide)

— $\mathcal{R} \subseteq \mathcal{Q}$, l'ensemble des états de rejet (qui peut être vide)

On définit une MT en donnant $M = (\Sigma, \mathcal{Q}, \textcircled{q}, \delta, \mathcal{A}, \mathcal{R})$

Définition 1 (Transitions) *Il existe plusieurs façons de noter les transitions d'une* MT

$\delta(\mathbf{q}, \ell) = (e, d, \mathbf{q}')$ ou bien $\textcircled{\mathbf{q}} \xrightarrow{\ell/e:d} \textcircled{\mathbf{q}'}$

1.3.3 Conventions adoptées en MCAL : machine de Turing à demi-ruban

Il existe de nombreuses variantes – toutes équivalentes – des machines de Turing. Dans ce cours, on adopte les conventions suivantes :

(C1) On considére uniquement des MT **déterministes**, c'est-à-dire qu'aucun état $\mathbf{q}$ ne peut avoir deux transitions différentes sur le même symbole lu.

Exemples :

- $(\mathbf{q}, \ell : H, \mathbf{q}')$ et $(\mathbf{q}, \ell : R, \mathbf{q}')$ n'est pas autorisé : deux actions différentes pour $\delta(\mathbf{q}, \ell)$.
- $(\mathbf{q}, \ell : H, \mathbf{q}')$ et $(\mathbf{q}, \ell : H, \mathbf{q}'')$ n'est pas autorisé : deux états d'arrivée différents $\delta(\mathbf{q}, \ell)$.

(C2) Au delà d'un blanc vers la gauche et au delà d'un blanc vers la droite, il n'y que des blancs. Autrement dit **le ruban ne doit jamais avoir la forme suivante**

$^\infty\square$	ω_1	$\square$	ω_2	$\square^\infty$

(C3) Le **symbole \$ marque le début de l'entrée**. Ainsi, au départ et à la fin de l'exécution d'une MT le ruban sera de la forme

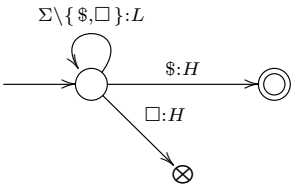
$^\infty\square$	\$	*un mot binaire*	$\square^\infty$

1.3.4 Exemples de machines de Turing

Exercice 4

Donnez une machine de Turing $M_{\overleftarrow{\$}}$ qui recherche le symbole de départ \$ vers la gauche et termine dans un état ◎ si elle a réussi et dans un état ⊗ sinon. Contrairement aux autres MT, celle-ci ne démarre pas sur le symbole \$. Que se passe t'il si la MT démarre sur un $\square$?

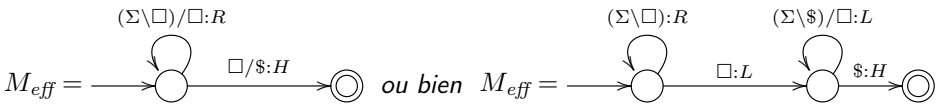
SOLUTION



Exercice 5

Donnez une machine de Turing M_{eff} qui efface le ruban sauf le symbole \$. On suppose que M_{eff} commence sur le symbole \$ du ruban.

SOLUTION



1.4 Exécution d'une machine de Turing

Considérons une mt $M = (\Sigma, \mathscr{Q}, \textcircled{$\mathbf{q}_i$}, \delta, \mathcal{A}, \mathcal{R})$

Définition 2 *Une **configuration** $(\omega_1, \mathbf{q}, \ell.\omega_2)$ contient les informations nécessaires pour exécuter la prochaine transition de la machine de Turing M :*

- *son état courant :* $\mathbf{q}$
- *le contenu du ruban :* $\omega_1.\ell.\omega_2$ *désigne la partie intéressante du ruban*

$^\infty\square$	ω_1	ℓ	ω_2	$\square^\infty$

- ω_1 *désigne la partie du ruban situé à gauche de la tête*
- ℓ *est le symbole courant que lit la tête de lecture/écriture*
- ω_2 *désigne la partie du ruban situé à droite de la tête*

Définition 3 (Effet des différentes transitions sur une configuration) *Le symbole* $\boxed{.}$ *indique l'emplacement de la tête de lecture/écriture sur le ruban.*

1. $(\omega_1, \mathbf{q}, \ell.\omega_2) \xrightarrow{\ell/e} (\omega_1, \mathbf{q}', e.\omega_2)$
   $\omega_1.\boxed{\ell}.\omega_2 \qquad \omega_1.\boxed{e}.\omega_2$
2. $(\omega_1, \mathbf{q}, \ell.\omega_2) \xrightarrow{\ell:H} (\omega_1, \mathbf{q}', \ell.\omega_2)$
   $\omega_1.\boxed{\ell}.\omega_2 \qquad \omega_1.\boxed{\ell}.\omega_2$
3. $(\omega_1, \mathbf{q}, \ell_1.\ell_2.\omega_2) \xrightarrow{\ell_1:R} (\omega_1.\ell_1, \mathbf{q}', \ell_2.\omega_2)$
   $\omega_1.\boxed{\ell_1}.\ell_2.\omega_2 \qquad \omega_1.\ell_1.\boxed{\ell_2}.\omega_2$
4. $(\omega_1.\ell_1, \mathbf{q}, \ell_2.\omega_2) \xrightarrow{\ell_2:L} (\omega_1, \mathbf{q}', \ell_1.\ell_2.\omega_2)$
   $\omega_1.\ell_1.\boxed{\ell_2}.\omega_2 \qquad \omega_1.\boxed{\ell_1}.\ell_2.\omega_2$

Définition 4 (Configuration initiale) *Pour exécuter une machine de Turing M sur une **donnée d'entrée** ω*

***CAS 1 : à partir d'un ruban vierge** ie. qui ne contient que des blancs $\square$.*

1. *l'utilisateur insère une marque de début de donnée $ sur le ruban*
2. *Il inscrit ω sur le ruban à la suite du symbole $*
3. *Il place la tête de lecture/écriture sur le symbole $*
4. *La mt M démarre dans son état initial $\mathbf{q}_i$*

Le ruban a l'aspect

$^\infty\square$	$	ω	$\square$	$\square^\infty$

CAS 2 : la donnée ω a été inscrite sur le ruban par une mt précédente.

1. *La mt précédente doit avoir positionné la tête de lecture/écriture sur le symbole qui précède ω et avoir inséré une marque de début de donnée $.*
2. *La mt M démarre dans son état initial $\mathbf{q}_i$*

Le ruban a l'aspect

$^\infty\square$	...	$	ω	$\square$	$\square^\infty$

Dans les deux cas la partie du ruban située à gauche du symbole $ n'a pas d'importance puisqu'elle ne sera pas explorée.

La configuration initiale est donc toujours de la forme $(\ldots, \mathbf{q}_i, \$.\omega)$

Définition 5 (Configuration terminale) *Une **configuration** $(\omega, \mathbf{q}, \omega')$ est **terminale** si la machine de Turing M n'a pas de transition sortante dans l'état $\mathbf{q}$, ce qu'on indique par $\textcircled{q}\not\rightarrow$. L'exécution de M est terminée puisque M ne peut plus effectuer de transition. C'est pourquoi la configuration est dite terminale.*

Définition 6 (Exécution) *L'exécution d'une machine de Turing M sur une donnée ω est une séquence de configurations $(\omega_0, \mathbf{q}_0, \omega'_0) \xrightarrow[M]{} (\omega_1, \mathbf{q}_1, \omega'_1) \xrightarrow[M]{} \ldots$, commençant dans la configuration initiale $(\ldots, \overset{\searrow}{\textcircled{q}}, \$.\omega)$ et passant d'une configuration à la suivante par une transition de M. L'exécution peut s'arrêter sur une configuration terminale ou bien être infinie.*

Définition 7 (Résultat d'une machine de Turing) *Pour qu'une machine de Turing rende un résultat il faut que l'exécution **s'arrête** c'est-à-dire qu'elle atteigne une **configuration terminale**. On distingue 4 cas :*

1. $M(\omega) = \mathbb{V}(\omega')$ *si* $(\ldots, \mathbf{q}_i, \$.\omega) \xrightarrow[M]{}{}^*(\ldots, ◎, \omega')$ *et* $◎ \not\to$

   Interprétation Le booléen $\mathbb{V}$ indique que l'exécution s'est bien déroulé : c'est l'équivalent du `return 0;` à la fin du `main` en langage C. Autrement dit, la MT M accepte l'entrée ω et le résultat du calcul est le mot ω' situé à droite de la tête de lecture.

2. $M(\omega) = \mathbb{F}(\epsilon)$ *si* $(\ldots, \mathbf{q}_i, \$.\omega) \xrightarrow[M]{}{}^*(\ldots, \otimes, \omega')$ *et* $\otimes \not\to$

   Interprétation Le booléen $\mathbb{F}$ indique que l'exécution s'est arrêté dans un état rejet : c'est l'équivalent d'un signal d'erreur à la fin d'un programme C. Autrement dit, la MT M rejette l'entrée ω et il ne faut pas considéré le mot ω' inscrit sur le ruban comme le résultat du calcul.

3. $M(\omega) = ?$ *si et seulement si* $(\ldots, \mathbf{q}_i, \$.\omega) \xrightarrow[M]{}{}^\infty$ *ie. que l'exécution se poursuit indéfiniment.*

   Notation On notera simplement $M(\omega) = \mathbb{V}$ et $M(\omega) = \mathbb{F}$ lorsque le résultat du calcul n'a pas d'importance et qu'on s'intéresse uniquement à l'état terminal (◎ ou $\otimes$) de l'exécution.

1.5 Langage associé à une machine de Turing

Les AEF et AUP s'arrêtent forcément lorsqu'ils ont consommés toutes les lettres du mot à reconnaître. Contrairement aux automates AEF et AUP, l'exécution d'un MT peut ne pas terminer ; le langage reconnu par une MT ne concerne que les exécutions qui *terminent*.

Définition 8 *On note $\mathscr{L}(M)$ le **langage reconnu à une machine de Turing** M*

$$\mathscr{L}(M) = \{\omega \in \Sigma^* \mid M(\omega) = \mathbb{V}\}$$

*C'est l'ensemble des mots ω pour lesquels l'exécution de M **s'arrête** dans un état accepteur $◎ \not\to$.*

Exercice 6 ***Non-terminaison***

Donnez une machine qui ne termine pas pour ϵ et qui termine pour tous les autres mots. Donnez $\mathscr{L}(M)$, le langage accepté par M.

Exercice 7 ***Non-terminaison***

Donnez une machine M qui ne termine jamais. Donnez $\mathscr{L}(M)$, le langage accepté par M.

Exercice 8 ***Reconnaissance de langages classiques***

Soit l'alphabet $\Sigma = \{a, b\}$. Pour chacun des langages suivants, donnez une MT qui le reconnaît $L_1 = \Sigma^*$, $L_2 = \emptyset$, $L_3 = \{\epsilon\}$, $L_4 = \{a^n b^n \mid n \in \mathbb{N}\}$, $L_5 = \Sigma \cup \{\omega.R(\omega) \mid \omega \in \Sigma^*\}$ où R est l'opération qui renverse un mot et donc L_5 est l'ensemble des palindromes sur Σ.

1.6 Programmer avec des machines de Turing

L'objectif de cette section est de montrer qu'on peut simuler les langages de programmation usuel (par exemple C) à l'aide des machines de Turing et de travailler avec une représentation binaire des données. Ainsi nous aurons montrer que les machines de Turing peuvent simuler tous les comportements d'un ordinateur. La conséquence (au chapître suivant) sera que ce qui est impossible pour une MT le sera pour un ordinateur.
Pour programmer nous avons besoin de représenter les données (entrée et résultat), des fonctions et des prédicats de bases et des constructions des langages de programmation.

1.6.1 Représentation binaire des données

Définition 9 *On définit* $\mathbb{N}^* = \bigcup_{i\in\mathbb{N}} \mathbb{N}^i$ *l'ensemble des listes d'entiers comme l'ensemble des vecteurs d'entiers de taille 0, 1, 2, etc.*

Exemples :

- $\mathbb{N}^2$ est l'ensemble des couples d'entiers (n_1, n_2)
- $\mathbb{N}^p$ est l'ensemble des vecteurs d'entiers à p composantes $(n_1, \ldots, n_p)$
- $\mathbb{N}^1 = \{(0), (1), (2), \ldots\}$ contient les vecteurs à une seule composante
- $\mathbb{N}^0$ contient un unique vecteur : $()$

Pour représenter les vecteurs d'entiers sur le ruban on peut utiliser un alphabet à 8 symboles $\Sigma_8 = \{\$, (, ,,), 0, 1, \square, \S\}$ et utiliser la représentation binaire des entiers. On choisit la notation *little-endian* qui consiste à de mettre les unités à gauche. Ainsi, la représentation binaire de 6, $[6]_2 = 011 = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2$.

Exemple : le vecteur $(1, 2, 6)$ se notera

$^\infty\square$	**(**	1	**,**	0	1	**,**	0	1	1	**)**	$\square^\infty$

Au TD 1 vous verrez comment se ramener à un alphabet binaire $\Sigma_2 = \{\boxed{0}, \boxed{1}\}$.

Exercice 9 ***Codage d'un vecteur d'entiers en binaire***

1. Donnez un codage dans l'alphabet $\Sigma_8 = \{\$, (, ,,), 0, 1, \square, \S\}$ du vecteur $(0, 1, 2)$ en transformant les entiers en représentation binaire *little-endian* : **(**0**,**1**,**01**)**
2. Donnez un codage des symboles de l'alphabet $\Sigma_8 = \{\$, (, ,,), 0, 1, \square, \S\}$ sous forme de triplet de symboles $\Sigma_2 = \{\boxed{0}, \boxed{1}\}$

Σ_8	$\to$	$\Sigma_2 \times \Sigma_2 \times \Sigma_2$
0	$\mapsto$	0 0 0
1	$\mapsto$	1 1 1
(	$\mapsto$	1 1 0
)	$\mapsto$	0 1 1
,	$\mapsto$	0 1 0
$	$\mapsto$	0 0 1
§	$\mapsto$	1 0 1
$\square$	$\mapsto$	1 0 0

3. Donnez le codage dans l'alphabet $\Sigma_2 \times \Sigma_2 \times \Sigma_2$ du vecteur **(**0**,**1**,**2**)** :
   1 1 0 . 0 0 0 . 0 1 0 . 1 1 1 . 0 1 0 . 0 0 0 . 1 1 1 . 0 1 1
4. Donnez le mot $\omega \in \{\boxed{0}, \boxed{1}\}^*$ correspondant au codage de l'entrée $\$$**(**0**,**1**,**2**)**$\square$:
   0 0 1 . 1 1 0 . 0 0 0 . 0 1 0 . 1 1 1 . 0 1 0 . 0 0 0 . 1 1 1 . 0 1 1 . . 1 0 0
5. Décodez le résultat 0 0 1 . 1 1 0 . 1 1 1 . 0 1 0 . 0 0 0 . 1 1 1 . 0 1 0 . 1 1 1 . 0 0 0 . 1 1 1 . 0 1 1 trouvé sur le ruban après l'exécution d'une machine de Turing sur l'entrée précédente. Donnez le résultat sous la forme d'un vecteur d'entiers naturels : (1,2,3)

1.6.2 Un langage impératif à base de machines de Turing

Pour montrer que les machines de Turing ont l'expressivité d'un langage de programmation impératif, on donne pour chaque construction du langage un moyen de construire une MT équivalente.

- **Les fonctions et instuctions** de base sont des MT écrites à la main : $M_{\vec{\square}}$, M_{inc}, $M_{nul?}$, *etc.*
- **La séquence** $[M_1$ **;** $M_2]$ de deux MT est une MT qui exécute M_1 puis exécute M_2, cf. TD 1.
- **Les prédicats/tests** : une MT M réalise un prédicat si elle s'arrête pour toute donnée binaire $\omega \in \{\boxed{0}, \boxed{1}\}^*$ et termine
  - soit sur ◎ on dit alors que $M(\omega)$ vaut $\mathbb{V}$
  - soit sur ⊗ on dit alors que $M(\omega)$ vaut $\mathbb{F}$.
- **La conditionnelle** [if (M_{cond}) then M_1 else M_2] est une MT qui exécute M_1 si M_{cond} termine dans l'état ◎ et qui exécute M_2 si M_{cond} termine dans l'état ⊗
- **L'itération** [while (M_{cond}) do M_{body}] est une MT qui exécute M_{body} tant que M_{cond} termine dans l'état ◎, cf. EXAMEN 2015.

— **Le calcul** $f(i_1, \ldots, i_d)$ consiste à appliquer la MT M_f qui réalise la fonction f sur la donnée $[(i_1, \ldots, i_d)]_2$ inscrite sur le ruban.

— **Les variables** $i, j, \ldots$ sont représentées par des rubans séparés. Le ruban B_i contient la valeur de la variable i.

$B_i =$	$^\infty\square$	\$	*valeur de i*	$\square^\infty$
$B_j =$	$^\infty\square$	\$	*valeur de j*	$\square^\infty$
	⋮			

On verra en TD qu'une machine M à plusieurs rubans peut être simulée par une MT M' à un seul ruban, *ie.* que la MT M' construit sur son ruban un résultat correspondant à celui de M sur ces multiples rubans. Le résutlat est équivalent, les machines de Turing à un ruban ont donc le même pouvoir de calcul que les machines à plusieurs rubans, par contre elles sont plus lentes.

— **L'affectation** $i := \omega$ consiste recopier ω (soit une constante, soit le résultat d'un calcul, soit le contenu d'une variable) sur la bande B_i

On peut enrichir ce langage en ajoutant de nouvelles constructions, à condition d'expliquer comment réaliser la MT correspondant à cette construction.

1.6.3 Les fonctions arithmétiques

On s'intéresse aux fonctions $f : \mathbb{N}^p \to \mathbb{N}^*$ qui opérent sur *(la réprésentation binaire d')* un vecteur d'entiers $\vec{d} = (d_1, \ldots, d_p)$ représentant les p données d'entrée et qui retourne *(la représentation binaire d')* un vecteur d'entiers $\vec{r} = (r_1, \ldots, r_q)$ de taille q variable, représentants q résultats entiers.

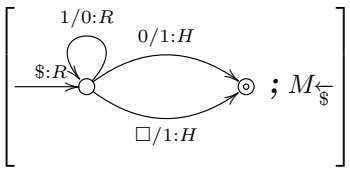
Exemples :

- *euclide* : $\mathbb{N}^2 \to \mathbb{N}^2 : (a, b) \mapsto (q, r)$ tels que $a = b \times q + r$ et $r < b$
- *inc* : $\mathbb{N}^1 \to \mathbb{N}^1 : (n) \mapsto (n + 1)$ où (n) est un vecteur à une seule composante
- *sub* : $\mathbb{N}^2 \to \mathbb{N}^* : (n, m) \mapsto (n - m) \in \mathbb{N}^1$ si $n \geq m$ *et* $() \in \mathbb{N}^0$ sinon pour indiquer que le résultat n'est pas défini.
- *dfp* : $\mathbb{N}^1 \to \mathbb{N}^* : (n) \mapsto (p_1, \ldots, p_k)$ tels que $n = p_1 \times \ldots \times p_k$ et p_i premiers
- *pi* : $\mathbb{N}^1 \to \mathbb{N}^1 : (n) \mapsto n^{\text{e}}$ décimale de π

Exercice 10

Donnez les MT M_{inc} et M_{dec} qui incrémente (resp. décrémente) un entier binaire *little-endian* inscrit sur le ruban.

SOLUTION

$M_{inc} =$ 

Certaines de ces fonctions se définissent simplement en donnant directement la MT qui la réalise ; les autres sont définies à l'aide des MT de bases en utilisant les constructions du langage de programmation du §1.6.2.

Notation On n'écrira pas explicitement le contenu des vecteurs mais seulement $f(\vec{d}) = \vec{r}$ et on désignera par $\omega_d = [\vec{d}]_2$ et $\omega_r = [\vec{r}]_2$ les représentations binaires des vecteurs $\vec{d}$ et $\vec{r}$.

1.6.4 Les prédicats

Un prédicat $P : \mathbb{N}^p \to Bool$ est un cas particulier de fonction $\mathbb{N}^p \to \mathbb{N}^*$ puisque
$Bool = \{\mathbb{F}, \mathbb{V}\} \simeq \{(0), (1)\} \subset \mathbb{N}^*$

Exemples :

- *nul ?* : $\mathbb{N}^1 \to Bool : (n) \mapsto \mathbb{V}$ si $n = 0$ *et* $\mathbb{F}$ sinon
- *inf ?* : $\mathbb{N}^2 \to Bool : (a, b) \mapsto \mathbb{V}$ si $a < b$ *et* $\mathbb{F}$ sinon

Exercice 11

Donnez une MT $M_{nul?}$ qui réalise le prédicat *nul ?* Attention, ϵ – représenté par le ruban $\overset{\infty}{\square}\,|\,\$\,|\,\square^{\infty}$ – n'est pas un nombre. Il serait erronné de l'assimiler à 0.

1.6.5 Application

Exercice 12 ***Énumération des mots de*** $\{0, 1\}^*$

Donnez une MT M_{next} qui à partir d'un mot $\omega \in \{0, 1\}^*$ écrit sur le ruban dans un autre mot binaire garantissant qu'en partant du ruban vide et en appliquant M_{next} on énumérera $\{0, 1\}^* \setminus \{\epsilon\}$. Notez que $M_{next} := M_{inc}$ qui incrémente un entier binaire ne convient pas car elle oublierait des mots binaires tels que $00, 010, 0100, 0010, \ldots$ qui contiennent des 0 non significatifs.

$$M_{next}(\epsilon) = 0,\ M_{next}(0) = 1,\ M_{next}(1) = 00,\ M_{next}(00) = 10,\ M_{next}(10) = 01,\ M_{next}(01) = 11,\ \ldots$$

Indication : On s'inspire de la MT M_{inc} mais au lieu de transformer $11\ldots11$ en $00\ldots001$ comme le ferait M_{inc} on transforme $11\ldots11$ en $00\ldots000$. Autrement dit M_{next} se comporte comme M_{inc} mais elle oublie la retenue lorsqu'on arrive sur le $\square$ qui suit le nombre binaire.

SOLUTION

$$M_{next} = \left[\ \begin{array}{l} \text{1/0:}R \\ \text{0/1:}H \\ \$\text{:}R \\ \square\text{/0:}H \end{array}\ ;\ M_{\overset{\leftarrow}{\$}}\ \right]$$

Résumé

Nous avons vu qu les machines de Turing permettent de définir un langage de programmation basique, qui permet de coder tout algorithme qu'on pourrait écrire dans des langages plus évolué.

Nous avons vu qu'on pouvait coder dans l'alphabet binaire $\Sigma = \boxed{0}, \boxed{1}$ tout type de données : entiers, symboles, vecteurs, images, ... C'est le codage utilisé dans les ordinateurs pour représenter en mémoire les données.

Les MT sont capables de prouesses impresionnantes. Elles peuvent calculer les décimales de π aussi loin que l'on veut, elles peuvent énumérer tous les mots binaires, ... Cependant, nous démontrerons dans le chapître suivant – en raisonnement par l'absurde – que les fonctions $\mathbb{N} \to Bool$, en apparence simple, ne sont pas toutes réalisables par les machines de Turing.

Chapitre 2

Machine Chimique

À la fin des années 80, Le Métayer & Banâtre avec Γ [BLM93] puis Berry & Boudol avec la *CHemical Abstract Machine* [BB92], ont proposé un modèle de calcul très simple qui modélise le calcul massivement parallèle (et même le parallèlisme maximal). Depuis ce modèle a fait du chemin et a eu des utilisations très diverses [BFLM00]. Nous allons brièvement présenter leur proposition qui considère les réactions chimiques comme moteur du calcul. Nous utiliserons ce modèle au chapître suivant pour décrire les bijections de Cantor.

2.1 Le modèle chimique

Les algorithmes = des réactions chimiques Les algorithmes sont définis sous forme de règles comme les réactions chimiques

$$\begin{array}{lrcl} \text{forme générale :} & \textit{réactifs} & \xrightarrow{\textit{condition}} & \textit{produits} \\ \text{chimie} & : H,\ O,\ H & \longrightarrow & H_2O \\ \text{algorithmique} & : \quad x, y & \longrightarrow & x+y \end{array}$$

Les données = un amas de molécules non ordonné où l'on trouve plusieurs occurences de la même molécule. Inspiré de la métaphore chimique, la mémoire sera un **multi-ensemble** de données, c'est-à-dire un « *ensemble* » au sens où les éléments ne sont pas ordonnés, mais « *multi* » puiqu'on autorise la présence de plusieurs occurences du même élément.

Exemple : $\{\!|1,1,0,2,0,1|\!\}$ est un multi-ensemble à 6 éléments contenant 3 occurences de l'élément 1.

- **Les atomes** sont les éléments des type de base : $\mathbb{N}, \mathit{Bool}, \ldots$
- **Les molécules** sont des termes, c'est-à-dire des données complexes forgées à l'aide de constructeurs (notés en majuscule A,B,C,...) et d'atomes ou de plus petites molécules déjà formées.

  Exemple : Avec un seul constructeur $\mathrm{C}(\_,\_)$ à deux arguments on peut manipuler

  - **des couples d'entiers :**
    Appliquée au multi-ensemble $\{\!|\mathrm{C}(0,0), \mathrm{C}(7,3)|\!\}$, la règle $\mathrm{C}(x,y) \xrightarrow{y<x} \mathrm{C}(y,x)$ ordonne les composantes des couples et donne $\{\!|\mathrm{C}(0,0), \mathrm{C}(3,7)|\!\}$
  - **des couples de couples d'entiers :**
    Appliquée au multi-ensemble à un seul élément $\{\!|\ \mathrm{C}(\mathrm{C}(0,0), \mathrm{C}(3,7))\ |\!\}$, la règle $\mathrm{C}(x,y) \longrightarrow x, y$ scindent les couples et donne $\{\!|\mathrm{C}(0,0),\ \mathrm{C}(3,7)|\!\}$ après une étape d'exécution et $\{\!|0,0,3,7|\!\}$ au final.
- **Attention** à ne pas confondre les constructeurs $\mathrm{C}, \mathrm{S}, \mathrm{Z}, \ldots$ avec des fonctions $c : (x,y) \mapsto x^2 + y^2$, $s : x \mapsto x+1$ ou des variables $z = 0$:

  Exemples :
  - $\mathrm{C}(1,2)$ n'est pas un calcul, c'est un terme *ie.* une donnée inerte alors que $c(1,2)$ est un appel de fonction qui retourne $1^2 + 2^2 = 5$
  - $\mathrm{S}(\mathrm{S}(\mathrm{Z}))$ est un terme qui vaut ... $\mathrm{S}(\mathrm{S}(\mathrm{Z}))$ alors que $s(s(z))$ vaut $s(s(0)) = (0+1)+1 = 2$

Les conditions de la réaction Les conditions des réactions s'écrivent à l'aide des prédicats usuels (par exemple, $x < y \ \wedge \ x \neq z$) auxquels on ajoute la reconnaissance de motif (*pattern matching*) puisque les molécules sont des termes.

Exemple :

« si le terme t n'est pas un couple, l'éliminer » s'écrit $t \xrightarrow{t \neq \mathrm{C}(\_,\_)}$.

« si le terme t est un couple, le scinder en composantes mais sans répétition » s'écrit $\begin{cases} \mathrm{C}(x,y) & \xrightarrow{x \neq y} & x,\ y \\ \mathrm{C}(x,x) & \longrightarrow & x \end{cases}$

Le principe de calcul

1. Les réactions s'effectue en parallèle, simultanément, sur tous les réactifs qui satisfont leurs conditions et ce jusqu'à ce qu'aucune réaction ne puisse plus s'appliquer
2. Un élément réactif ne peut pas participer à plusieurs réactions simultanément
3. les réactifs sont consommés *ie.* retriés du multi-ensemble
4. les produits sont crées *ie.* ajoutés au multi-ensemble

2.1.1 Les limitations voulues de Gamma et l'application aux protocoles réseaux

La métaphore chimique s'applique encore pour définir ce qui est impossible en Gamma.

On ne peut pas

— **mettre des priorités entre les règles.** En chimie, les réactions s'effectuent sans demander la priorité ; si une réaction peut avoir lieu, elle a lieu.

— **demander si une règle a fini de s'appliquer.** En chimie, on ne sait jamais quand une solution a stabilisé ; le seul moyen serait d'attendre un temps très très très long. En Gamma c'est pareil, il se peut que la donnée D qui réagira avec la donnée D' ne l'ait pas encore croisée. Il faudrait donc attendre et attendre encore pour être sûr, mais alors on perdrait tout l'intérêt de la programmation parallèle.

— **définir l'opérateur de séquence « ; ».** La séquence de deux réactions « r_1 ; r_2 » signifie « lancer la réaction r_1 » puis « lancer la réaction r_2 » **quand** r_1 **a fini d'agir** : c'est impossible du fait des limitations précédentes.

— **tester l'absence d'une donnée**. En chimie on peut prouver la présence d'une molécule m en introduisant un produit qui réagit avec la molécule m pour créer quelque chose d'observable (un changement de couleur par exemple). En revanche on ne peut pas tester l'absence d'une molécule. Gamma reprend la même idée, on ne peut pas écrire de règle « $\neg m \rightarrow \ldots$ » car cela obligerait l'exécution à stopper toutes réactions pour examiner une par une les données du multi-ensemble afin de tester l'absence de m ; on perdrait alors tout le parallèlisme.

La programmation chimique est donc un art subtil qui consiste à contourner ces limitations. Elle est utilisée pour définir des protocoles réseaux car c'est typiquement un cas dans lequel on n'a pas de garantie *de séquence, de terminaison, de priorité, de test d'absence (de réponse ou d'erreur)* et dans lequel on souhaite un parallèlisme maximal et sans blocage.

2.2 Applications

Exercice 13

1. Donnez le multi-ensemble résultat de l'application de la règle *sum* : $x, y \rightarrow x + y$ à $M = \{\!|1, 1, 0, 2, 0, 1|\!\}$: $\{\!|5|\!\}$
2. Combien d'étapes de calcul faut-il pour arriver au résultat ? 3 puisque que *sum* opère en parallèle
3. Quel(s) résultat(s) obtient-on en appliquant la règle $x, y \rightarrow x - y$ au multi-ensemble M ?
   Même question avec $x, y \rightarrow \frac{x}{y}$

SOLUTION

Selon l'ordre des réactions on peut obtenir $\{\!\{-1\}\!\}$ *ou* $\{\!\{1\}\!\}$ *car la soustraction n'est ni commutative* ($1,2 \to -1$ *alors que* $2,1 \to 1$)*, ni associative puisque* $(1-2)-3 = -4 \neq 2 = 1-(2-3)$*. Même remarque pour* $(1/2)/3 = \frac{1}{2\times 3} \neq \frac{3}{2} = 1/(2/3)$

Exercice 14

1. Donnez une règle qui à partir de $\{\!\{0\}\!\}$ génère un multi-ensemble contenant tous les entiers : $x \to x,\ x+1$
2. Donnez le multi-ensemble qu'on obtient après trois pas de calcul : $\{\!\{0,1,1,2,1,2,2,3\}\!\}$

SOLUTION

$\{\!\{0\}\!\} \xrightarrow{1} \{\!\{0,1\}\!\} \xrightarrow{2} \{\!\{0,1,1,2\}\!\} \xrightarrow{3} \{\!\{0,1,\ 1,2,\ 1,2,\ 2,3\}\!\}$

---

3. Donnez une règle qui à partir de $\{\!\{\mathrm{G}(0)\}\!\}$ génère exactement $\mathbb{N} = \{\!\{0,1,2,\dots\}\!\}$ sans répétition : $\mathrm{G}(n) \to n, \mathrm{G}(n+1)$
4. Ajoutez une règle avec condition pour effectuer le crible d'Eratosthène qui élimine tout nombre qui admet un diviseur strictement plus petit que lui et différent de 1 : $x, y \xrightarrow{x|y \ \wedge\ x<y \ \wedge\ x\neq 1} x$

Exercice 15

On considère un constructeur T à un argument et un multi-ensemble contenant des éléments de type $\mathrm{T}(e)$.

1. Donnez deux règles qui permettent de compter le nombre d'élément de type T d'un multi-ensemble. Utilisez des constructeurs T' et Card afin de ne pas compter plusieurs fois le même élément.

SOLUTION

$$\mathrm{T}(e) \to \mathrm{T'}(e), \mathrm{Card}(1) \quad || \quad \mathrm{Card}(i), \mathrm{Card}(j) \to \mathrm{Card}(i+j)$$

---

2. Appliquez vos règles au multi-ensemble $\{\!\{\mathrm{T}(1), \mathrm{T}(1), \mathrm{T}(0), \mathrm{T}(0)\}\!\}$: $\{\!\{\mathrm{T'}(1), \mathrm{T'}(1), \mathrm{T'}(0), \mathrm{T'}(0), \mathrm{Card}(4)\}\!\}$
3. Donnez des règles pour calculer le cardinal d'un multi-ensemble.

SOLUTION

$$e \xrightarrow{e\neq \mathrm{P}(\_) \ \wedge\ e\neq \mathrm{Card}(\_)} \mathrm{P}(e), \mathrm{Card}(1) \quad || \quad \mathrm{Card}(i), \mathrm{Card}(j) \to \mathrm{Card}(i+j)$$

Chapitre 3

Calculabilité

3.1 Fonction calculable

Intuitivement, une fonction calculable est une fonction $f : \mathbb{N}^p \to \mathbb{N}^*$ dont le résultat peut être obtenu par un procédé automatisable. Cette définition est imprécise et il existe plusieurs définitions mathématiques des fonctions calculables. Nous allons éclaircir cette notion.

Définition 10 (au sens de Turing) *Les* ***fonctions calculables*** *sont les machines de Turing qui s'arrêtent pour toute entrée binaire* $\omega \in \{0,1\}^*$.

3.1.1 Hypothèse de Church-Turing

De 1931 à 1936, Alan Turing présente ses machines, Jacques Herbrand & Kurt Gödel – les systèmes d'équations HG, Stephen Cole Kleene – les fonctions μ-récursives et Alonso Church propose le λ-calcul. Ces propositions sont des tentatives très différentes pour définir des fonctions calculables ... Elles se sont avérées être équivalentes. La preuve de ces équivalences consiste à trouver un codage d'un système dans un autre et réciproquement.

> *Puisque des modèles de calcul très différents conduisent à une notion équivalente de fonctions calculables, il est raisonnable de penser – mais ce n'est qu'une hypothèse – que la notion de fonctions calculables est indépendante du modèle de calcul. Puisqu'on ne parvient pas à concevoir un modèle de calcul plus puissant que ceux déjà connus, on adopte ces modèles comme définitions de la notion de* fonctions calculables.

En théorie de la calculabilité, on emploie aussi le terme historique de *«fonctions récursives»*. On lui préferera le terme *fonctions calculables* afin de pas confondre avec la notion de *fonctions récursives en programmtion* qui sont des fonction qui s'appellent elle-mêmes (on dit qu'elles font des appels récursifs).

3.1.2 Réalisation d'une fonction par une machine de Turing

On s'intéresse aux fonctions de $\mathbb{N}^p \to \mathbb{N}^*$ qui associent à un vecteur $\vec{d} = (d_1, \ldots, d_p)$ représentant les p données d'entrée un vecteur $\vec{r} = (r_1, \ldots, r_q)$ représentants q résultats entiers.

Notation On a montré au § 1.6.1 qu'on pouvait représenter un vecteur d'entiers $\vec{d}$ par un mot binaire $\omega_d = [\vec{d}]_2$. Désormais, au lieu de considérer des fonctions $f : \mathbb{N}^p \to \mathbb{N}^*$, on s'intéressera à la version $f : \Sigma^* \to \Sigma^*$ avec $\Sigma = \{0,1\}$ qui travaille en représentation binaire.

Définition 11 *La* MT M_f ***réalise*** *la fonction* f *si*

- *l'exécution de* M_f *s'arrête pour tout mot* $\omega_d \in \Sigma^*$
- $M_f(\omega_d) = \mathbb{V}(\omega_r)$ *chaque fois que* $f(\omega_d) = \omega_r$
- $M_f(\omega_d) = \mathbb{F}(\epsilon)$ *si* f *n'est pas définie pour* ω_d*, ce qu'on note* $f(\omega_d) = \epsilon$ *pour indiquer que* f *ne rend pas de résultat pour l'entrée* ω_d

f est **calculable** si et seulement si **il existe** une MT M qui **réalise** la fonction f
ie. M s'arrête pour tout mot de Σ^* et $M(\omega_d) = f(\omega_d)$

3.2 Ensemble dénombrable

3.2.1 Définition et théorème fondamental

Définition 12 *Un ensemble E est **dénombrable** s'il est en bijection avec $\mathbb{N}$, noté $E \simeq \mathbb{N}$*

Théorème 1 (de Cantor-Bernstein) *il existe une bijection $E \simeq \mathbb{N}$ si et seulement si il existe deux fonctions injectives $g : \mathbb{N} \to E$ et $h : E \to \mathbb{N}$.*

Attention, à ne pas conclure que (g, h) forment une bijection. Le théorème n'affirme pas que $\forall e \in E,\ g(h(e)) = e$ ni que $\forall n \in \mathbb{N},\ h(g(n)) = n$; il garantie seulement qu'une bijection existe.

Remaque Cette définition mathématique autorise à prendre pour g et h des *fonctions non-calculables.*

3.2.2 Ensembles dénombrables

Exercice 16 ***Application du théorème de Cantor-Bernstein***

Utilisez le théorème pour démontrer que $\mathbb{N} \simeq \mathbb{N} \times \mathbb{N}$.

SOLUTION

On prend $g : \mathbb{N} \to \mathbb{N} \times \mathbb{N} : n \mapsto (n, n)$, c'est une fonction injective puisqu'elle vérifie $n \neq i \Rightarrow (n, n) = g(n) \neq g(i) = (i, i)$.
On pend pour $h : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ la fonction $(n, p) \mapsto 2^n 3^p$ qui est injective puisque la décomposition d'un nombre en puissance de facteurs premiers est unique. ***Conclusion :*** *d'après le théorème de Cantor-Bernstein, il existe donc une bijection $\mathbb{N} \simeq \mathbb{N} \times \mathbb{N}$ mais le théorème de la donne pas.*

Proposition 1 *Les ensembles $\mathbb{N}, \mathbb{N}^2, \ldots, \mathbb{N}^i$ pour $i \in \mathbb{N}$ sont chacuns dénombrables. Encore plus surprenant : $\mathbb{N}^* = \bigcup_{i \in \mathbb{N}} \mathbb{N}^i$ est dénombrable ie. en bijection avec $\mathbb{N}$.*

Preuve : Les ensembles précédents sont tous infinis, mais pas plus grand que $\mathbb{N}$. Nous allons le démontrer à l'aide du principe de numérotation de Cantor qui permet de construire explicitment une bijection entre chacun de ces ensembles et $\mathbb{N}$, voir exercice ci-après. □

Définition 13 (Liste d'entiers, cf. INF124) *L'ensemble $\mathbb{N}^* = \bigcup_{i \in \mathbb{N}} \mathbb{N}^i$ est l'ensemble des vecteurs d'entiers de taille 0, 1, 2, ... On peut l'interpréter comme l'ensemble des **listes d'entiers** de taille variable : il suffit d'écrire les vecteurs $[n_1, n_2, n_3]$ au lieu de (n_1, n_2, n_3)*

Exemples :

- $\mathbb{N}^1 = \{(0), (1), (2), \ldots\}$ contient les vecteurs à une seule composante $\simeq \{[0], [1], [2], \ldots\}$
- $\mathbb{N}^0$ contient un unique vecteur : $()$ que l'on note $[]$ en caml

Exercice 17 ***Principe de numérotation de Cantor (à finir en TD)***

On a montré que $\mathbb{N}^2$ l'ensemble des vecteurs d'entiers de taille 2 est dénombrable.

1. Décrire en Gamma un procédé de numérotation des vecteurs d'entiers de taille 2 *(fait en cours)*
2. En déduire que l'ensemble des vecteurs d'entiers (i, j, k) vus comme des couples $((i, j), k)$ forme un ensemble dénombrable
3. En déduire par itération du procédé que l'ensemble des vecteurs d'entiers de taille $n \in \mathbb{N}$ est dénombrable
4. Montrez que l'ensemble des listes d'entiers est dénombrable.
5. En déduire que l'ensemble des chaînes de caractères est dénombrable

6. En déduire que l'ensemble des machines de Turing et celui des programmes est dénombrable.

SOLUTION

1. *On définit un tableau* $[0..\mathbb{N}[\times[0..\mathbb{N}[$ *qui contient tous les couples d'entiers possibles : à la ligne* ℓ *et la colonne* c *on trouve le couple* (ℓ, c)*. On obtient un tableau à deux dimensions (infinie) de la forme :*

$\mathbb{N}$:	0	1	2	3	4	...
0	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	...
1	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	...
2	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	...
⋮						

Pour attribuer un numéro unique à chaque couple, on suit le parcours de Cantor des anti-diagonales de Cantor en commençant par donner le numéro 0 au couple (0,0). On établit ainsi une bijection entre $\mathbb{N}$ *et l'ensemble* $\mathbb{N}\times\mathbb{N}$ *des couples d'entiers. Au départ le multi-ensemble est réduit à un élément* $\{\!|\text{GO}|\!\}$ *qui va initier la numérotation de Cantor des vecteurs d'entiers, de taille quelconque. La numérotation de Cantor des vecteurs de 2 entiers se code en* GAMMA *de la manière suivante :*

$$\begin{cases} \text{GO} \to C_2(0,(0,0)) \\ C_2(n,(l,c)) \xrightarrow{l-1\geq 0} C_2(n+1,(l-1,c+1)) \\ C_2(n,(0,c)) \to C_2(n+1,(c+1,0)) \end{cases}$$

2. $C_2(n,(i,j))$, $C_2(n',(n,k)) \to C_3(n',(i,j,k))$ *où* C_3 *est la numérotation de Cantor des vecteurs à 3 éléments.*

   Pourquoi obtient-on une bijection ? *La numérotation* C_2 *encode une bijection. Par conséquent,* $C_2(n',(n,k))$ *équivaut à* $n' \leftrightarrow (n,k)$ *et* $C_2(n,(i,j))$ *équivaut* $n \leftrightarrow (i,j)$*. Autrement dit,* n *est l'identifiant unique du couple* (i,j) *donc on peut sans risque de confusion interpréter le* n *du couple* (n,k) *comme le couple* (i,j) *– on obtient* $((i,j),k)$ *– puis interpréter cette écriture comme celle du triplet* (i,j,k)*. On définit* C_3 *comme la combinaison de deux applications de* C_2 *comme suit :*

$$n' \overset{C_2}{\leftrightarrow} (n,k) \overset{C_2}{\leftrightarrow} ((i,j),k) \leftrightarrow (i,j,k)$$

3. *On remarque que* $(i_1,i_2,i_3,\ldots,i_{n-1},i_n)$ *peut s'écrire* $(((i_1,i_2),i_3),\ldots,i_{n-1}),i_n)$ *et on utilise récursivement la numérotation sur les couples – comme on l'a fait pour les triplets* (i,j,k) *– jusqu'à obtenir* $C_n(e,(i_1,i_2,i_3,\ldots,i_{n-1},i_n))$ *où* e *est l'entier correspondant au vecteur de* n *éléments* $(i_1,i_2,i_3,\ldots,i_{n-1},i_n)$*.*

4. *Pour construite l'ensemble des listes d'entiers, on range les vecteurs d'entiers de tailles variables dans un tableau* $[0..\mathbb{N}[\times[0..\mathbb{N}[$*.*

   (a) *À la ligne 1 on range les vecteurs* (i) *de taille 1*

   (b) *À la ligne 2 on range les couples* (i,j) *dans l'ordre croissant de leur numérotation de Cantor : le* n *de* $C_2(n,(i,j))$

   (c) *À la ligne* ℓ *on range les vecteurs de taille* ℓ *dans l'ordre de la numérotation de Cantor des* C_ℓ *et ainsi de suite.*

   On obtient un tableau de la forme :

$\mathbb{N}$:	0	1	2	3	4	...
C_1	(0)	(1)	(2)	(3)	(4)	...
C_2	(0,0)	(1,0)	(0,1)	(2,0)	(1,1)	...
⋮						

   On réapplique alors le parcours de Cantor de la question 1 pour numéroter les vecteurs de taille variable. On parvient ainsi à établir une bijection entre $\mathbb{N}$ *et l'ensemble* $\mathbb{N}^*$ *des listes d'entiers.*

Ensemble dénombrable = en bijection avec $\mathbb{N}$

3.2.3 Ensembles non-dénombrables : Principe de diagonalisation de Cantor

On peut montrer en utilisant le principe de diagonalisation de Cantor [1] que les ensembles suivants ne sont **pas dénombrables** :

- $\mathbb{N} \to \mathbb{N} = \{f \mid f : \mathbb{N} \to \mathbb{N}\}$ = l'ensemble des fonctions à un paramètre entier et un résultat entier
- $\mathbb{N} \to \mathit{Bool} = \{p^? \mid p^? : \mathbb{N} \to \mathit{Bool}\}$ = l'ensemble des prédicats à un paramètre entier
- $\mathscr{P}(\mathbb{N}) = \{S \mid S \subseteq \mathbb{N}\}$ = l'ensemble des parties de $\mathbb{N}$, *ie.* l'ensemble de tous les sous-ensembles de $\mathbb{N}$
- $[0,1[\cap\mathbb{R}$ = l'ensemble des réels du segment ouvert $[0,1[$.

Preuve : Montrons $\mathbb{N} \to \mathbb{N}$ non dénombrable: Considérons une fonction $f : \mathbb{N} \to \mathbb{N}$. Elle est complètement définie par un tableau $[0..\mathbb{N}[$ qui indique pour chaque entier n la valeur $f(n)$ associée. On range alors les fonctions dans un tableau $[0..\mathbb{N}[\times[0..\mathbb{N}[$ comme suit :

$\mathbb{N} =$	0	1	2	3	4	5	...
$nul = f_0$	0	0	0	0	0	0	...
$id = f_1$	0	1	2	3	4	5	...
$inc = f_2$	1	2	3	4	5	6	...
$dbl = f_3$	0	2	4	6	8	10	...
f_4	..	..	..	..	$f_4(4)$	..	...
$\vdots$							

On peut donc repérer une fonction $\mathbb{N} \to \mathbb{N}$ par son numéro de ligne. Par exemple, la fonction identité serait f_1.

Supposons que l'ensemble $\mathbb{N} \to \mathbb{N}$ soit dénombrable c'est-à-dire en bijection avec $\mathbb{N}$. Alors il existe un tableau $[0..\mathbb{N}[\times[0..\mathbb{N}[$ comme le précédent qui décrit la bijection $\mathbb{N} \to (\mathbb{N} \to \mathbb{N}) : \ell \mapsto f_\ell$. Autrement dit le tableau contient toutes les fonctions $\mathbb{N} \to \mathbb{N}$: la ligne ℓ définit la fonction f_ℓ.

Mais alors la fonction $g : \mathbb{N} \to \mathbb{N}$ définie par $g(n) = f_n(n) + 1$ doit apparaître dans le tableau à une certaine ligne, disons ℓ, donc $g = f_\ell$.

> **Exemple :** La fonction g correspond à la diagonale du tableau incrémentée de 1. Dans le cas du tableau précédent, la fonction g serait $g(0) = 0+1,\ g(1) = 1+1,\ g(2) = 3+1,\ g(3) = 6+1,\ g(4) = f_4(4)+1, \ldots$

Dans ce cas $g(\ell) = f_\ell(\ell)$ d'après l'égalité précédente et $g(\ell) = f_\ell(\ell)+1$ par définition de g : Contradiction.

Conclusion : On a supposé $\mathbb{N} \to \mathbb{N}$ dénombrable et on aboutit à une contradiction, donc $\mathbb{N} \to \mathbb{N}$ n'est pas dénombrable. □

Exercice 18 *(à chercher pour s'entraîner au cas où ça tomberait en examen)*

Les trois autres exemples se démontrent de la même manière : on construit un tableau $[0..\mathbb{N}[\times[0..\mathbb{N}[$ censé contenir tous les éléments (un par ligne) et on montre avec une construction diagonale de la forme $f_n(n)$ qu'on peut construite un élément qui n'est pas dans le tableau.

- Pour $\mathbb{N} \to \mathit{Bool}$ la ligne ℓ définit le prédicat p_ℓ. La case à la ligne ℓ colonne c contient le booléen [2] ($\mathbb{V}$ ou $\mathbb{F}$) qui correspond au résultat du prédicat p_ℓ pour l'entier (c). Pour obtenir une contradiction on procède comme dans le cas $\mathbb{N} \to \mathbb{N}$: on utilise la diagonale du tableau pour définir un prédicat qui n'apparait pas dans le tableau.
- Pour $\mathscr{P}(\mathbb{N})$ la ligne ℓ contient le sous-ensemble S_ℓ de $\mathbb{N}$ défini par sélection des colonnes : un 0 dans la colonne c indique que $c \notin S_\ell$ un 1 dans la colonne c indique que $c \in S_\ell$.
- Pour $\mathbb{R} \cap [0,1[$, la ligne ℓ contient le réel r_ℓ de la forme $\mathbf{0}{,}d_0d_1d_2\ldots$ où chaque décimale d_c est un entier entre 0 et 9 inscrit dans la colonne c.

1. Certains parlent de la «diagonale du fou» puisque les intuitions géniales de Cantor sur l'infini et ses paradoxes l'ont conduit plus d'une fois en hôpital psychiatrique avec interdiction formelle de refaire des maths. Heureusement qu'il n'a pas respecté l'injonction de ses médecins.
2. 0 ou 1 si vous préférez

3.3 Ensemble énumérable

Définition 14 *Un ensemble E est* ***énumérable***

(*i*) *s'il est* ***dénombrable****, c'est-à-dire en bijection avec* $\mathbb{N}$.
Il existe donc une fonction surjective $\mathbb{N} \to E$*, que l'on nommera* get*,*
et qui vérifie $\forall e \in E,\ \exists n \in \mathbb{N},\ get(n) = e$

(*ii*) *s'il existe une* MT M_{get} *qui* ***réalise*** *la fonction* get.

Définition 15 (Piqûre de rappel : injection, surjection, bijection)

(*a*) *Une fonction g est* ***bijective*** *si et seulement si elle est injective et surjective.*

(*b*) *Une fonction g est* ***injective*** *si et seulement si* $g(x) = g(y) \Rightarrow x = y$ *ou de manière équivalente* $x \neq y \Rightarrow g(x) \neq g(y)$.

(*c*) *Une fonction* $g : \mathbb{N} \to E$ *est* ***surjective*** *si et seulement si* $\forall e \in E,\ \exists n \in \mathbb{N},\ g(n) = e$.

Exercice 19 Σ^* ***est énumérable***

Considérons $\Sigma = \{0, 1\}$. Montrez que Σ^*, l'ensemble des mots binaires, est énumérable en exhibant une MT M_{get} qui réalise une fonction surjective de $\mathbb{N} \to \{0, 1\}^*$.

SOLUTION

(*i*) *On a montré à l'17 que* $\mathbb{N}^*$ *était dénombrable. Or* $\Sigma = \{0, 1\} \subseteq \mathbb{N}$ *donc* $\Sigma^* \subseteq \mathbb{N}^*$ *est lui aussi dénombrable.*

(*ii*) *On prouve que la fonction* $get : \mathbb{N} \to \Sigma^* : [n]_2 = \omega.1 \mapsto \omega$ *est surjective et on donne une MT* M_{get} *qui réalise cette fonction. La fonction* get *n'est pas définie pour* 0 *mais on peut la définir :* $get(0) = \epsilon$*. Remarquez que* get *n'est pas une bijection puisque* $get(0) = \epsilon = get(1)$ *mais ça reste une surjection.*
Montrons que get *est une surjection : il faut montrer que pour tout élément* $\omega \in \Sigma^*$ *il existe un entier binaire* $n \in [\mathbb{N}]_2$ *tel que* $get(n) = \omega$.

Preuve : Pour l'élément ω, on prend l'entier n qui s'écrit $\omega.1$ en binaire. On a bien $get(n) = \omega$. □

Ensemble E énumérable = E dénombrable et il existe une MT M_{get} qui réalise la fonction surjective $get : \mathbb{N} \to E$ qui à i associe le i^{e} élement de E.

Exercice 20 $\Sigma^* \times \Sigma^*$ ***est énumérable***

Ce résultat nous sera utile au chapître 4. Considérons $\Sigma = \{0, 1\}$. Montrez que $\Sigma^* \times \Sigma^*$, l'ensemble des couples de mots binaires, est énumérable en exhibant une MT M_{get} qui réalise une fonction surjective de $\mathbb{N} \to \{0, 1\}^* \times \{0, 1\}^*$.

Indication : Utilisez la fonction $[i]_2$ qui donne l'écriture binaire d'un entier i, la fonction get_{Σ^*} de l'19 et la bijection de Cantor $C_2 : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ définie à l'17.

SOLUTION

On construit la fonction $get : \mathbb{N} \to \Sigma^* \times \Sigma^*$ *de la manière suivante*

$$\begin{array}{rccccccc} get : & \mathbb{N} & \xrightarrow{C_2} & \mathbb{N} \times \mathbb{N} & \xrightarrow{[.]_2} & \Sigma^* \times \Sigma^* & \xrightarrow{get_{\Sigma^*}} & \Sigma^* \times \Sigma^* \\ & n & \mapsto & C_2(n) = (i, j) & \mapsto & ([i]_2, [j]_2) & \mapsto & \big(get_{\Sigma^*}([i]_2), get_{\Sigma^*}([j]_2)\big) \end{array}$$

Résumé

Ce chapître nous a appris que **ce qui est calculable par un procécédé automatique correspond à ce que peuvent faire les machines de Turing.**

Nous savions qu'on pouvait coder dans l'alphabet binaire $\Sigma = \boxed{0}, \boxed{1}$ tout type de données : entiers, symboles, vecteurs, images, ... Nous avons vu que **l'ensemble infini, Σ^*, de tous les mots binaires (toutes les données possibles) est énumérable.** Puisqu'une machine de Turing peut se représenter par un vecteur $(\Sigma, \mathscr{Q}, \overset{\searrow}{\textcircled{q}}, \delta, \mathcal{A}, \mathcal{R})$, on peut la représenter par le codage binaire de ce vecteur ; on détaillera ce codage au chapître suivant. On en conclut que les mt sont des mots de Σ^* ; elles sont donc dénombrables. On verra même que **les machines de Turing sont énumérables**.

Par ailleurs nous avons montré qu'**il existe des ensembles infinis, « trop infini » pour être énumérables** : par exemple, celui des prédicats $\mathbb{N} \to \mathit{Bool}$ et celui des fonctions $\mathbb{N} \to \mathbb{N}$.

La conclusion est qu'**il existe infiniment plus de fonctions que de machines de Turing :** il existe une infinité de fonction $\mathbb{N} \to \mathbb{N}$ non calculables, c'est-à-dire des fonctions pour lesquelles il n'y a pas de machine de Turing correspondante.

Chapitre 4

Décidabilité

Le chapître précédent nous a appris qu'**il existe infiniment plus de fonctions que de machines de Turing :** il existe une infinité de fonctions $\mathbb{N} \to \mathbb{N}$ non-calculables, c'est-à-dire des fonctions pour lesquelles il n'y a pas de machine de Turing correspondante.

Soit, **mais lesquelles ? Comment les reconnaître ? Peut-on donner des exemples de fonctions non-calculables ?** Voici les questions qui vont guider la suite du cours. Dorénavant notre objectif sera de mieux cerner ce qui n'est pas calculable.

Pour apporter des réponses nous allons simplifier la question et **réduire la question de la calculabilité d'une fonction**

> « existe-t'il une MT capable de calculer le résultat $\omega_r = f(\omega_d)$ à partir de la donnée ω_d ? »

à celle de la décidabilité d'une relation, qui se contente de répondre $\mathbb{V}$ ou $\mathbb{F}$ lorsqu'on lui donne l'entrée ω_d ainsi qu'une proposition de résultat ω_r

> « existe-t'il une MT qui reconnaît les couples (ω_d, ω_r) tels que $\omega_r = f(\omega_d)$? »

Dans ce chapître nous allons définir la décidabilité et établir une correspondance entre fonctions et relations.

4.1 Ensemble décidable

On s'intéresse à des ensembles d'éléments pris dans l'ensemble $\mathscr{U}$ de tous les éléments possibles, on appelle l'ensemble $\mathscr{U}$ l'**Univers.**

Définition 16 *La **fonction indicatrice/caractéristique** d'un sous-ensemble $E \subseteq \mathscr{U}$, notée $\in_E^?$ est un prédicat $\mathscr{U} \to Bool$ qui **test l'appartenance à** E des éléments u de l'Univers. Il est défini par :*

$$\in_E^? (u) = \mathbb{V} \iff u \in E$$
$$\in_E^? (u) = \mathbb{F} \iff u \notin E$$

Définition 17 (ensemble décidable) *Un ensemble $E \subseteq \mathscr{U}$ est **décidable** si et seulement si*

*(i) l'univers $\mathscr{U}$ est **énumérable***

*(ii) et si la **fonction indicatrice** $\in_E^?$ de E est **calculable***

Ensemble Décidable = Univers énumérable & Appartenance Calculable

Proposition 2 *Considérons l'alphabet $\Sigma = \{0, 1\}$. **L'ensemble Σ^* des mots binaires est décidable.***

Preuve : On doit montrer (i) et (ii). Auparavant il faut préciser l'Univers dans lequel on se place. Puisque les éléments de $\Sigma^* = \{0,1\}^*$ sont des mots binaires on choisit $\mathscr{U} = \Sigma^*$. On a bien $\Sigma^* \subseteq \mathscr{U}$.
(i) On a déjà montré à l'19 que Σ^*, l'univers est énumérable. Ce qui démontre (i).
(ii) $\in^?_{\Sigma^*}$, la fonction indicatrice de Σ^* est calculable puisqu'il existe une MT qui termine toujours pour tout mot de l'univers et qui accepte tous les mots de Σ^* : c'est la MT réduite à une seul état initial et accepteur, $\rightarrow$◎.

On peut généraliser la proposition à n'importe quel alphabet $\Sigma = \{s_1, \ldots, s_{2^n}\}$ en codant les symboles par des vecteurs de n-bits comme dans l'9 et en considérant l'univers $\mathscr{U} = \{0,1\}^n$. □

4.2 Langage décidable (dit aussi *«langage récursif»*)

Dans la suite on s'intéresse aux cas particuliers des **langages** *ie.* des ensembles de mots écrits sur un **alphabet fini** Σ. Un langage est donc un sous-ensemble de l'univers Σ^* qui est énumérable (cf. 19). La définition 17 peut alors être simplifiée dans le cas des langages :

Définition 18 (Langage décidable) *Soit Σ un alphabet fini.*
*Un langage $L \subseteq \Sigma^*$ est **décidable** si et seulement si $\in^?_L$ la **fonction indicatrice** de L est **calculable**.*

Langage Décidable = Alphabet fini & Appartenance Calculable

Vocabulaire historique En théorie de la calculabilité, on emploie aussi le terme historique *«langage récursif»* pour langage décidable, en référence au fait que la fonction indicatrice est *«récursive»* au sens de calculable.

Proposition 3 (Langage décidable, définition équivalente) *Un langage L est **décidable** s'il existe un machine de Turing M_L dont l'exécution **s'arrête pour toute entrée** $\omega \in \Sigma^*$ avec $M_L(\omega) = \mathbb{V}$ si $\omega \in L$ et $M_L(\omega) = \mathbb{F}$ si $\omega \notin L$. On dit que la machine M_L **décide** L.*

Preuve : La machine de Turing M_L code la fonction indicatrice $\in^?_L$ puisque $M_L(\omega) = \mathbb{V} \iff \omega \in L$ et $M_L(\omega) = \mathbb{F} \iff \omega \notin L$. Mais alos, la fonction indicatrice de L est calculable puisque M_L s'arrête pour toute entrée $\omega \in \Sigma^*$. **Conclusion :** L est donc un langage décidable d'après la définition 18. □

M décide L = M s'arrête pour tous les mots de Σ^* et répond $\mathbb{V}$ si $\omega \in L$ et répond $\mathbb{F}$ si $\omega \notin L$

4.3 Calcul d'une fonction / Décision d'une relation

Définition 19 (Relation binaire décidable) *Soit $R \subseteq \Sigma^* \times \Sigma^*$ une relation binaire. La MT M **décide** la relation R si et seulement si*

- *l'exécution de M sur tout mot $(\omega_d, \omega_r) \in \Sigma^* \times \Sigma^*$ s'arrête*[1]
- *$M(\omega_d, \omega_r) = \mathbb{V}$ chaque fois que $(\omega_d, \omega_r) \in R$*
- *$M(\omega_d, \omega_r) = \mathbb{F}$ chaque fois que $(\omega_d, \omega_r) \notin R$*

4.3.1 Codage d'une fonction calculable sous la forme d'une relation décidable

Une fonction $f : \Sigma^* \to \Sigma^*$ peut être représentée par **une relation $R_f \subseteq \Sigma^* \times \Sigma^*$ c'est-à-dire un ensemble de couples** (ω_d, ω_r) tels que $f(\omega_d) = \omega_r$.

$$R_f = \{(\omega_d, \omega_r) \mid f(\omega_d) = \omega_r\}$$

Une relation R_f peut aussi être représentée par un prédicat qui répond $\mathbb{V}$ si le couple (ω_d, ω_r) appartient à l'ensemble R_f et répond $\mathbb{F}$ sinon. Un tel prédicat est exactement la fonction indicatrive de R_f.
Les notions suivantes sont équivalentes :

1. On exclut ainsi le cas $M(\omega_d, \omega_r) =?$. Il ne reste alors que deux cas possibles.

$M_f(\omega_d) = \omega_r$	mt *qui calcule la fonction* f
$\equiv$	
$f(\omega_d) = \omega_r$	*fonction* f
$\equiv$	
$(\omega_d, \omega_r) \in R_f$	*relation* $R_f = \{(\omega_d, \omega_r) \mid f(\omega_d) = \omega_r\}$
$\equiv$	
$M_{R_f}(\omega_d, \omega_r) = \mathbb{V}$	mt *qui décide la relation* R_f

4.3.2 Réduction de la calculabilité à la décidabilité

Proposition 4 (Lien entre Calculabilité et Décision)

La fonction $f : \Sigma^* \to \Sigma^*$ *est calculable* $\iff$ *La relation binaire* $R_f \subseteq \Sigma^* \times \Sigma^*$ *est décidable.*

Que dit cette proposition ? Que si une fonction f est calculable, sa relation R_f est décidable (et réciproquement).

On peut réduire l'étude des fonctions calculables à l'étude des relations binaires décidables.

C'est principalement l'implication ($\Rightarrow$) qui nous intéresse

f calculable $\implies$ R_f décidable

mais surtout sa version équivalente sous forme **contraposé** [2]

R_f non-décidable $\implies$ f non-calculable

qui nous permet de découvrir des fonctions non-calculables en étudiant les ensembles non-décidables. Au passage on a montre aussi la **réciproque** [3] pour obtenir une équivalence entre les notions de calculabilité et décidabilité.

f calculable $\impliedby$ R_f décidable

4.3.3 Preuve de la réduction

La preuve de la propostion 4 n'est pas compliquée et c'est un exemple classique de raisonnement basé sur les machines de Turing.

Preuve :

($\Rightarrow$) La fonction f **est calculable** signifie qu'il existe une mt M_f qui termine pour tout mot ω de Σ^* et telle $M_f(\omega_d) = \omega_r$ si f est définie pour ω_d et $f(\omega_d) = \omega_r$. **On doit montrer que l'ensemble** $R_f \subseteq \Sigma^* \times \Sigma^*$ **est décidable.** Pour cela,

- (i) on doit montrer que l'univers $\mathscr{U} \stackrel{def}{=} \Sigma^* \times \Sigma^*$ est énumérable : c'est le but de l'20.
- (ii) on doit montrer qu'il existe une machine de Turing M_{R_f} qui décide le langage $R_f \stackrel{def}{=} \{(\omega_d, \omega_r) \mid f(\omega_d) = \omega_r\}$, c'est-à-dire une mt qui termine pour toute entrée (ω, ω') de $\mathscr{U}$, qui répond $\mathbb{V}$ si $(\omega, \omega') \in R_f$ et qui répond $\mathbb{F}$ si $(\omega, \omega') \notin R_f$.

  Pour montrer qu'une telle machine de Turing existe, on va la construire : On définit M_{R_f} comme une machine de Turing à deux bandes ; on a vu qu'il était possible ensuite de la transformer en une mt classique à une bande. Pour interroger le prédicat M_{R_f} sur le couple (ω_d, ω_r) on écrit ω_d sur une première bande B_d et ω_r sur la seconde bande B_r. On lance M_f

2. La contraposé est simplement l'application de l'équivalence logique $A \Rightarrow B \equiv \neg B \Rightarrow \neg A$
3. Ne pas confondre réciproque et contraposé. La réciproque de $A \Rightarrow B$ est $B \Rightarrow A$: elle ne se déduit pas de $A \Rightarrow B$.

sur la bande B_d, l'exécution s'arrête et écrit le résultat de $f(\omega_d)$ sur la bande B_d. On utilise ensuite la MT M_{comp} pour comparer les bandes B_d et B_r si elle sont identitques, on va dans un état ◎, sinon dans ⊗.

$$M_{R_f}(\omega_d, \omega_r) \stackrel{def}{=} [B_d := \omega_d \ ; \ B_r := \omega_r \ ; \ B_d := M_f(B_d) \ ; \ M_{comp}(B_d, B_r)]$$

($\Leftarrow$) La relation R_f **est décidable** signifie qu'il existe une MT M_{R_f} qui termine pour tout mot (ω_d, ω_r) de $\Sigma^* \times \Sigma^*$ et répond $\mathbb{V}$ si $(\omega_d, \omega_r) \in R_f$ et répond $\mathbb{F}$ sinon. **Montrons qu'on peut alors constuire une machine de Turing** M_f qui pour chaque entrée ω_d calcule le résultat de $f(\omega_d)$.

On construit une machine de Turing M_f à trois bandes B_d, B_r, B_n qu'on pourrait transformer en une MT classique à une bande. Puisque Σ^* est énumérable (voir l'19), on peut utliser la MT $M_{get} : [\mathbb{N}]_2 \to \Sigma^*$ pour énumérer les mots de Σ^*.

M_f calcule f de la manière suivante, pas efficace du tout mais qui termine et donne le résultat attendu :

(1) On inscrit ω_d sur la bande B_d et 0 sur la bande B_n

(2) On recopie la bande B_n sur la bande B_r.

(3) On applique M_{get} sur B_r, on obtient un mot de Σ^*.

(4) On applique M_{R_f} sur (B_d, B_r).

(5) Si M_{R_f} répond $\mathbb{V}$ alors, par définition de R_f, le contenu de B_r est le résultat de $f(\omega_d)$ cherché.

(6) Si M_{R_f} répond $\mathbb{F}$ alors on passe au mot suivant de Σ^* : on applique M_{inc} sur la bande B_n et on reprend à l'étape (2).

$$M_f(\omega_d) \stackrel{def}{=} \left[\begin{array}{l} B_d := \omega_d \ ; \\ B_n := 0 \ ; \\ \text{do } [B_r := M_{get}(B_n) \ ; \ B_n := M_{inc}(B_n)] \text{ until } (M_{R_f}(B_d, B_r)) \ ; \\ \text{return } B_r \end{array} \right]$$

Pour garantir la terminaison de cette MT, il faut montrer que pour chaque entrée $\omega_d \in \Sigma^*$ il existe un résultat ω_r qui correspond au résultat de f. Examinons le cas d'une fonction $\hat{f} : \Sigma^* \setminus \{\Omega\} \to \Sigma^*$ qui ne serait pas définie si le mot en entrée est Ω. Dans ce cas, on ne parviendrait jamais à trouver le résultat ω_r correspondant à l'entrée $\omega_d = \Omega$ puisqu'il n'est pas défini. Il n'y aurait donc aucun couple (Ω, ω) dans $R_{\hat{f}}$ et la MT qu'on a défini ne terminerait pas.

Pour résoudre ce problème on impose que les fonctions soient définies pour toute entrée, quitte à forcer une fonction $\hat{f}$ à rendre un symbole spécifique (ϵ ou $\perp$) pour les entrées où elle n'est pas définie. On compléte alors la fonction $\hat{f}$ pour l'étendre à tout Σ^* de la façon suivante : $\hat{f}(\Omega) = \epsilon$ ou $\hat{f}(\Omega) = \perp$ selon la convention choisie.

Dans ce cours, on choisira de retourner ϵ mais pour être plus rigoureux il faudrait créer un caractère nouveau $\perp$ et étudier les fonctions $\Sigma^* \to \Sigma^* \cup \{\perp\}$ Ce qui ne change pas fondamentalement les preuves mais complique un peu la présentation. □

Résumé

Le raisonnement de la section précédente se nomme **réduction** : On a réduit l'étude des fonctions calculables par les MT à la question un peu plus simple[4] des langages décidables par les MT. On reverra la notion de réduction lorsqu'on abordera le problème de l'arrêt d'une machine de Turing.

4.4 La MT universelle, un interpréteur de machines de Turing

A.Turing a défini les MT afin d'étudier les limites du calculable : ce qui est faisable par un ordinateur, ce qui ne le sera jamais quel que soit l'ordinateur ou le langage de programmation utilisé.

4. La question est plus simple puisque dans le cas de la décision, on ne se préoccupe pas de ce qu'il y a d'écrit sur le ruban après l'exécution, mais uniquement de l'arrêt de la MT et, le cas échéant, de l'état dans lequel elle s'est arrêtée : ◎ ou ⊗.

Les raisonnments qui lui ont permis d'exhiber des problèmes concrets qui n'ont pas de solution informatique reposent sur l'utilisation d'une MT universelle U capable d'exécuter toute machine de Turing M sur un mot ω. Nous allons donc présenter le codage qui permet de représenter par un mot binaire m une MT M *ie.* $m = [M]_2$ Ensuite nous présenterons le principe de fonctionnement de la MT universelle U. C'est une MT à deux bandes qui prend en premier paramètre la représentation binaire m d'une MT M inscrite sur la bande B_2, puis prend en argument un mot $\omega \in \{0,1\}^*$ sur la bande B_1. L'exécution de $U(m)(\omega)$ doit donner le même résultat (soit $\mathbb{V}$, soit $\mathbb{F}$, soit ?) que l'exécution de M sur ω. Ce qu'on résume par l'égalité :

$$U(m) = M \text{ où } m = [M]_2$$
En particulier, $U(m)(\omega) = M(\omega)$ et $U(m)(\omega) \to^\infty$ si et seulement si $M(\omega) \to^\infty$

Remarque m représente la description de M, *ie.* c'est un programme (pas très lisible mais il décrit bien un algorithme). U est donc un **interpréteur** qui lit le programme m et l'exécute sur ω, tandis que M peut être considérée comme une **version compilée** de m. $M(\omega)$ correspond alors à un appel au **programme exécutable** M sur la donnée ω passée en ligne de commande.

4.4.1 Représentation binaire d'une machine de Turing (PROJET MT'2019)

Considérons une MT $M = (\Sigma, \mathscr{Q}, \mathbf{q}_i, \delta)$ avec $\Sigma = \{0, 1, \square, \$\}$.
Pour coder cette représentation de M sur le ruban on va considérer un alphabet Σ_U contenant les symboles de Σ et enrichi de nouveaux symboles :

— A,R,Q, (, :,) afin de représenter les états quelconque par Q, accepteur par A, rejet par R.

Exemples :

- l'état (5) sera représenté par (A : 101) ce qui donne

$^\infty\square$	(	A	:	1	0	1	)	$\square^\infty$

sur le ruban

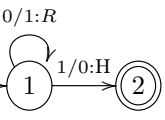
- l'état $\otimes$ sera représenté par (R : 0)
- l'état (2) sera représenté par (Q : 01).

— L,H,R afin de représenter les déplacements $d \in \{$L,H,R$\}$ de la tête lors des transitions.

Exemple : La transition (1) $\xrightarrow{\ell/e:d}$ ((2)) avec $\ell, e \in \Sigma$ sera représentée sur le ruban par

(	Q	:	1	)	ℓ	e	d	(	A	:	0	1	)

— Les transitions sont inscrites les unes derrière les autres. La séparation entre transitions est indiquée par la succession de cases |) | (|.

Exemple : La MT 

sera représentée par la séquence de transitions

(Q : 1)01R(Q : 1) (Q : 1)10H(A : 01)

Pour obtenir le mot m décrivant M on fait précéder la séquence des transitions de l'état initial suivi du symbole \$, on obtient alors le mot [5]

$m =$	(Q : 1)	\$	(Q : 1)	0	1	R	(Q : 1)	(Q : 1)	0	1	R	(A : 01)

Définition 20 *(codage binaire des machine de Turing) On note* $\mathscr{M} = \{m \in \{0,1\}^* \mid m = [M]_2,\ M \in \text{MT}\}$ *l'ensemble des mots binaires qui correspondent à des* MT *sur l'alphabet* $\{0, 1, \square, \$\}$.

Remarque Si on lance l'exécution de U sur un couple (m, ω) où $m \notin \mathscr{M}$, c'est-à-dire lorsque m ne correspond pas à codage valide de MT alors U s'arrête dans un état $\otimes$.

5. Pour rester lisible, on ne fait pas apparaître les séparations en cases des états.

4.4.2 Fonctionnement de la machine de Turing universelle (PROJET MT'2020)

On définit la MT U comme une machine à deux bandes. L'exécution de $U(m)(\omega)$ doit simuler l'exécution de M sur le mot ω.

— La bande B_2 contient le mot m correspondant à la représentation binaire de M, cf. § 4.4.1.

— La bande B_1 contiendra la configuration courante de M *ie.* $(\omega_L, \mathbf{q}, \ell.\omega_R)$ où ω_L est la partie du ruban de M située à gauche de T_M (la tête de M), ℓ est le symbole courant sur lequel pointe T_M, et ω_R est la partie du ruban située à droite de T_M. On représente la configuration $(\omega_L, \mathbf{q}, \ell.\omega_R)$ de la manière suivante sur la bande B_1 où l'état $\mathbf{q} = (t : n)$ avec $t \in \{\text{A,R,Q}\}$ est le statut de l'état, $n \in \mathbb{N}$ son numéro d'état et $[n]_2$ la représentation binaire de n.

$$B_1 = \begin{array}{|c|c|c|c|c|c|c|c|c|}\hline {}^{\infty}\square & \omega_L & (& t & : & [n]_2 &) & \ell & \omega_R & \square^{\infty} \\ \hline \end{array}$$
$$\uparrow$$
$$T_M$$

Par souci de lisibilité on désignera désormais par $\boxed{(\mathbf{q}_n)}$ un état $\boxed{(\ t\ :\ [n]_2\ )}$.

Exécution de $U(m)(\omega)$

1. Au départ B_1 contient le mot ω et B_2 contient le mot $m = [M]_2 = (\mathbf{q}_i)\$\underbrace{\ldots\delta\ldots}_{transitions}$.

   Par convention, les têtes T_1, T_2 des bandes 1 et 2 sont positionnées sur le symbole \$:

   $$B_1 = \begin{array}{|c|c|c|c|}\hline {}^{\infty}\square & \$ & \omega & \square^{\infty} \\ \hline \end{array} \quad \text{et} \quad B_2 = \begin{array}{|c|c|c|c|c|c|}\hline {}^{\infty}\square & \$ & (\mathbf{q}_i) & \$ & \delta & \square^{\infty} \\ \hline \end{array}$$
   $$\uparrow \qquad\qquad \uparrow$$
   $$T_1 \qquad\qquad T_2$$

2. On commence par modifier la bande 1 pour qu'elle contienne la **configuration initiale de l'exécution** $(\epsilon, \mathbf{q}_i, \omega)$ et pas seulement le mot ω. Pour cela on recopie au début de B_1 l'**état initial** de M, inscrit au début de la bande B_2.

   $$B_1 = \begin{array}{|c|c|c|c|c|c|}\hline {}^{\infty}\square & \$ & (\mathbf{q}_i) & \$ & \omega & \square^{\infty} \\ \hline \end{array} \quad \text{et} \quad B_2 = \begin{array}{|c|c|c|c|c|c|}\hline {}^{\infty}\square & \$ & (\mathbf{q}_i) & \$ & \delta & \square^{\infty} \\ \hline \end{array}$$
   $$\uparrow \qquad\qquad\qquad\qquad \uparrow$$
   $$T_1 \qquad\qquad\qquad\qquad T_2$$

3. Considérons le cas général auquel on aboutit après exécutions de plusieurs transitions.

   $$B_1 = \begin{array}{|c|c|c|c|c|c|}\hline {}^{\infty}\square & \omega_L & (\mathbf{q}) & \ell & \omega_R & \square^{\infty} \\ \hline \end{array} \quad \text{et} \quad B_2 = \begin{array}{|c|c|c|c|c|c|}\hline {}^{\infty}\square & \$ & (\mathbf{q}_i) & \$ & \delta & \square^{\infty} \\ \hline \end{array}$$
   $$\uparrow \qquad\qquad\qquad\qquad \uparrow$$
   $$T_1 \qquad\qquad\qquad\qquad T_2$$

   On parcourt les transitions δ sur B_2 jusqu'à trouver la transition à exécuter c'est-à-dire celle qui commence par $\boxed{(\mathbf{q})\ |\ \ell}$.

4. Si aucune transitions de δ ne correspond à $\boxed{(\mathbf{q})\ |\ \ell}$, alors la MT U s'arrête : son résultat $\mathbb{V}, \mathbb{F}, \omega'$ dépend de l'état $\mathbf{q}$ (accepteur, rejet, quelconque) et le mot résultat ω' correspond au contenu de la bande B_1 lorsqu'on a effacé $(\mathbf{q})$ ce qui fait passer de la configuration terminale au **mot résultat** .

5. Si on trouve dans la partie δ de B_2 une transition $(\mathbf{q}) \xrightarrow{\ell/e:d} (\mathbf{q}')$ alors on modifie la configuration sur B_1 :

   (a) on remplace ℓ de B_1 par e,

   (b) on remplace l'état courant par $(\mathbf{q}')$

   (c) on simule le déplacement d de la tête T_M en décalant l'état $(\mathbf{q}')$ d'un symbole vers la droite pour $d = \text{L}$ et vers la gauche pour $d = \text{R}$.

   Ensuite,

   (a) on déplace la tête T_1 sur l'état courant $(\mathbf{q}')$ et la tête T_2 sur le début de δ

   (b) on poursuit l'exécution de U en reprenant à l'étape 3

Indication : Pour faciliter les étapes (5b) et (5c), on adopte la convention que l'état $(\mathbf{q}_n)$ de B_1 a une taille fixe de $N+4$ symboles, nécessaire pour réprésenter $\boxed{(\mid t \mid : \mid [n]_2 \mid)}$ où N correspond au nombre de bits nécessaires pour représenter le plus grand indice n des états $\mathbf{q}_n$ de M, *ie.* $2^{N-1} < n \leq 2^N$. Si un état n'utilise pas les $N+4$ symboles, on compléte sa représentation par des symboles «)» comme suit :

Exemples :

- $\mathbf{q}_{16} \mapsto \boxed{(\mid \text{Q} \mid : \mid 0 \mid 0 \mid 0 \mid 1 \mid)}$
- $\mathbf{q}_5 \mapsto \boxed{(\mid \text{Q} \mid : \mid 1 \mid 0 \mid 1 \mid) \mid)}$
- $\mathbf{q}_0 \mapsto \boxed{(\mid \text{Q} \mid : \mid 0 \mid) \mid) \mid) \mid)}$

On doit placer dés la **configuration initiale** un état de taille N. Le plus simple est de compléter l'état initial noté sur m pour qu'il soit de taille $N = log_2(|\mathscr{Q}|)$.

La machine de Turing universelle que l'on a décrit garantie les propriétés suivantes :

$U(m)(\omega) = \mathbb{V}$	*ie.*	$U(m)(\omega) \rightarrow^* \textcircled{\textcircled{}}$	*si et seulement si*	$M(\omega) \rightarrow^* \textcircled{\textcircled{}}$
$U(m)(\omega) = \mathbb{F}$	*ie.*	$U(m)(\omega) \rightarrow^* \otimes$	*si et seulement si*	$M(\omega) \rightarrow^* \otimes$
$U(m)(\omega) = ?$	*ie.*	$U(m)(\omega) \rightarrow^\infty$	*si et seulement si*	$M(\omega) \rightarrow^\infty$
$U(m)(\omega) = \omega'_1.\omega'_2$	*ie.*	$U(m)(\omega) \rightarrow^* \omega'_1.\omega'_2$ *sur* B_1	*si et seulement si*	$M(\omega) \rightarrow^* \omega'_1.\omega_2$

Conclusion La machine de Turing universelle U est une MT à deux rubans, définie sur l'alphabet $\Sigma_U = \{\square, \$, (, \text{A}, \text{R}, \text{Q}, :, 0, 1), \text{L}, \text{H}, \text{R}\}$. On a vu au TD n° 2 qu'on pouvait se ramener à une MT classique à un seul ruban et on a vu au TD n° 1 qu'on pouvait se ramener à un alphabet réduit à $\{\boxed{0}, \boxed{1}\}$. On est donc capable au moyen de ces 2 transformations de construire un MT classique à un ruban travaillant sur l'alphabet $\{\boxed{0}, \boxed{1}\}$ qui réalise la machine de Turing universelle U. Par contre il est extrémement pénible d'appliquer ces transformations à la main. Les projets signalés dans les pages de ce cours visent à programmer ces transformations afin de pouvoir générer la version classique de la machine de Turing universelle.

4.5 Les projets MT

(2015) Représentation et exécution de MT classique sur $\Sigma = \{\square, \$, 0, 1, \ldots\}$ cf. chapître 1.

(2016) Transformation d'une MT classique définie sur Σ en une MT équivalente définie sur $\{\square, \$, 0, 1\}$, cf. TD n° 1.

(2017) Représentation et exécution de MT à deux rubans, cf. TD n° 2.

(2018) Transformation d'une MT à 2 rubans en MT classique, cf. TD n° 2.

(2019) Représentation d'une MT classique par un mot sur l'alphabet Σ_U, cf. § 4.4.1.

(2020) Réalisation de la machine de Turing universelle U et transformation en une MT classique sur $\{\square, \$, 0, 1\}$, cf. § 4.4.2.

Chapitre 5

Indécidabilité, premiers exemples

Le chapître précédent nous a appris qu'**on pouvait ramener l'étude des fonctions calculables** à l'étude des **langages décidables** c'est-à-dire des ensembles de mots (ou couples de mots) pour lesquels il existe des machines de Turing capables de décider si un mot appartient à l'ensemble ou non. Évidemment cela exige que l'exécution de la machine de Turing sur le mot termine.

Au chapître précédent nous avons également conçu une **machine de Turing universelle** U qui, à partir de la description binaire m d'une MT M, permet d'exécuter M. La machine U est un interpréteur de machines de Turing puisque

$$U(m)(\omega) = M(\omega) \quad \textit{si} \quad m = [M]_2$$

Nous allons dans ce chapître exhiber **deux cas concrets de langages indécidables**, qui s'appuient sur la machine U. À chaque fois il s'agit d'un langage L pour lequel il existe une MT M qui répond $\mathbb{V}$, en un temps fini, si on lui donne un mot ω qui appartient à L. Un néophyte pourrait croire – à tort – que si on dispose d'une telle machine alors on sait alors tester si un mot appartient à L : il suffit d'exécuter M usr ω et de voir si elle répond $\mathbb{V}$. Évidemment, lorsqu'on fait appel à la machine M c'est qu'on ne sait pas si ω appartient à L, c'est justement ce qu'on veut tester.

L'effet de l'indécidabilité apparaît lorsqu'on a appellé M avec un mot ω n'appartenant pas à L : Si, pour certains mots n'appartenant pas à L, la MT M peut répondre $\mathbb{F}$ dans un temps fini, pour d'autres mots n'appartement pas à L, la MT M peut ne pas terminer.

Pourquoi la machine M n'est pas utilisable en pratique ? Supposons qu'on appelle $M(\omega)$ pour tester si $\omega \in L$ et que la réponse se fait attendre ... longtemps.

— Est-ce parce que ω appartient L mais que le calcul pour le découvrir prend du temps ?

— Est-ce parce que ω n'appartient pas à L et que la machine M est partie dans un calcul qui ne finira jamais ?

Que faire dans ce cas ? Faut-il attendre 10 minutes de plus ou interrompre le calcul ? Vous avez tous fait l'expérience d'un ordinateur qui ne répond plus aux commandes, avec un ventilateur en marche et avec un processus qui chauffe. Est-il en train de faire quelque chose d'important qu'il ne faut surtout pas interrompre sous peine de perdre des données ? ou bien, est-il parti dans un calcul inutile qu'il faut arrêter ? **L'indécidabilité vous met face au même dilemne.**

La suite du cours va établir une liste de problèmes apparemment simple qui se révèlent indécidables. Dans ce chapître nous commençons par les deux premiers cas, qui donnent naissance aux autres.

Ce chapître introduit le vocabulaire utilisé dans les ouvrages classiques sur la décidabilité. Certains termes sont peu intuitifs mais historiques.

5.1 Indécidabilité : vocabulaire et définitions

Rappel 1 *Soit Σ un alphabet fini. Σ^* est l'ensemble de tous les mots écrits sur l'alphabet Σ. C'est un ensemble énumérable. Un* ***langage*** *est un sous-ensemble de Σ^*. Dans ce chapître on considère $\Sigma = \{0, 1\}$.*

Définition 21 (Langage récursivement énumérable = reconnaissable par une MT) *Un langage L est* ***récursivement énumérable*** *si il existe une MT M qui reconnaît L*

- *ie. $\mathscr{L}(M) = L$*
- *ie. $M(\omega)$ s'arrête sur un état accepteur si et seulement si $\omega \in L$.*
- *ie. $M(\omega) = \mathbb{V} \iff w \in L$*

  Remarque : En revanche, le comportement de M n'est pas certain pour les mots $\omega \notin L$: M peut ne pas s'arrêter ou bien s'arrêter sur l'état rejet.

Définition 22 (Langage co-récursivement énumérable) *Un langage L est* ***co-récursivement énumérable*** *si et seulement si son langage complémentaire $\overline{L}$ est récursivement énumérable, ie. $\overline{L}$ est reconnaissable.*

Proposition 5 (Langage décidable) *Un langage est* ***décidable*** *au sens de la définition 18*

$\Leftrightarrow$ *il est récursivement énumérable et co-récursivement énumérable*
$\Leftrightarrow$ ***le langage et son complémentaire*** *sont* ***reconnaissables.***

Preuve :

($\Rightarrow$) Ce sens de l'implication est facile à montrer. Il suffit d'appliquer les définitions.

($\Leftarrow$) L est récursivement énumérable signifie qu'il existe une MT M_1 telle que $\mathscr{L}(M_1) = L$.
L est co-récursivement énumérable signifie qu'il existe une MT M_2 telle que $\mathscr{L}(M_2) = \overline{L}$.
Notez que rien ne dit que $M_1 = M_2$. On doit supposer qu'il s'agit de machines de Turing différentes.
On doit montrer que L est décidable, or la définition 18 dit : «Un langage L est décidable s'il existe **une** MT qui termine pour tout mot de $\omega \in \{0, 1\}^*$ sur un état accepteur si $\omega \in L$ et sur un état rejet si $\omega \notin L$». On doit donc construire à partir de M_1 et M_2 une unique MT M qui accepte L et rejette $\overline{L}$.

L'idée On constuit la MT M de la manière suivante : $M(\omega)$ copie le mot ω sur deux bandes B_1 et B_2 puis effectue alternativement une transition de M_1 sur B_1 et une transition de M_2 sur B_2 jusqu'à ce que l'une des deux machines s'arrêtent.

- Si M_1 s'arrête dans l'état ◎ alors, par définition, $M_1(\omega) = \mathbb{V}$ *ie.* $\omega \in \mathscr{L}(M_1) = L$ donc M passe dans un état ◎
- Si M_2 s'arrête dans l'état ◎ alors, par définition, $M_2(\omega) = \mathbb{V}$ *ie.* $\omega \in \mathscr{L}(M_2) = \overline{L}$ donc M passe dans un état $\otimes$

Ordonanceur Pour construire M on a besion d'un ordonanceur (*scheduler* en anglais), noté ${}_1||_1$, qui effectue alternativement une transition de chaque machine, cf. 21.

$$M(\omega) \stackrel{def}{=} \left[\begin{array}{l} B_1 := \omega \ ; \ B_2 := \omega \ ; \\ (m_1, B_1)\,{}_1||_1 (m_2, B_2) \ ; \\ \text{if } (\ \textit{état courant } m_1 = ◎\ ) \text{ then } \rightarrow ◎ \text{ else } \rightarrow \otimes \end{array} \right]$$

□

Exercice 21 ***Ordonanceur***

Étant donné deux machines de Turing décrites par leur codage binaire m_1 et m_2, donnez une MT qui effectue alternativement une transition de m_1 sur B_1 et une transition de m_2 sur B_2 jusqu'à l'arrêt d'une des deux machines.

Proposition 6 (Langage indécidable = non-décidable) *Un langage L est* ***indécidable***

$\Leftrightarrow \neg($ *L est reconnaissable et $\overline{L}$ est reconnaissable*$)$
$\Leftrightarrow$ ***le langage*** L ***ou son complémentaire*** $\overline{L}$ *est* ***non-reconnaissable***

Remarque : C'est juste la négation de la proposition 5.

Exercice 22

Soit $\overline{L}$ un langage indécidable. Appliquez les définitions afin de montrer que L est indécidable.

SOLUTION

Preuve : Il suffit d'appliquer les définitions :

(1) $\overline{L}$ est indécidable $\overset{def}{\iff}$ $\overline{L}$ ou $\overline{\overline{L}}$ est non-reconnaissable

(2) L est indécidable $\overset{def}{\iff}$ L ou $\overline{L}$ est non-reconnaissable

Or, $\overline{\overline{L}} = L$, donc les cas (1) et (3) sont équivalents.

Conclusion : on a démontré plus que ce que demandait l'énoncé : $\overline{L}$ indécidable $\iff$ L indécidable. □

5.2 L'ensemble des codages binaires des machines de Turing

Dans ce chapître on considére l'ensemble $\mathcal{M} \times \{0,1\}^*$ des couples (m,ω) formés d'un mot binaire ω et du codage binaire m d'une MT. C'est un ensemble énumérable, donc un langage et on peut alors s'intéresser à la décidabilité de ses sous-ensembles.

Proposition 7 (L'ensemble $\mathcal{M}$ des codages binaires de machine de Turing est décidable)

$$\mathcal{M} = \{m \in \{0,1\}^* \mid m = [M]_2,\ M \in \text{MT}\} \textit{ est } \textbf{\textit{un langage décidable.}}$$

Preuve : Que doit-on montrer ?

1. **que $\mathcal{M}$ est un langage, *ie.* un sous-ensemble d'un ensemble énumérable :** Il suffit de remarquer que les codages binaires des machines de Turing sont des mots de $\{0,1\}^*$ qui est énumérable.
2. **qu'il existe une MT $M_{\mathcal{M}}$ capable de décider si un mot $\omega \in \{0,1\}^*$ appartient ou non à $\mathcal{M}$, *ie.* décider si ω est bien le codage d'une MT :** On sait que tout langage régulier est reconnaissable par une MT (cf. exercice). On va montrer que les codages binaires de MT forment un langage régulier.
   - Le codage binaire des états peut-être décrit par une expression régulière $ExpReg_{\textit{état}} \overset{def}{=} \texttt{"("} \cdot \{\text{A} \mid \text{R} \mid \text{Q}\} \cdot \texttt{":"} \cdot \{0,1\}^* \cdot \texttt{")"}$
   - Le codage binaire des transitions $\mathbf{q} \xrightarrow{\ell/e:d} \mathbf{q}'$ est reconnaissable par l'expression régulière $ExpReg_{\tau} \overset{def}{=} ExpReg_{\textit{état}} \cdot \Sigma \cdot \texttt{"/"} \cdot \Sigma \cdot \texttt{":"} \cdot \{\text{L},\text{H},\text{R}\} \cdot ExpReg_{\textit{état}}$
   - Le codage binaire d'une MT par « $\mathbf{q}_{init} \cdot \$ \cdot$ *listes des transitions* » est reconnaissable par l'expression régulière $ExpReg_{\mathcal{M}} \overset{def}{=} ExpReg_{\textit{état}} \cdot \texttt{"\$ "} \cdot (ExpReg_{\tau})^*$, d'où $\mathcal{M} = \mathscr{L}(ExpReg_{\mathcal{M}})$
   - Toute expression régulière correspond à un AEF et tout AEF est réalisable par une MT donc il existe $A_{\mathcal{M}}$ équivalent à $E_{\mathcal{M}}$ et $M_{\mathcal{M}}$ équivalente à $A_{\mathcal{M}}$

Conclusion : $\mathcal{M} = \mathscr{L}(E_{\mathcal{M}}) = \mathscr{L}(A_{\mathcal{M}}) = \mathscr{L}(M_{\mathcal{M}})$ est reconnaissable pour une MT $M_{\mathcal{M}}$ □

On donne maintenant deux exemples de langages indécidables : le langage universel et le langage des exécutions finies. Il en existe d'autres qui s'obtiennent en se ramenant à ceux-ci.

5.3 Indécidabilité : premier exemple, preuve directe

Proposition 8 (Le langage universel n'est pas décidable) *Le* ***langage universel*** L_U *est l'ensemble défini par*

$$L_U = \{(m,\omega) \in \mathcal{M} \times \{0,1\}^* \mid m = [M]_2,\ M(\omega) = \mathbb{V}\}$$

C'est l'ensemble des couples (m,ω) tels que la machine m accepte le mot ω.

(i) L_U *est* ***récursivement énumérable****, ie. reconnaissable par une* MT

(ii) L_U *n'est pas* ***co-récursivement énumérable****, ie.* $\overline{L_U}$ *n'est pas reconnaissable par une* MT

(iii) L_U *n'est* ***pas décidable****.*

Preuve :

(i) On doit montrer qu'il existe une MT qui reconnaît L_U : la MT cherchée c'est U. En effet,
$\mathscr{L}(U) \stackrel{def}{=} \{(m,\omega) \mid U(m,\omega) = \mathbb{V}\}$ par définition du langage reconnu par une MT.
or $U(m,\omega) = U(m)(\omega) = M(\omega)$ avec $m = [M]_2$ par définition de la U
donc $\mathscr{L}(U) = \{(m,\omega) \mid M(\omega) = \mathbb{V},\ m = [M]_2\} \stackrel{def}{=} L_U$ d'après la définition de L_U.
Conclusion : $\mathscr{L}(U) = L_U$ ce qui signifie que la machine universelle U reconnaît le langage universel L_U.

(ii) On va montrer par contradiction qu'il n'existe pas de MT qui reconnaisse $\overline{L_U}$.

Que représente $\overline{L_U} \stackrel{def}{=} (\mathcal{M} \times \{0,1\}^*) \setminus L_U$ **?** Les éléments de $\overline{L_U}$ sont les couples (m,ω) que la machine universelle U n'accepte pas.

$$\overline{L_U} = \{(m,\omega) \in \mathcal{M} \times \{0,1\}^* \mid U(m)(\omega) \neq \mathbb{V}\}$$

Preuve de (ii) par contradiction : SUPPOSONS qu'il existe une MT $M_{\overline{L_U}}$ qui reconnaisse $\overline{L_U}$. On peut l'utiliser pour construire une MT $M_C(\omega) \stackrel{def}{=} M_{\overline{L_U}}(\omega,\omega)$ qui duplique le mot binaire ω pour en faire un couple et exécute $M_{\overline{L_U}}$ sur ce couple.

$$\begin{array}{rcll}
\mathscr{L}(M_C) & = & \{\omega \mid M_C(\omega) = \mathbb{V}\} & \textit{par définition du langage reconnu par une MT} \\
& = & \{\omega \mid M_{\overline{L_U}}(\omega,\omega) = \mathbb{V}\} & \textit{par définition de } M_C \\
& = & \{\omega \mid (\omega,\omega) \in \overline{L_U}\} & \textit{par définition du langage reconnu par une MT} \\
& & & \textit{donc } \omega \textit{ n'est pas un mot quelconque de } \{0,1\}^* \textit{ mais un élément de } \mathcal{M} \\
& = & \{m \in \mathcal{M} \mid (m,m) \in \overline{L_U}\} & \\
& = & \{m \in \mathcal{M} \mid U(m)(m) \neq \mathbb{V}\} & \textit{par définition de } \overline{L_U} \\
\mathscr{L}(M_C) & = & \{m \in \mathcal{M} \mid m = [M]_2,\ M(m) \neq \mathbb{V}\} & \textit{par définition de la MT universelle}
\end{array}$$

$\mathscr{L}(M_C)$ est donc l'ensemble des mots binaires de $\mathcal{M}$ qui correspondent à *des* MT *qui n'acceptent pas, en tant que mot, leur description binaire, ie.* $M(m) \rightarrow^* \otimes \ \vee\ M(m) \rightarrow^\infty$.

Exhibons la contradiction : Considérons maintenant m_c le codage binaire de la MT M_C que l'on vient de construire.
On peut alors se demander si m_c **appartient à** $\mathscr{L}(M_C)$ **?**

$$\begin{array}{rcll}
m_c \in \mathscr{L}(M_C) & \iff & m_c \in \{m \in \mathcal{M} \mid m = [M]_2,\ M(m) \neq \mathbb{V}\} & \textit{par définition de } \mathscr{L}(M_C) \\
& \iff & M_c(m_c) \neq \mathbb{V} & \textit{puisque } m_c = [M_c]_2
\end{array}$$

Ainsi,

$$(\dagger) \quad m_c \in \mathscr{L}(M_C) \iff M_C(m_c) \neq \mathbb{V}$$

Par ailleurs,

$$(\ddagger) \quad m_c \in \mathscr{L}(M_C) \iff M_C(m_c) = \mathbb{V} \quad \textit{par définition du langage reconnu par une MT}$$

Les équivalences $(\dagger)$ et $(\ddagger)$ donnent la CONTRADICTION cherchée.

Conclusion : En supposant qu'il existait une MT qui reconnaît $\overline{L_U}$ nous aboutissons à une contradiction. Donc $\overline{L_U}$ est indécidable, ce qui termine la preuve de (ii).

(iii) D'après la propostion 5 un langage L est décidable si et seulement si L et $\overline{L}$ sont reconnu par une MT. $\overline{L_U}$ n'étant pas reconnaissable par une MT, cf. (ii). L_U n'est pas décidable.

□

Exercice 23 *Preuve de l'indécidabilité de* L_{EF}

En suivant exactement le même schéma de preuve que pour le langage universel, complétez la preuve que L_{EF} est reconnaissable par un MT et que $\overline{L_{EF}}$ n'est pas reconnaissable par une MT.

5.4 Indécidabilité : second exemple, preuve directe

Proposition 9 (Le langage des exécutions finies n'est pas décidable) *Le* ***langage des exécutions finies*** L_{EF} *est l'ensemble défini par*

$$L_{EF} = \{(m,\omega) \in \mathcal{M} \times \{0,1\}^* \mid U(m)(\omega) \not\to \infty\}$$

C'est l'ensemble des couples (m,ω) *tels que l'exécution de la machine* m *termine quand on l'appelle avec le mot* ω.

(i) L_{EF} *est* ***récursivement énumérable****, ie. reconnaissable par une* MT

(ii) L_{EF} *n'est* ***pas co-récursivement énumérable****, ie.* $\overline{L_{EF}}$ *n'est pas reconnaissable.*

Preuve :

(i) Montrons L_{EF} reconnaissable : Montrons qu'il existe une MT M_{EF} qui reconnaît L_{EF}, *ie.* $\mathscr{L}(M_{EF}) = L_{EF}$, *ie.* $M_{EF}(m,\omega) = \mathbb{V} \iff (m,\omega) \in L_{EF}$, *ie.* $M_{EF}(m,\omega) = \mathbb{V} \iff U(m)(\omega) \not\to \infty$.

La MT M_{EF} doit s'arrêter dans un état accepteur pour tout couple (m,ω) de L_{EF}, c'est-à-dire pour les couples qui correspondent à des exécutions finies. M_{EF} consiste à exécuter $U(m)(\omega)$ – le résultat nous importe peu – puis à passer dans l'état accepteur ◎. Puisque les couples de L_{EF} sont précisement les couples pour lesquels l'exécution de U s'arrête, on a la garantie que la MT M_{EF} ci-dessous terminera dans l'état ◎ pour les couples de L_{EF}.

$$M_{EF}(m,\omega) \stackrel{def}{=} [\ U(m)(\omega)\ ;\ \to ◎\ ]$$

(ii) Montrons $\overline{L_{EF}}$ non-reconnaissable : On va montrer par contradiction qu'il n'existe pas de MT qui reconnaisse $\overline{L_{EF}}$.

Que représente $\overline{L_{EF}}$? $\overline{L_{EF}} \stackrel{def}{=} (\mathcal{M} \times \{0,1\}^*) \setminus L_{EF}$ Les éléments de $\overline{L_{EF}}$ sont les couples (m,ω) sur lesquels que la machine universelle U ne termine pas.

$$\overline{L_{EF}} \stackrel{def}{=} (\mathcal{M} \times \{0,1\}^*) \setminus L_{EF} = \{(m,\omega) \in \mathcal{M} \times \{0,1\}^* \mid U(m)(\omega) \to \infty\}$$

Preuve de (ii) par contradiction : SUPPOSONS qu'il existe une MT $M_{\overline{EF}}$ qui reconnaisse $\overline{L_{EF}}$. On peut l'utiliser pour construire une MT $M_C(\omega) \stackrel{def}{=} M_{\overline{EF}}(\omega,\omega)$ qui duplique le mot binaire ω pour en faire un couple et exécute $M_{\overline{EF}}$ sur ce couple.

$$\begin{array}{rcll}
\mathscr{L}(M_C) & = & \{\omega \mid M_C(\omega) = \mathbb{V}\} & \textit{par définition du langage reconnu par une MT} \\
& = & \{\omega \mid M_{\overline{EF}}(\omega,\omega) = \mathbb{V}\} & \textit{par définition de } M_C \\
& = & \{\omega \mid (\omega,\omega) \in \overline{L_{EF}}\} & \textit{par définition du langage reconnu par une MT} \\
& & & \textit{donc } \omega \textit{ n'est pas un mot quelconque de } \{0,1\}^* \textit{ mais un élément de } \mathcal{M} \\
& = & \{m \in \mathcal{M} \mid (m,m) \in \overline{L_{EF}}\} & \\
& = & \{m \in \mathcal{M} \mid U(m)(m) \to \infty\} & \textit{par définition de } \overline{L_{EF}} \\
\mathscr{L}(M_C) & = & \{m \in \mathcal{M} \mid m = [M]_2,\ M(m) \to \infty\} & \textit{par définition de la MT universelle}
\end{array}$$

$\mathscr{L}(M_C)$ est donc l'ensemble des mots binaires de $\mathcal{M}$ qui correspondent à *des* MT *qui ne s'arrête pas lorsqu'on les exécute sur leur description binaire.*

Exhibons la contradiction : Considérons maintenant m_c le codage binaire de la MT M_C que l'on vient de construire. **On peut alors se demander si m_c appartient à $\mathscr{L}(M_C)$?**

$$\begin{array}{rcll}
m_c \in \mathscr{L}(M_C) & \iff & m_c \in \{m \in \mathcal{M} \mid m = [M]_2,\ M(m) \to \infty\} & \textit{par définition de } \mathscr{L}(M_C) \\
& \iff & M_c(m_c) \to \infty & \textit{puisque } m_c = [M_c]_2
\end{array}$$

Ainsi, (†) $m_c \in \mathscr{L}(M_C) \iff M_C(m_c) \to \infty$

Par ailleurs, par définition du langage reconnu par une MT, on a aussi l'équivalence :

(‡) $m_c \in \mathscr{L}(M_C) \iff M_C(m_c) = \mathbb{V} \iff M_C(m_c) \to^* ◎$

Les équivalences (†) et (‡) donnent la CONTRADICTION cherchée puisque l'exécution $M_C(m_c)$ est censée terminer (dans l'état ◎) d'après (‡), et ne pas terminer d'après (‡).

Conclusion : En supposant qu'il existait une MT qui reconnaît $\overline{L_{EF}}$ nous aboutissons à une contradiction. Donc $\overline{L_{EF}}$ est indécidable, ce qui termine la preuve de (ii).

□

5.5 D'où vient l'indécidabilité ?

Certrains pensent que l'indécidabilité apparaît du fait que les mt peuvent avoir des exécutions infinies : ils ont tort. Si vous relisez la preuve précédente vous verrez que l'existence d'exécution infinie ne joue aucun rôle. L'argument principal qui conduit à l'indécidabilité est la possibilité qu'une mt M prenne en argument son propre code m. On retrouve un argument de diagonalisation à la Cantor puisque parmi les couples (m, ω) on s'intéresse au couple (m, m), auxquelles on applique U pour obtenir $U(m)(m) = M(m)$.
Cette capacité à s'auto-analyser, s'auto-modifier, s'auto-répliquer apporte à l'informatique la capacité de s'auto-vérifier, s'adapater, de créer des virus, *etc.* La contre-partie de cette puissance est que la plupart des questions non-triviales qu'on se pose sur les programmes *(terminent-ils ? sont-ils bogués ?)* ne sont pas décidables ... par des programmes.

Chapitre 6

Principe de réduction & applications

6.1 Principe de réduction

Le **principe de réduction** permet de **relier la décidabilité / l'indécidabilité de deux langages**, disons L et L' au moyen d'une traduction qui réduit la question d'appartenance à L à la question de l'appartenance à L'. **Prenez garde au sens des implications de la proposition 11.**

Définition 23 (Diagramme de réduction) *Soit L un langage sur l'alphabet Σ et L' un langage sur l'alphabet Σ'. Un* ***diagramme de réduction de L à L'*** *est un* ***schéma*** *comme suit,* ***accompagné***

$$\begin{array}{ccc} \omega \in \Sigma^* & \xrightarrow[traduction]{M_R} & R(\omega) = \omega' \in \Sigma'^* \\ \omega \in L & \underset{(\ddagger)}{\iff} & R(\omega) \in L' \end{array}$$

1. *d'une* MT M_R *qui termine* ***pour tout mot*** $\omega \in \Sigma^*$ *et traduit ω en un mot de Σ'^*.*
   On note $R(\omega)$ le résultat de $M_R(\omega)$. La traduction doit être bien choisie de sorte qu'on ait l'équivalence $(\ddagger)$ qui relie l'appartenance d'un mot à L à l'appartenance de sa traduction à L'
2. *et de la* ***preuve de l'équivalence*** $(\ddagger)$

Proposition 10 (Diagramme de réduction complémentaire) *Le même diagramme de réduction est valable pour les complémentaires de L et L'.*

$$\begin{array}{ccc} \omega \in \Sigma^* & \xrightarrow[traduction]{M_R} & R(\omega) = \omega' \in \Sigma'^* \\ \omega \in \overline{L} & \underset{(\ddagger)}{\iff} & R(\omega) \in \overline{L'} \end{array}$$

Preuve : Il suffit de montrer que l'équivalence $(\ddagger)$ est valide pour les complémentaires de L et de L'.

$$\begin{array}{rccl} & \omega \in L & \underset{(\ddagger)}{\iff} & R(\omega) \in L' \\ (\ddagger) \textit{ est une équivalence donc } & \neg(\omega \in L) & \iff & \neg(R(\omega) \in L') \\ & \equiv \omega \notin L & & \equiv R(\omega) \notin L' \\ & \equiv \omega \in \overline{L} & \underset{(\ddagger)}{\iff} & R(\omega) \in \overline{L'} \end{array}$$

□

Proposition 11 *Un diagramme de réduction comme celui de la définition 23 permet de conclure :*

(i)	L *reconnaissable*	$\Longleftarrow$	L' *reconnaissable*	
(ii)	L *décidable*	$\Longleftarrow$	L' *décidable*	
(iii)	L *non-reconnaissable*	$\Longrightarrow$	L' *non-reconnaissable*	*: c'est la contraposé de* (i)
(iv)	L *indécidable*	$\Longrightarrow$	L' *indécidable*	*: c'est la contraposé de* (ii)

(v) (i), (ii), (iii), (iv) *sont valables en remplaçant L et L' par leurs complémentaires $\overline{L}$ et $\overline{L'}$*

Preuve :

(i) **Le principe de réduction repose sur la construction suivante :** Supposons L' reconnaissable par une MT M' et utilisons M' pour construire une MT M qui reconnaît L.

L'hypothèse « M' reconnaît L' » signifie $M'(\omega') = \mathbb{V} \underset{(1)}{\iff} \omega' \in L'$.

Prenons $M \overset{def}{=} [M_R \ ; \ M']$. **On doit montrer** $M(\omega) = \mathbb{V} \overset{?}{\iff} \omega \in L$.

$$\begin{array}{ccc} M(\omega) = \mathbb{V} & \overset{\text{OK}}{\iff} & w \in L \\ \Updownarrow \text{ def} & & \Uparrow \\ \omega \xrightarrow{M_R} R(\omega) \xrightarrow{M'} \textcircled{\textcircled{}} & & \text{d'après } (\ddagger) \\ & & \Downarrow \\ M'(R(\omega))=\mathbb{V} & \underset{(1)}{\iff} & R(\omega) \in L' \end{array}$$

(ii) **On doit montrer** L reconnaissable et $\overline{L}$ reconnaissable. L'hypothèse « L' décidable » signifie L' reconnaissable et $\overline{L'}$ reconnaissable. D'après (i) L' reconnaissable implique L reconnaissable. Il reste à montrer $\overline{L}$ reconnaissable : il suffit d'appliquer (i) au diagramme des complémentaires de la proposition 10, pour obtenir « $\overline{L'}$ reconnaissable implique $\overline{L'}$ reconnaissable ». Mais alors L est décidable puisque L et $\overline{L}$ sont reconnaissables.

(iii) et (iv) s'obtiennent comme contraposé de (i) et (ii) : $B \Longleftarrow A \equiv$ *par contraposé* $\equiv \neg B \Longrightarrow \neg A$

Preuve par contradiction: Montrons que de $A \Longrightarrow B$ et de $\neg B$ on peut déduire $\neg A$. SUPPOSONS A. En utilisant l'implication $A \Longrightarrow B$, on peut en déduire B. Ce qui contredit l'hypothèse $\neg B$. CONCLUSION : En supposant A on aboutit à une contradiction, ce qui prouve $\neg A$. □

(iii) peut aussi se démontrer directement par contradiction

SUPPOSONS L' reconnaissable (c'est A) alors on peut construire $M \overset{def}{=} [M_r \ ; \ M']$ qui reconnaît L comme au (i), donc L serait reconnaissable (c'est B). Or, (iii) fait l'hypothèse que L est non-reconnaissable (c'est $\neg B$), on aboutit donc à une CONTRADICTION. Conclusion : L' est non-reconnaissable (c'est $\neg A$).

(iv) peut aussi se démontrer directement : **On doit montrer** L' indécidable *ie.* L' **ou** $\overline{L'}$ non-reconnaissable. L'hypothèse « L indécidable » signifie que L ou $\overline{L}$ est non-reconnaissable. Examinons les deux cas :

(a) Si c'est L qui est non-reconnaissable alors L' est non-reconnaissable d'après (iii).

(b) Si c'est $\overline{L}$ qui est non-reconnaissable : il suffit d'appliquer (iii) au diagramme des complémentaires de la proposition 10, pour obtenir « $\overline{L}$ non-reconnaissable implique $\overline{L'}$ non-reconnaissable ».

Dans les deux cas, L' est indécidable puisque soit L', soit $\overline{L'}$ est non-reconnaissable.

(v) D'après la proposition 10 le diagramme de réduction est valide pour les complémentaires de L et L'. Les résultats précédents (i), (ii), (iii), (iv) qui reposent sur le diagramme de réduction sont donc applicables à $\overline{L}$ et $\overline{L'}$.

□

6.2 Applications du principe de réduction

6.2.1 Réduction de L_{EF} à L_{TT}

Proposition 12 (L'ensemble des machines de Turing qui Terminent Toujours est indécidable) *Le langage des machines de Turing qui terminent toujours L_{TT} est défini par*

$$L_{TT} = \{m' \in \mathcal{M} \mid \forall \omega' \in \{0,1\}^*, \ U(m', \omega') \not\to \infty\}$$

C'est l'ensemble des codages binaires de MT *dont les exécutions terminent quel que soit le mot binaire en entrée. Le langage L_{TT} est indécidable puisque son complémentaire $\overline{L_{TT}}$ n'est pas reconnaissable.*

Remarques

1. $\overline{L_{TT}}$ est l'ensemble des codages binaires m de MT tels qu'il existe au moins une entrée sur laquelle l'exécution de m ne termine pas.

$$\overline{L_{TT}} = \mathcal{M} \setminus L_{TT} = \{m' \in \mathcal{M} \mid \exists \omega' \in \{0,1\}^*, \ U(m', \omega') \to \infty\}$$

2. On va montrer qu'**on peut réduire la question** de l'appartenance à $\overline{L_{EF}}$ à celle de l'appartenance à $\overline{L_{TT}}$. On pourrait alors conclure, d'après le (iv) de la proposition 11, que L_{TT} est indécidable en s'appuyant sur le fait connu que L_{EF} est indécidable. Nous allons être plus précis et montrer pourquoi L_{TT} est indécidable : la raison est que $\overline{L_{TT}}$ est non-reconnaissable.
3. L'ensemble $\overline{L_{EF}}$ des exécutions infinies est l'ensemble des couples (m, ω) tels que l'exécution de la MT m sur le mot w ne termine pas.

$$\overline{L_{EF}} = (\mathcal{M} \times \{0,1\}^*) \setminus L_{EF} = \{\,(m,\omega) \in \mathcal{M} \times \{0,1\}^* \mid U(m,\omega) \to \infty\,\}$$

Preuve de la proposition 12 – montrons que $\overline{L_{TT}}$ est non-reconnaissable:
$\overline{L_{EF}}$ est un ensemble de couples $(m,\omega) \in \mathcal{M} \times \{0,1\}^*$ et $\overline{L_{TT}}$ est un ensemble de codages binaires de MT $m \in \mathcal{M}$. Un **diagramme de réduction de $\overline{L_{EF}}$ à $\overline{L_{TT}}$** est donc de la forme

$$\begin{array}{ccc} (m,\omega) \in \mathcal{M} \times \Sigma^* & \xrightarrow[traduction]{M_R} & R(m,\omega) = m' \in \mathcal{M} \\ (m,\omega) \in \overline{L_{EF}} & \underset{(\ddagger)}{\Longleftrightarrow} & R(m,\omega) \in \overline{L_{TT}} \end{array}$$

Que doit-on faire ?

1. **On doit définir une** MT M_R qui traduit en un temps fini tout couple (m,ω) en une MT $m' \overset{def}{=} R(m,\omega)$.
2. **On doit montrer** que l'exécution de M_R termine pour tout couple (m,ω) de $\mathcal{M} \times \{0,1\}^*$.
3. **On doit ensuite démontrer l'équivalence** $(\ddagger)$ c'est-à-dire que l'exécution $U(m,\omega)$ ne termine pas si et seulement si la MT $R(m,\omega)$ ne termine pas pour une certaine entrée.

Allons-y

1. Étant donné (m,ω) avec $\omega = s_1.s_2.\ldots.s_n$, **la MT M_R constuit la réduction** $m' \overset{def}{=} R(m,\omega)$ de la façon suivante :

$$m' = M_R(m,\omega) \overset{def}{=} \underbrace{[\ M_{eff}\ ;\ \mathbf{q}_0 \xrightarrow{\square/s_1:R} \mathbf{q}_1 \xrightarrow{\square/s_2:R} \ldots \xrightarrow{\square/s_n:R} \mathbf{q}_n\ ;\ M_{\overleftarrow{\$}}\ ]_2\ ;\ m}_{\textit{codage binaire d'une MT}}$$

À partir de m et ω, la M_R produit un codage binaire m' d'une MT qui effface la donnée inscrite sur le ruban, puis inscrit le mot $\omega = s_1.s_2.\ldots.s_n$ symbole par symbole à la place, puis se replace sur le symbole \$ de début de ruban, et enfin lance la machine m sur ce ruban (donc sur le mot ω). La machine m' ainsi produite ignore la donnée d'entrée inscrite sur le ruban et exécute en fait $U(m,\omega)$.

2. **La MT M_R termine pour tout couple** $(m,\omega) \in \mathcal{M} \times \{0,1\}^*$: en effet, elle laisse m intacte sur le ruban ; elle copie le codage binaire de $[\ M_{eff}\ ;\ \mathbf{q}_0 \xrightarrow{\square/s_1:R} \mathbf{q}_1 \xrightarrow{\square/s_2:R} \ldots \xrightarrow{\square/s_n:R} \mathbf{q}_n\ ;\ M_{\overleftarrow{\$}}\ ]$ avant m or la taille de ce codage binaire dépend uniquement de la taille de ω qui est un mot fini.
3. **Démontrons l'équivalence** $(\ddagger)$ Examinons l'exécution de $R(m,w)$ sur un mot d'entrée $\omega' = \ell_1.\ldots.\ell_k$ à k symboles :

$$\underset{\uparrow}{\$}.\underbrace{\ell_1.\ldots.\ell_k}_{\textit{entrée}} \xrightarrow{M_{eff}} \underset{\uparrow}{\$}.\square \xrightarrow{\square/s_1:R} \xrightarrow{\square/s_2:R} \ldots \xrightarrow{\square/s_n:R} \underbrace{s_1.\ldots.s_n}_{\omega}.\underset{\uparrow}{\square} \xrightarrow{M_{\overleftarrow{\$}}} \underset{\uparrow}{\$}.\omega \xrightarrow{U(m)} \ldots$$

Puisque le mot d'entrée est fini (de taille k), la MT M_{eff} termine (en $2k$ étapes) ; les n transitions qui inscrivent ω sur le ruban terminent en n étapes ; la MT $M_{\overleftarrow{\$}}$ retrouve le \$ de début de ruban en $n+1$ étapes. Toutes ces instructions terminent et **l'exécution ne peut éviter la dernière instruction qui consiste à exécuter $U(m)$ sur le mot ω.**

Finalement, **exécuter $m' \overset{def}{=} R(m,\omega)$ sur le mot ω' revient à exécuter $U(m,\omega)$.** Utilisons cette observation pour démontrer $(\ddagger)$:

$(\Rightarrow)$ $(m,\omega) \in \overline{L_{EF}}$ signifie que l'exécution $U(m,\omega)$ ne termine pas, donc il existe au moins une entrée (en fait toute) sur laquelle $m' \overset{def}{=} R(m,\omega)$ ne termine pas et donc $R(m,\omega) \in \overline{L_{TT}}$.

$(\Leftarrow)$ $R(m,\omega) \in \overline{L_{TT}}$ signifie que il existe au moins une entrée ω' sur laquelle $m' \overset{def}{=} R(m,\omega)$ ne termine pas. Or, quel que soit ω' l'exécution de m' finit inévitablement par exécuter $\$MU(m,\omega)$. Donc, c'est $U(m,\omega)$ la cause de la non-terminaison et donc $(m,\omega) \in \overline{L_{EF}}$.

$\square$

6.2.2 Réduction de L_{EF} à PCP

En td vous verrez un autre exemple de réduction de L_{EF} vers un probléme très différent « l'existence d'une solution au Problème des Correspondances de Post (pcp)».

Remarques

1. Dans les exemples de langages indécidables traités jusqu'à présent on a pris soin de bien identifier la raison de l'indécidabilité : est-ce L qui est non-reconnaissable ? ou bien $\overline{L}$? ou bien les deux ?
2. **En pratique** on se moque de savoir lequel de L ou $\overline{L}$ est non-reconnaissable puisqu'il faut que le **problème soit décidable pour pouvoir donner un algorithme qui termine toujours et réponde $\mathbb{V}$ ou $\mathbb{F}$.**
3. Un **algorithme** qui répond $\mathbb{V}$ quand l'entrée $\omega \in L$ mais qui **peut boucler** quand l'entrée $\omega \in \overline{L}$ est intéressant du point de vue théorique mais peu utile en pratique. En effet, lorsqu'on lui donne un mot ω et qu'il n'a toujours pas répondu au bout d'1 min... de 10 min... d'1h... de 10h... que conclure ? Peut-être que $\omega \in L$ mais que cette vérification demande beaucoup de calcul et qu'il faut attendre encore un peu pour avoir la réponse, ou bien la mt est en train de boucler ; on n'a aucun moyen de le savoir.
4. Dans la suite nous nous contentons donc d'indiquer si le problème est décidable ou indécidable, sans préciser en cas d'indécidabilité si c'est le langage ou son complémentaire qui est non-reconnaissable.

Chapitre 7

Théorème de Rice & applications

7.1 Théorème de Rice

7.1.1 Version classique du théorème de Rice

Définition 24 (Langages reconnaissables) *Étant donné un alphabet Σ. On note $\mathcal{L}$ l'ensemble des langages reconnaissables par les machines de Turing :*

$$\mathcal{L} = \{\mathscr{L}(M) \mid M \in \text{MT}\}$$

Théorème 2 (de Rice, version langage) *Soit $\mathcal{P}_\mathcal{C}$ un sous-ensemble non-trivial de $\mathcal{L}$, défini comme l'ensemble des langages reconnaissables qui satisfont une* ***condition non-triviale*** *$\mathcal{C}$, ie.*

$$\mathcal{P}_\mathcal{C} = \{L \in \mathcal{L} \ \mid \ \mathcal{C}(L)\}$$

alors ***l'appartenance d'un langage à $\mathcal{P}_\mathcal{C}$ est indécidable, ie. $\mathcal{P}_\mathcal{C}$ ou $\overline{\mathcal{P}_\mathcal{C}}$ est non-reconnaissable.***

Remarques

1. La condition $\mathcal{C}$ est un prédicat sur les langages, *ie.* $\mathcal{C} : \mathcal{L} \to \mathit{Bool}$.
2. Une condition est triviale si elle ne dépend pas de son paramètre. Il n'y a donc que deux conditions $\mathcal{C}$ triviales : celle qui vaut toujours $\mathbb{V}$, dans ce cas $\mathcal{P}_\mathcal{C} = \mathcal{L}$ et celle qui vaut toujours faux, dans ce cas $\mathcal{P}_\mathcal{C} = \emptyset$.
3. la condition $\mathcal{C}$ est non-triviale $\Leftrightarrow$ l'ensemble $\mathcal{P}_\mathcal{C}$ est non-trivial,
   ie. $\mathcal{C} \neq \mathbb{V}$ et $\mathcal{C} \neq \mathbb{F}$ *ie.* $\mathcal{P}_\mathcal{C} \neq \mathcal{L}$ et $\mathcal{P}_\mathcal{C} \neq \emptyset$.

Avant de donner la preuve de ce théorème, pour en comprendre la portée et les conséquences nous allons le reformuler en terme de machines de Turing.

7.1.2 Version machine de Turing du théorème de Rice

Au lieu de considérer $\mathcal{L}$, l'ensemble des langages reconnaissables, on peut énoncer le théorème de Rice en considérant $\mathcal{M}$, l'ensemble des codes de machines de Turing. C'est cette version que nous utiliserons dans les exemples et que nous allons démontrer.

Rappel m est le codage binaire de la MT M signifie $m = [M]_2$ et $U(m) = M$ et $U(m)(\omega) = M(\omega)$

Définition 25 (Ensemble de Rice) *Un ensemble de Rice, noté $\mathcal{P}_\mathcal{C}$ est un sous-ensemble de $\mathcal{M}$, défini comme l'ensemble des codes m de* MT *qui satisfont une* ***condition non-triviale*** *$\mathcal{C} : \mathcal{L} \to \mathit{Bool}$* ***sur le langage reconnu par*** *m, ie. un $\mathcal{P}_\mathcal{C}$ est de la forme*

$$\{m \in \mathcal{M} \ \mid \ \mathcal{C}(\mathscr{L}(U(m))\}$$

Remarque : $\mathscr{L}(U(m))$ désigne le langage reconnu par la MT de code m.

$$\begin{array}{lrcl} \text{En effet,} & \mathscr{L}(U(m)) & = & \{\omega \in \Sigma^* \mid U(m)(\omega) = \mathbb{V}\} \\ & & = & \{\omega \mid M(\omega) = \mathbb{V}\} \textit{ puisque } U(m)(\omega) = M(\omega) \textit{ où } m = [M]_2 \\ & & = & \mathscr{L}(M) \end{array}$$

Théorème 3 (de Rice, version MT) *Soit $\mathcal{P}_\mathcal{C}$ un ensemble de Rice défini par une* ***condition non-triviale*** *$\mathcal{C} : \mathcal{L} \to Bool$* ***sur le langage reconnu par un code*** *m* ***de machine de Turing****, ie.*

$$\mathcal{P}_\mathcal{C} = \{m \in \mathcal{M} \mid \mathcal{C}(\mathscr{L}(U(m))\}$$

Alors ***l'appartenance d'une*** MT *m* ***à $\mathcal{P}_\mathcal{C}$ est indécidable, ie. $\mathcal{P}_\mathcal{C}$ ou $\overline{\mathcal{P}_\mathcal{C}}$ est non-reconnaissable****.*

7.1.3 Applications du théorème de Rice

1. $\mathcal{P}_1 = \{m \mid U(m)(1) = \mathbb{V}\}$ correspond à la condition $\mathcal{C} : \mathcal{L} \to Bool$ définie par $\mathcal{C}(L) \stackrel{def}{=} 1 \in L$. D'après le Théorème de Rice, $\mathcal{P}_1$ est indécidable ; en fait, $\overline{\mathcal{P}_1}$ est non-reconnaissable. En français cela donne : L'ensemble des machines de Turing qui acceptent le mot binaire 1 est indécidable.
2. On peut généraliser l'exemple précédent à n'importe quel mot ω. L'ensemble des MT $\mathcal{P}_\omega = \{m \mid U(m)(\omega) = \mathbb{V}\}$ correspond à la condition $\mathcal{C}(L) \stackrel{def}{=} \omega \in L$. D'après le Théorème de Rice, $\mathcal{P}_\omega$ est indécidable ; en fait, $\overline{\mathcal{P}_\omega}$ est non-reconnaissable. En français cela donne : Étant donné un mot ω, l'ensemble $\mathcal{P}_\omega$ des MT qui acceptent le mot binaire ω est indécidable.
3. $\mathcal{P}_{reg} = \{m \mid \exists Aut \in \text{AEF},\ \mathscr{L}(Aut) = \mathscr{L}(U(m))\}$ correspond à la condition $\mathcal{C}(L) \stackrel{def}{=} \exists Aut \in \text{AEF},\ \mathscr{L}(A) = L$. D'après le Théorème de Rice, $\mathcal{P}_{reg}$ est indécidable. En français cela donne : L'ensemble des machines de Turing régulières, c'est-à-dire les MT dont le langage est régulier, est indécidable.
4. $\mathcal{P}_\emptyset = \{m \mid \mathscr{L}(U(m)) = \emptyset\}$ correspond à la condition $\mathcal{C}(L) \stackrel{def}{=} L = \emptyset$. D'après le Théorème de Rice, $\mathcal{P}_\emptyset$ est indécidable. En français cela donne : L'ensemble des machines de Turing qui n'acceptent aucun mot est non-reconnaissable.
5. $\mathcal{P}_{fini} = \{m \mid |\mathscr{L}(U(m))| \in \mathbb{N}\}$ correspond à la condition $\mathcal{C}(L) \stackrel{def}{=} |L| \in \mathbb{N}$ D'après le Théorème de Rice, $\mathcal{P}_{fini}$ est indécidable. En français cela donne : L'ensemble des machines de Turing dont le langage est fini, est indécidable.

7.1.4 Mais alors que reste-t'il de décidable ?

Les propriétés décidables sur $\mathcal{M}$ sont celles qui ne s'intéressent pas seulement sur le résultat, la terminaison ou le langage reconnu mais qui porte aussi sur la structure des MT (ou la longueur de l'exécution).

Exemples :

- $\{m \mid U(m)(\omega) \xrightarrow{\leq 1000} \omega'\}$ l'ensemble des MT dont l'exécution sur ω termine en moins de 1000 pas de calcul.
- L'ensemble des MT qui ne contiennent pas de boucle.
- L'ensemble des MT dont les boucles sont bornées par des constantes, *ie.* for i = $N..M$ avec N et M constant.

Remarques

1. Puisque les MT correspondent aux fonctions calculables, elles représentent tous les algorithmes, indépendamment du langage de programmation choisi. **Les raisonnements sur les limitations des MT s'appliquent donc aux algorithmes.**
2. **La plupart des propriétés intéressantes sont indécidables.** La source du problème est qu'il n'existe pas d'algorithme capable de décider pour tout programme qu'on lui donnerait en entrée s'il termine ou non.
3. Toutefois, en restreignant la forme des programmes qu'on prend en entrée, on est capable de décider certaines classes de programmes. On parvient ainsi à mieux cerner la frontière entre décidable et indécidable. On sait par exemple quelles restrictions imposer sur les langages de programmation afin

(a) d'éliminer les fuites de mémoire et les segmentation fault – grâce au ramasse-miettes (*garbage collector* en anglais) ;

(b) d'éliminer les erreurs de type ou d'écrasement mémoire – grâce aux typages forts comme en CAML ou ADA ;

(c) de garantir la terminaison des programmes par un typage encore plus fort que celui de CAML

Malgré cela les programmeurs continuent de programmer dans des langages qui n'offrent aucune garantie et ils se retrouvent à devoir passer 60% de leur temps à tester et à debogguer.

4. D'autre part, on utilise en pratique des algorithmes semi-décidables (qui peuvent ne pas terminer) en leur ajoutant un minuteur qui interrompt l'exécution au delà d'un certain délai. Si l'algorithme termine avant le délai imparti il donne une réponse ; si l'algorithme est interrompu il conclut par "je ne sais pas".

7.1.5 Théorème de Rice et MT équivalentes

Définition 26 (Équivalence de machines de Turing) *Deux* MT M_1 *et* M_2 *sont équivalentes si et seulement si* $\mathscr{L}(M_1) = \mathscr{L}(M_2)$ *ie. si elles reconnaissent le même langage.*

Remarques

1. L'équivalence portent sur **les résultats** des exécutions de M_1 et M_2 et pas sur leur code ; ce qu'on pourrait énoncer de manière caricaturale par «***en calculabilité**, peu importe le programme, du moment qu'il donne le résultat souhaité*».
2. Au contraire, ***en complexité**, pour un même résultat final, on s'intéresse au code le plus efficace*. Dans ce cas on cherchera à comparer les codes.

Rappel m est le codage binaire de la MT M signifie $m = [M]_2$ et $U(m) = M$ et $U(m)(\omega) = M(\omega)$

Proposition 13 (Ensemble de Rice et MT équivalentes) *Étant donné un ensemble de Rice* $\mathcal{P}_\mathcal{C}$, *par définition* $\mathcal{P}_\mathcal{C} \stackrel{def}{=} \{m \in \mathcal{M} \mid \mathcal{C}(\mathscr{L}(U(m)))\}$ *où la condition* $\mathcal{C}$ *porte sur le langage reconnu par* m

***si** des codages binaires m **et** m' **représentent des** MT **équivalentes**,*
ie. $\mathscr{L}(U(m)) = \mathscr{L}(U(m'))$
alors $m \in \mathcal{P}_\mathcal{C} \Longrightarrow m' \in \mathcal{P}_\mathcal{C}$

Preuve :

$$m \in \mathcal{P}_\mathcal{C} \Longrightarrow \mathcal{C}(\underbrace{\mathscr{L}(U(m))}_{=\mathscr{L}(U(m'))}) \Longrightarrow \mathcal{C}(\mathscr{L}(U(m'))) \Longrightarrow m' \in \mathcal{P}_\mathcal{C}$$

Autrement dit, $m \in \mathcal{P}_\mathcal{C}$ signifie $\mathcal{C}(\mathscr{L}(U(m))) = \mathbb{V}$ par définition de $\mathcal{P}_\mathcal{C}$. Prenons maintenant une MT m' équivalente à m, *ie.* $\mathscr{L}(U(m')) = \mathscr{L}(U(m))$. Forcément $\mathcal{C}(\mathscr{L}(U(m'))) = \mathcal{C}(\mathscr{L}(U(m))) = \mathbb{V}$ et donc $m' \in \mathcal{P}_\mathcal{C}$. □

Exercice 24 ***Indécidabilité de l'équivalence de machines de Turing*** (2.5 pt)

Le théorème de Rice permet de montrer qu'il n'existe pas de MT (*ie.* de programme) capable de dire si deux MTs (ou programmes) fournis en paramètre sont équivalents (c'est-à-dire qu'ils retournent le même résultat sur toutes les entrées possibles).

Q1. (2.5 pt) Étant donnée une MT M', montrez, à l'aide du théorème de Rice, que l'ensemble $\mathcal{P}_{\equiv M'}$ des machines de Turing équivalentes à M' est indécidable

$$\mathcal{P}_{\equiv M'} \stackrel{def}{=} \{m \mid U(m) \equiv M'\}$$

SOLUTION

Que doit-on montrer ? *Il suffit de montrer que $\mathcal{P}_{\equiv M'}$ est un ensemble de Rice pour conclure à son indécidabilité.*

Preuve : Rappelons que deux MT sont équivalentes si et seulement si elles reconnaissent le même langage, *ie.* $U(m) \equiv M' \iff \mathscr{L}(U(m)) = \mathscr{L}(M')$ Mais alors,

$$\begin{aligned}\mathcal{P}_{\equiv M'} &= \{m \mid U(m) \equiv M'\} \\ &= \{m \mid \mathscr{L}(U(m)) = \mathscr{L}(M')\} \\ &= \{m \mid \mathcal{C}(\mathscr{L}(U(m)))\} \quad \textit{avec} \quad \mathcal{C}(L) \stackrel{\textit{def}}{=} L = \mathscr{L}(M')\end{aligned}$$

Conclusion : On a réussi à exprimer $\mathcal{P}_{\equiv M'}$ sous la forme d'un ensemble de Rice $\{m \mid \mathcal{C}(\mathscr{L}(U(m)))\}$ donc $\mathcal{P}_{\equiv M'}$ est indécidable. □

7.1.6 Preuve du théorème de Rice

Soit $\mathcal{P}_\mathcal{C}$ un ensemble non-trivial de Rice. Montrons que $\mathcal{P}_\mathcal{C}$ est indécidable par réduction à $\mathcal{P}_\mathcal{C}$ d'un langage connu pour être indécidable.

Remarques préliminaires

1. Soit $\mathcal{P}_\mathcal{C}$ un ensemble non-trivial de Rice. $\mathcal{P}_\mathcal{C}$ non-trivial signifie que $\mathcal{P}_\mathcal{C} \neq \mathcal{M}$ et $\mathcal{P}_\mathcal{C} \neq \emptyset$. On en déduit (1) $\exists m_1 \notin \mathcal{P}_\mathcal{C}$ et (2) $\exists m_2 \in \mathcal{P}_\mathcal{C}$. En effet, si (1) est faux alors $\mathcal{P}_\mathcal{C} = \mathcal{M}$, ce qui contredit $\mathcal{P}_\mathcal{C}$ *non-trivial* et si (2) est faux alors $\mathcal{P}_\mathcal{C} = \emptyset$, ce qui contredit aussi $\mathcal{P}_\mathcal{C}$ *non-trivial* .
2. La MT $M_\emptyset \stackrel{\textit{def}}{=} \rightarrow \otimes$ reconnaît le langage vide
3. La preuve comporte une subtilité qui nous oblige à distinguer deux cas :

   (cas 1) $M_\emptyset \notin \mathcal{P}_\mathcal{C}$: on montre $\mathcal{P}_\mathcal{C}$ indécidable en réduisant la question $\stackrel{?}{\in} L_{EF}$ à $\stackrel{?}{\in} \mathcal{P}_\mathcal{C}$

   (cas 2) $M_\emptyset \in \mathcal{P}_\mathcal{C}$: on se ramène au (cas 1) en considérant l'indécidabilité de $\overline{\mathcal{P}_\mathcal{C}}$ qui alors ne contient pas $M_\emptyset$ puisque $m_\emptyset \in \mathcal{P}_\mathcal{C}$.

Première partie de la preuve (cas 1)

Preuve du théorème de Rice (cas 1) en supposant $M_\emptyset \notin \mathcal{P}_\mathcal{C}$: On prouve l'indécidabilité de $\mathcal{P}_\mathcal{C}$ par **réduction de L_{EF} à $\mathcal{P}_\mathcal{C}$**. Considérons le diagramme de réduction

$$\begin{array}{ccc} (m,\omega) \in \mathcal{M} \times \{0,1\}^* & \xrightarrow[traduction]{M_R} & R(m,\omega) \in \mathcal{M} \\ \underbrace{(m,\omega) \stackrel{?}{\in} L_{EF}}_{\textit{indécidable}} & \underset{(\ddagger)}{\iff} & \underbrace{R(m,\omega) \stackrel{?}{\in} \mathcal{P}_\mathcal{C}}_{\textit{indécidable}} \end{array}$$

Que doit-on faire ?

(a) On doit donner une MT qui traduit tout couple (m,ω) de $\mathcal{M} \times \{0,1\}^*$ en une MT, notée $R(m,\omega)$

(b) On doit montrer l'équivalence $(\ddagger)$.

Allons-y !

(a) **Définition de la traduction :** Puisque $\mathcal{P}_\mathcal{C}$ est non-trivial, $\mathcal{P}_\mathcal{C} \neq \emptyset$ donc il existe (au moins) une MT m_1 qui appartient à $\mathcal{P}_\mathcal{C}$ – cf. remarque préliminaire 1. On utilise m_1 pour définir $R(m,\omega)$:

$$\underbrace{\boxed{R(m,\omega)}}_{\text{MT}}(\omega_1) \stackrel{\textit{def}}{=} U(m)(\omega) \ \textbf{;} \ U(m_1)(\omega_1)$$

$R(m,\omega)$ est une MT à 4 bandes $B_1 = \omega_1$, $B_2 = m_1$, $B_3 = \omega$, $B_4 = m$. Par défaut le mot d'entrée ω_1 est inscrit sur la bande B_1. La MT $R(m,\omega)$ fait appel à la machine universelle U pour exécuter m sur ω ; puis (*si cette exécution termine*) elle exécute m_1 sur le mot d'entrée ω_1, toujours grâce à U.

(b) **Montrons l'équivalence** $(\ddagger)$

$(\Rightarrow)$ $(m,\omega) \in L_{EF}$ signifie que l'exécution $U(m)(\omega)$ termine mais alors l'exécution de $R(m,w)$ sur le mot d'entrée ω_1 se comporte comme celle de $U(m_1)(\omega_1)$; sauf qu'elle est précédée par un calcul $U(m)(\omega)$ qui termine et dont on ignore le résultat. On en déduit que $\mathscr{L}(R(m,\omega)) = \mathscr{L}(U(m_1))$. Autrement dit le langage de $R(m,\omega)$ est le même que celui de m_1 et puisque $m_1 \in \mathcal{P}_\mathcal{C}$ la proposition 13 permet de conclure que $R(m,\omega) \in \mathcal{P}_\mathcal{C}$ □

($\Leftarrow$) Montrons la réciproque : si $R(m,\omega) \in \mathcal{P}_\mathcal{C}$ alors $\mathscr{L}(R(m,\omega) \neq \emptyset$. En effet :

Preuve par contradiction: SUPPOSONS $\mathscr{L}(R(m,\omega) = \emptyset$ alors $\mathscr{L}(R(m,\omega) = \mathscr{L}(M_\emptyset)$ et comme $R(m,\omega) \in \mathcal{P}_\mathcal{C}$, d'après la proposition 13, $M_\emptyset$ appartient aussi à $\mathcal{P}_\mathcal{C}$: CONTRADICTION puisqu'on a justement supposé que $M_\emptyset \notin \mathcal{P}_\mathcal{C}$ $(cas\ 1)$. □

Maintenant, puisque $\mathscr{L}(R(m,\omega) \neq \emptyset$ il existe au moins un mot $\omega_1 \in \mathscr{L}(R(m,\omega))$, *ie.* que la MT $R(m,\omega)$ accepte le mot ω_1. Autrement dit l'exécution de $R(m,\omega)$ sur ω_1 *termine* sur un état accepteur. Et comme l'exécution de $R(m,\omega)$ commence par celle de $U(m)(\omega)$, *l'exécution de* $U(m)(\omega)$ *a dû au préalable terminer*, ce qui permet de conclure que $(m,\omega) \in L_{EF}$.

□

Seconde partie de la preuve (cas 2) La preuve exploite le résultat de l'exercice 22 page 31 qu'on rappelle ici.

$\overline{L}$ indécidable $\Longleftrightarrow$ L indécidable

Preuve du théorème de Rice, suite et fin, en supposant $M_\emptyset \in \mathcal{P}_\mathcal{C}$ (cas 2): D'après l'exercice 22 il suffit de montrer que $\overline{\mathcal{P}_\mathcal{C}}$ est indécidable pour conclure que $\mathcal{P}_\mathcal{C}$ est indécidable.

1. D'après la remarque préliminaire 1. On sait qu'il existe une MT $m_2 \notin \mathcal{P}_\mathcal{C}$ et donc $\boxed{m_2 \in \overline{\mathcal{P}_\mathcal{C}}}$
2. De même, puisque la MT $M_\emptyset$ appartient à $\mathcal{P}_\mathcal{C}$ $(cas\ 2)$, elle n'appartient pas à son complémentaire. Donc $\boxed{M_\emptyset \notin \overline{\mathcal{P}_\mathcal{C}}}$
3. On peut alors montrer l'indécidabilité de $\overline{\mathcal{P}_\mathcal{C}}$ par réduction de L_{EF} à $\overline{\mathcal{P}_\mathcal{C}}$ en recopiant la preuve du (cas 1) dans laquelle il suffit de remplacer $\mathcal{P}_\mathcal{C}$ par $\overline{\mathcal{P}_\mathcal{C}}$ et $m_1 \in \mathcal{P}_\mathcal{C}$ par $m_2 \in \overline{\mathcal{P}_\mathcal{C}}$.

□