

Chapitre I

Machine de Turing

Introduction

Nous allons maintenant introduire des machines plus puissantes, et voir un premier ensemble de propriétés qui en découlent :

- la machine de Turing.
- les variations sur les machines de Turing.
- l'hypothèse de Church-Turing.
- la complétude de Turing.

1 Machine de Turing

Rappel : un ADF ou un AP ne peut lire l'entrée qu'une seule fois.

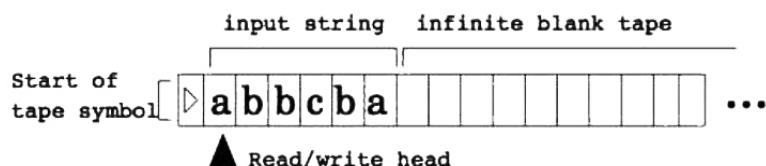
une machine de Turing (MT) est constituée :

- d'une bande (de taille infinie) : la chaîne d'entrée est placée au début de la bande
- d'un pointeur pouvant avancer/reculer en lecture/écriture : le pointeur est placé sur le premier symbole
- d'une machine à état déterministe qui contrôle l'évolution de la machine.

L'acceptation/rejet est immédiat(e).

Elle peut donc :

- avancer et reculer sur la chaîne d'entrée.
- écrire et lire des informations sur une bande (= mémoire)
en particulier, réécrire sur la chaîne d'entrée.



Entrée :

soit une chaîne d'entrée w de longueur n .

$\Rightarrow w = w_1 w_2 \cdots w_n$ où chaque $w_i \in \Sigma$ (i.e. $w \in \Sigma^*$).

Initialisation :

- **de la bande** : la chaîne d'entrée w est placée au début dans les n premières cases de la bande, un symbole w_i par case.

- le reste de la bande contient des blancs \square .
- \Rightarrow premier blanc = fin de la chaîne d'entrée
- **du pointeur** : le pointeur se trouve au début de la bande
donc, sur le premier symbole de la chaîne d'entrée
- **de l'état courant** : à l'état initial (q_0) de la machine à état.

Exécution :

- en fonction du caractère sous le pointeur de lecture, remplace ce caractère (éventuellement), puis déplace le curseur d'une case (à gauche ou à droite) sur la bande.
en début de la bande, une commande L ne déplace pas le pointeur.
- l'exécution continue jusqu'à ce que l'état acceptant (q_a) ou l'état rejettant (q_r) soit atteint
Sinon, la machine ne s'arrête jamais.

1.1 Définition

NOTATIONS:

- Q ensemble des états.
- Γ alphabet de la bande = alphabet des symboles + symboles supplémentaires (précision plus tard).
- $\{L, R\}$ mouvement du pointeur sur la bande (L = gauche, R = droite).

Fonction de transition $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

$\delta(q, a) = (r, b, L)$ signifie :

- **état courant** :
- sur le graphe** : l'état est q .
- sur la bande** : la tête est sur le symbole a .
- **transition** :
- sur le graphe** : l'état devient r .
- sur la bande** : la machine écrit b (remplace le a) et la tête va à gauche (L).

δ est une fonction **déterministe** : à savoir pour tout état, pour tout symbole sous le curseur, il existe une et une seule transition possible qui écrit sur la bande une valeur unique et se déplace dans la direction déterminée associée.

D'où la définition formelle d'une machine de Turing :

DÉFINITION 1 :

Une machine de Turing est un 7-uple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ où :

- Q est l'ensemble fini des états.
- Σ est l'alphabet d'entrée
ne contient pas le symbole blanc \square .
- Γ est l'alphabet de la bande
contient au moins \square et Σ .
 \square est la valeur par défaut d'un case de la bande.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ est la fonction de transition.
- $q_0 \in Q$ est l'état de départ.
- $q_a \in Q$ est l'état acceptant.
- $q_r \in Q$ est l'état rejettant.
- **Comment reconnaître que l'on est à la fin de la chaîne ?**
Lorsque l'on se déplace à droite, on lit le blanc \square .
- **Comment reconnaître que l'on est au début de la chaîne ?**
Plusieurs possibilités :
 - placer un blanc (ou tout autre caractère spécial) au début de la chaîne.

- soit après la lecture du premier caractère.
- soit en décalant la chaîne d'un caractère vers la droite.
- utiliser le fait que la tête ne bouge pas
 - se souvenir du symbole courant.
 - remplacer le symbole courant avec un symbole spécial.
 - se déplacer à gauche.
 - si le symbole spécial est toujours là, la tête est au début.
 - sinon, replacer la symbole initial et se déplacer à gauche.

— **Comment marquer un symbole ?**

Pour chaque symbole à marquer x , ajouter un symbole \bar{x} à l'alphabet de la bande Γ .

1.2 Exemples

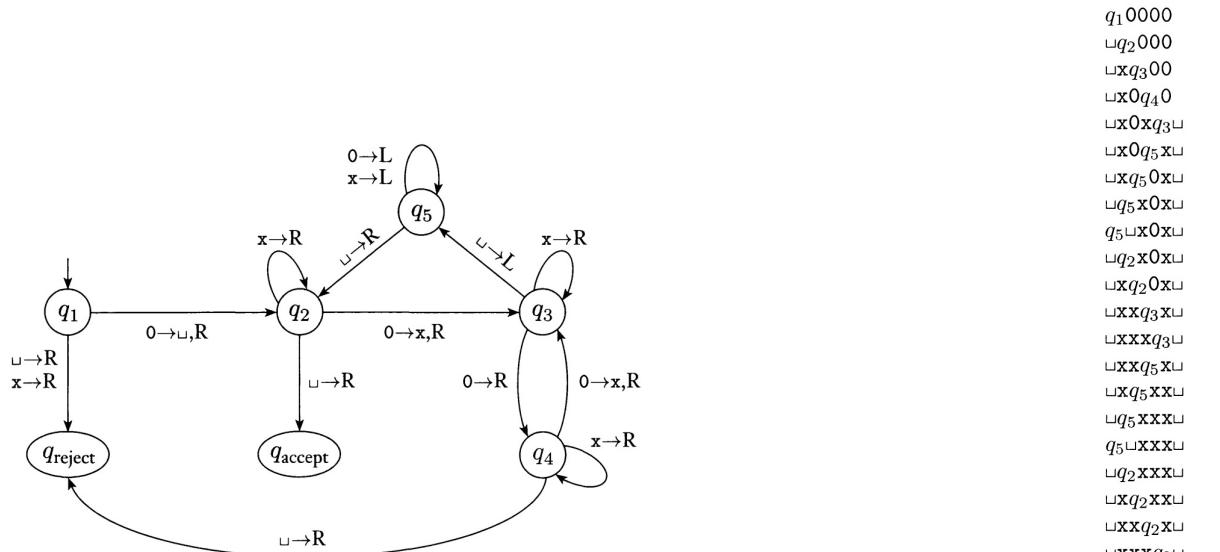
MT M qui décide $L = \{0^{2^n} \mid n \geq 0\}$.

Idée de fonctionnement d'une MT qui accepte le mot w que s'il appartient à L :

1. parcourir la bande de gauche à droite, en barrant un 0 sur deux.
2. si à l'étape 1, la bande contient un seul 0, accepter.
3. si à l'étape 1, la bande contient plus d'un 0 et que le nombre de 0 est impair, rejeter.
4. replacer le pointeur au début de la bande.
5. recommencer à l'étape 1.

Description formelle de la MT $M = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$ associée

$$\begin{aligned} Q &= \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\} \\ \Sigma &= \{0\} \\ \Gamma &= \{0, x, \sqcup\} \end{aligned}$$



Variation sur la méthode indiquée : on barre le premier 0 (= le dernier 0 qui reste), et on le remplace par un blanc afin de marquer le début de la bande.

Note : $0 \rightarrow R$: lit 0 et va à droite (pas d'écriture).

$0 \rightarrow x, R$: lit 0, écrit x et va à droite.

Construction de la MT qui décide $L = \{a^i b^j c^k \mid i, j, k \geq 1\}$

1. lire la chaîne de gauche à droite et vérifier qu'elle est bien de la forme $a^* b^* c^*$.

2. retourner au début de la bande.
3. rayer un a , et scanner à droite jusqu'à ce l'on rencontre un b .
4. faire la navette entre les b et les c , en en rayant un de chaque jusqu'à ce qu'il n'y ait plus de b (rejeter si je ne peux pas rayer de c).
5. rétablir tous les b barrés.
6. s'il existe encore des a non barrés, alors retourner à l'étape 3.
7. sinon si tous les c sont barrés, alors accepter la chaîne.
8. sinon refuser la chaîne.

1.3 Configuration d'une machine de Turing

Configuration \triangleq notation qui contient :

- l'**état**,
- la **position du pointeur**,
- le **contenu de la bande**

de la MT à un instant donné.

Exemple :

une configuration $1011q_30111$ signifie :

- **état** : q_3
- **bande** :
 - **partie gauche** : 1011
 - **partie droite** : 0111 ($__$ non représentés)
 - **pointeur** : sur le premier symbole de la partie droite (i.e. 0).

Configurations particulières :

- | | |
|--------------------------|----------------------------|
| configuration de départ | q_0w |
| configuration acceptante | $w_0q_aw_1$ |
| configuration refusante | $w_0q_rw_1$ |
| configuration d'arrêt | = acceptante ou refusante. |

Exemple de transformation d'une configuration

à partir d'une configuration initiale $uaq bv$:

- la fonction de transition $\delta(q_i, b) = (q_j, c, L)$
conduit à la configuration uq_jacv
- la fonction de transition $\delta(q_i, b) = (q_j, c, R)$
conduit à la configuration $uacq_jv$

Cas particuliers de transitions :

1. pointeur sur le 1^{er} symbole à gauche :

changement d'état, écriture mais le pointeur ne bouge pas.

- | | |
|--------------------------|--------------------------------|
| configuration initiale : | $q_i bv$ |
| transition : | $\delta(q_i, b) = (q_j, c, L)$ |
| configuration finale : | $q_j cv$ |

2. pointeur sur le dernier symbole à droite :

wq_i et $wq_{i__}$ représentent la même configuration.

- | | |
|--------------------------|---------------------------------|
| configuration initiale : | wq_i |
| transition : | $\delta(q_i, _) = (q_j, c, R)$ |
| configuration finale : | wcq_j |

1.4 Modèle de calcul d'une machine de Turing

DÉFINITION 2 : chaîne acceptée par une MT M

M accepte un chaîne w en entrée s'il existe une suite de configurations C_1, C_2, \dots, C_k telles que :

- C_1 est la configuration de départ de M sur w .
- pour tout i , la configuration C_i conduit à la configuration C_{i+1} .
- C_k est une configuration acceptante.

DÉFINITION 3 : langage accepté par une machine de Turing

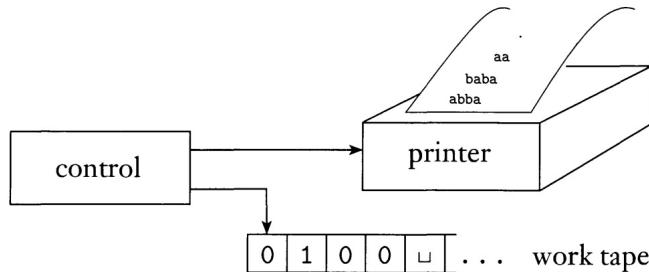
Un langage L reconnu par une machine de Turing M est constitué de l'ensemble des mots $w \in \Sigma^*$ tels que $M(w)$ accepte.

Notation : $L = \mathcal{L}(M)$

1.5 Enumérateur

Enumérateur : type particulier de MT avec une imprimante attachée servant à écrire l'ensemble des chaînes contenues dans le langage reconnu par cette MT.

- énumération dans n'importe quel ordre, éventuellement avec répétition.
- ne possède pas d'entrée (mais possède une bande pour travailler).



THÉORÈME 1 : lien énumérateur-MT

Un langage est accepté par une MT si et seulement si il existe un énumérateur qui l'énumère.

DÉMONSTRATION:

$$\Leftarrow : \exists \text{ énumérateur } E \text{ qui énumère } L \Rightarrow \exists \text{ MT } M \text{ qui reconnaît } L$$

Construire M comme : pour la chaîne d'entrée w de M

- Exécuter E : pour chaque sortie v de E , comparer v avec w .
- Si $w = v$, alors accepter sinon continuer l'exécution de E .

Clairement, M accepte les sorties de E , donc M reconnaît L .

$$\Rightarrow : \exists \text{ MT } M \text{ qui reconnaît } L \Rightarrow \exists \text{ énumérateur } E \text{ qui énumère } L$$

Soit s_1, s_2, s_3, \dots l'ensemble des chaînes possibles de Σ^* .

Construire E de la façon suivante :

Répéter pour $i = 1, 2, 3, \dots$

Pour chaque $s_j = s_1, s_2, \dots, s_i$

Exécuter M avec i transitions sur s_j .

Si M accepte s_j , alors afficher s_j .

La limitation de l'exécution à i transitions permet d'éviter d'être bloqué au cas où un s_j ferait boucler M .

Inconvénient : la même sortie peut apparaître de très nombreuses fois.

□

EXEMPLE 3: énumérateur

Code de l'énumérateur qui écrit tout les couples de la forme (n, p) où $n \in \mathbb{N}$, $p \in \mathbb{N}$ et n est un multiple de p .

```

pour  $n = 1, 2, \dots$ 
  pour  $1 \leq p \leq n$ 
    si  $n \% p = 0$  alors afficher  $(n, p)$ 
```

REMARQUE 1 :

Evidemment, si un énumérateur E reconnaît un langage L , et que $\#L$ n'est pas fini, l'énumérateur ne s'arrête jamais.

1.6 Langage récursivement énumérable

DÉFINITION 4 : langage récursivement énumérable

Un langage L est dit (récursivement) énumérable s'il existe une machine de Turing qui l'accepte.

Notes : Initialement, la définition d'un langage énumérable était "qui peut être énuméré par un énumérateur".

DÉFINITION 5 : Classe des langages récursivement énumérables

La classe des langages récursivement énumérables est constituée de l'ensemble des langages reconnus par une machine de Turing.

Notation : On note \mathcal{RE} cette classe.

Pour un langage L récursivement énumérable, une MT M qui reconnaît L ne peut faire que trois choses sur un mot d'entrée w ,

- l'accepter ($w \in L$),
- le rejeter ($w \notin L$),
- ne pas s'arrêter (boucle infinie, $w \notin L$).

Si $w \notin L$, il est possible que M ne donne jamais de réponse.

C'est la définition la plus faible qui puisse être donnée au fait qu'un langage est reconnu par une MT.

1.7 Décidabilité

En même temps, ceci est problématique, car on s'attendrait à ce qu'une machine qui reconnaît un langage donne une réponse définitive pour toute entrée.

On introduit alors la notion de décidabilité :

DÉFINITION 6 : langage décidable

Un langage L est décidable s'il est récursivement énumérable (*i.e.* il existe une machine de Turing M qui accepte L) et que M s'arrête pour toutes les entrées.

On dit alors que la MT M décide L .

Autrement dit, si L est décidable, alors il existe une MT M telle que :

- si $w \in L$, alors M accepte w .
- si $w \notin L$ alors M rejette w .

i.e. M s'arrête dans tous les cas.

DÉFINITION 7 : Classe des langages décidables

La classe des langages décidables est constituée de l'ensemble des langages décidables.

Notation : On note \mathcal{R} cette classe.

Cette notion sera approfondie dans la leçon suivante où nous verrons que la décidabilité est une notion plus forte que la recursive-énumérabilité.

2 Autres modèles de machine de Turing

On se propose de déterminer l'impact d'une modification du modèle de la machine de Turing.

Modifications étudiées :

- Ajout d'une commande S (= stay) permettant à la MT de lire/écrire sans se déplacer,
- Restriction de l'alphabet Σ à $\{0, 1\}$ (= codage d'un alphabet fini en binaire),
- Utilisation d'une bande de travail infinie des deux cotées,
- Utilisation de plusieurs bandes de travail au lieu d'une seule,
- Utilisation du non-déterminisme

Cet impact peut être de plusieurs ordres :

- sur la performance,
- sur l'ensemble des langages reconnus par rapport à une MT classique.

2.1 Ajout de commande

On veut ajouter une commande S permettant à la machine de Turing de lire/écrire sans se déplacer.

La fonction de transition est alors définie dans :

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$

Analyse : il est possible de simuler une MT LRS sur une MT LR en remplaçant chaque transition

S par deux transitions (R puis L)

note : pas LR (problème au début de la bande).

Conséquences :

- une MT LRS modifiée n'est pas plus puissante puisqu'elle peut être simulée sur une MT LR.
⇒ L'ensemble des langages reconnus par les MT LRS et les MT LR est le même.
- la simulation ralentit la MT LR au plus d'un facteur 2 par rapport à la MT LRS.

Important : idée de simuler une MT *LRS* à partir d'une MT *LR*, et inversement (utile plus tard pour les équivalences).

2.2 Restriction d'alphabet

Une machine avec un alphabet plus étendu est-elle plus puissante qu'une machine avec un alphabet plus restreint ?

Considérons une MT M_Σ utilisant un alphabet Σ étendu. Est-il possible de simuler M_Σ sur une MT M_B en utilisant un alphabet binaire $B = \{0, 1\}$?

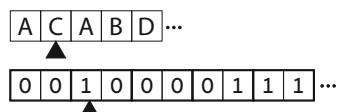
Evidemment, cette simulation ne peut s'effectuer qu'à travers l'encodage des symboles de Σ dans B .

Avec un codage à taille fixe, il faut $p = \lfloor \log_2 \#\Sigma \rfloor$ bits pour coder l'ensemble des symboles de Σ dans B^p .

La bande est découpée en blocs de b bits dont chacun représente un symbole de Σ .

EXEMPLE 4:

$\Sigma = \{A, B, C, D\}$ codé dans B^2 en $\{00, 01, 10, 11\}$.



Sur M_B , la simulation d'une transition $a \rightarrow b, L$ (resp. $a \rightarrow b, R$) de M_Σ consiste en :

- lecture à droite du bloc de p bits contenant le code de $a = p$ transitions.
- retour d'un caractère à gauche (= retour sur le dernier caractère du bloc),
- écriture inversée à gauche du bloc de p bits contenant le code de $b = p$ transitions.
- retour d'un caractère à droite (= retour sur le premier caractère du bloc),
- déplacement du pointeur de lecteur à gauche (resp. à droite) sur le bloc précédent = p transitions.

Soit un facteur multiplicatif de l'ordre de $3p$ transitions.

Conséquences : une MT M_Σ peut être simulée sur une MT M_B .

- Elle n'est pas plus puissante qu'une M_B ,
- La simulation M_B augmente d'un facteur $3 \cdot \lfloor \log_2 \#\Sigma \rfloor$ le nombre de transitions nécessaires sur une entrée par rapport à M_Σ .

2.3 Bande doublement infinie

Considérons maintenant le cas où la bande est infinie des deux côtés.

Cette MT M_b peut être transformée en une MT M classique en transformant la bande de la manière suivante :

$$\dots [C|A|A|B] [A|B|D|C] \dots \Rightarrow \begin{array}{c} \overbrace{A|B|D|C}^{\text{...}} \\ \overbrace{B|A|A|C}^{\text{...}} \end{array} \Rightarrow [AB|BA|DA|CC] \dots$$

- "plier" la bande en deux à partir de son milieu (= position à laquelle le pointeur de lecteur est initialisé).
- chaque symbole de la bande est codé par un couple de symboles.
- pour un déplacement sur la portion droite (resp. gauche), considérer le symbole de droite (gauche) (= une version de M_b gauche et une droite).
- pour savoir quand passer de l'un à l'autre, placer un marqueur # au centre de la bande. Sa détection s'effectue en 2 transitions :
 - si le symbole courant $\neq \#$ alors on continue sur le M_b courant.
 - sinon passer sur l'autre M_b (avec inversion déplacement).

Conséquences : une MT M_b peut être simulée sur une MT M .

- Elle n'est pas plus puissante qu'une M ,
- La simulation M_b augmente d'un facteur multiplicatif 2 le nombre de transitions nécessaires sur une entrée par rapport à M .

2.4 Machine de Turing multibandes

Une MT est modifiée en une MT_k à k bandes de la manière suivante :

- la mémoire est constituée de k bandes (infinies à droite),
- il y a k pointeurs de bande indépendants (un par bande),
- à l'initialisation, la chaîne d'entrée placée sur la première bande, les autres bandes sont blanches.
- la fonction de transition est définie par $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$
- $\delta(q_i, a_1, \dots, a_k) = (q_j, b_1, \dots, b_k, L, \dots, R)$ signifie :
 - si la machine est dans l'état q_i , et que les pointeurs $\{1, \dots, k\}$ lisent $\{a_1, \dots, a_k\}$ sur les k bandes,
 - alors la machine passe à l'état q_j , les pointeurs $\{1, \dots, k\}$ écrivent $\{b_1, \dots, b_k\}$ sur les k bandes puis bougent respectivement dans les directions $\{L, \dots, R\}$.

Typiquement, pour ce type de modèle de MT, un rôle particulier est affecté à chaque bande (exemple pour un transducteur : bande 1 = bande d'entrée en lecture seule, bande 2 = bande de travail, bande 3 = bande de sortie en écriture seule).

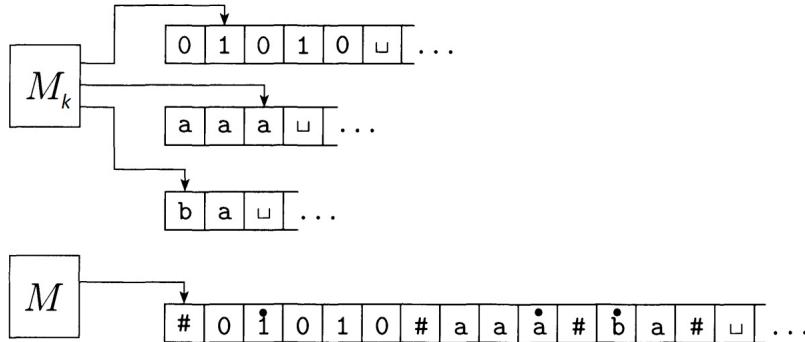
THÉORÈME 2 : équivalence MT simple bande - MT multi-bande

Toute MT M_k a k bandes a une MT M simple-bande équivalente.

DÉMONSTRATION: (constructive)

On stocke les k bandes de M_k sur la bande de M avec :

- un marqueur séparateur de bande : symbole #.
- un marqueur de position de pointeur : pour tout symbole $x \in \Gamma_k$, on ajoute une version pointée \dot{x} à Γ signifiant que le pointeur est placé sur ce symbole.



Fonctionnement de M : avec l'entrée $w = w_1 \cdots w_n$

1. **initialisation de la bande :** $\# \dot{w}_1 w_2 \cdots w_n \# \# \# \cdots \# \#$
2. **simulation d'une transition :**
 - **détermination de l'état** : scanner la bande du premier # au $(k+1)^{\text{ème}}$ qui marque la fin de la dernière bande pour déterminer les symboles pour chaque pointeur de chaque bande.
 - **transition** : faire une seconde passe pour effectuer la transition associé à chaque symbole pointé, en accord avec la fonction de transition de M_k .
3. **ajustement de la taille d'une bande :**
si l'un des pointeurs virtuels se déplace sur un # (arrivée en bout de bande virtuelle), alors M écrit un symbole $_$, et décale tous les symboles à droite d'une case vers la droite.

M reproduit ainsi très exactement le comportement des k bandes de M_k . \square

On montrera que la complexité supplémentaire engendrée par la MT à une bande est un facteur multiplicatif $5.k.T(n)^2$ où $T(n)$ est le nombre de transitions de la MT à k bandes pour une entrée de taille n .

2.5 Machine de Turing non déterministe

Une machine de Turing non déterministe (MTND) est une généralisation similaire à celle des ADFs en ANFs :

- pour chaque état, il existe une ou plusieurs transitions possibles associées au symbole sous le pointeur, voir aucune.
- pour chaque choix non déterministe lors de l'exécution, la machine se découpe en autant de copies que nécessaires, chaque copie s'occupant d'une branche d'exécution.
- si l'exécution d'une branche ne peut se continuer, alors elle s'arrête (=rejet).
- une branche accepte si son exécution permet d'atteindre l'état acceptant.

Pour une entrée w , une MTND :

- accepte w s'il existe (au moins) une branche qui l'accepte.
- rejette w si toutes les branches la rejettent ou bouclent à l'infini.

En terme de configuration,

- Une exécution d'une MT déterministe est un chemin dans l'espace des configurations de la MT.
- Une exécution d'une MTND est un arbre dans l'espace des configurations de la MTND.

La fonction de transition est de la forme :

$$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

Une MTND semble visiblement plus puissante qu'une MT.
une MT = une branche d'exécution d'une MTND.

Donc, tout langage récursivement énumérable peut être aussi reconnu par une MTND.

THÉORÈME 3 : équivalence MTND-MT

Toute MTND M' a une MT M équivalente.

DÉMONSTRATION:

idée : simuler une MTND M' avec une MT M = explorer toutes les branches de la MTND M' .

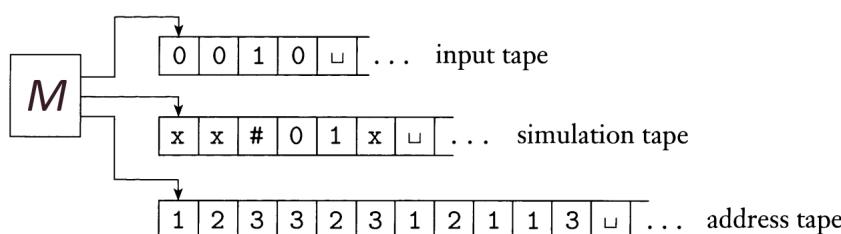
- M essaie toutes les branches possibles.
- si M accepte l'une de ces branches, alors M accepte.
- si toutes les branches sont rejetées, alors M rejette.
- si toutes les branches sont rejetées ou bouclent, alors M boucle.

L'exécution de M est un arbre :

- chaque branche de l'arbre est une branche de l'exécution non déterministe de M' .
- le noeud de l'arbre est la configuration initiale.
- chaque noeud de l'arbre est une configuration de M .
- le nombre d'enfants d'un noeud est au plus $n = \#Q \times \#\Gamma \times 2$
- note :** $2 = \#\{L, R\}$.
- parcours de l'arbre :
 - la recherche en profondeur d'abord ne fonctionne pas.
 - la branche courante peut entrer dans une boucle infinie.
 - la recherche en largeur d'abord est la solution.
 - on trouve ainsi la plus petite configuration qui accepte la chaîne.

On simule la MTND avec une MT à 3 bandes avec :

- **une bande d'entrée** : contient la chaîne d'entrée, et n'est jamais modifiée,
- **une bande de simulation** : contient la même chose que la bande de M' ,
- **une bande d'adresses** : contient de la position de M dans l'arbre d'exécution de M' .



Utilisation de la bande d'adresses :

- chaque noeud a au plus n enfants.
- à chaque noeud, on peut affecter une adresse qui est une chaîne dans l'alphabet $\Delta = \{1, \dots, n\}$ chaque symbole correspondant à une configuration.
- **utilisation d'une adresse :**
 - pour se rendre au noeud d'adresse 231
 - choisir le second fils de la racine.
 - puis choisir son troisième fils de ce second fils.
 - puis choisir son premier fils de ce troisième fils.

- ignorer les adresses qui n'ont pas de sens et qui correspondent à des configurations impossibles

Génération de tous les chemins possibles dans l'arbre :

exemple : pour 2 enfants par nœud ($\Delta = \{1, 2\}$), avec une recherche en largeur d'abord, revient à construire toutes les chaînes possibles dans l'ordre :

1, 2, 11, 12, 21, 22, 111, 112, 121, 122, 211, 212, 221, 222, ...

Tous les chemins partant de la racine sont ainsi simulés.

Simulation :

1. **Initialisation** : la bande d'entrée contient w , les deux autres sont vides.
2. copier la bande d'entrée sur la bande de simulation.
3. utiliser la bande de simulation pour simuler M' sur l'entrée w sur une portion limitée d'une des branches déterministes (l'entrée est simulée depuis le début).
 - A chaque choix,
 - consulter le symbole suivant sur la bande d'adresse.
 - si un état acceptant est atteint, alors accepter w .
 - passer à l'étape suivante si :
 - les symboles sur la bande d'adresse sont épuisés.
 - un choix non déterministe est invalide.
 - un état rejetant est atteint.
4. remplacer la chaîne sur la bande d'adresse par l'entrée suivante pour chaque choix possible.
5. sauter à l'étape 2 et simuler cette branche de l'exécution de M' .

On simule ainsi toutes les branches de la MTND. □

THÉORÈME 4 : simulation d'une MTND sur une MT (non démontré)

Une MT déterministe M en temps $t(n)$ peut être simulée sur une MTND N en temps proportionnel à $c(M)^{t(n)}$ où $c(M)$ est une constante dépendante de M .

La constante $c(M)$ dépend du nombre d'états, du nombre de bandes et de la taille de l'alphabet de M .

Conséquences : une MTND N peut être simulée sur une MT M ,

- N n'est pas plus puissante qu'une M ,
- Les MTNDs reconnaissent (exactement) les langages récursivement énumérables.
- en revanche, N peut reconnaître le langage potentiellement exponentiellement plus vite que M .

2.6 Conséquence

Le modèle de la MT est extrêmement stable : les variations sur ce modèle sont incapables de reconnaître autre chose que les langages récursivement énumérables.

En terme de performance,

- les variations déterministes peuvent être simulées sur une MT classique et n'entraîne qu'au plus un gain quadratique (proportionnel à $k \cdot t^2$ pour une MT à k bandes),
- les variations non déterministes peuvent également être simulées mais le gain dans ce cas peut être beaucoup plus conséquent (exponentiel) dans certains cas particuliers que nous préciserons.

En fait :

Tous les modèles de machines de calcul sont équivalents à une machine de Turing.

λ -calcul, machine à compteurs, grammaire sans restriction, RAMs, ...

3 Hypothèse de Church-Turing

3.1 Encodage

Pour l'instant, nous avons utilisé des MTs afin de reconnaître des langages.

Remarquons qu'il est possible de passer à une MT n'importe quel type de structures : graphes, matrices, polynômes, MT, ...

Tout type de structure peut être encodé dans une chaîne de caractères :

- codage = à voir comme un langage particulier (au choix).
- passé à la machine de Turing = stocké sur la bande d'entrée.

Par la suite,

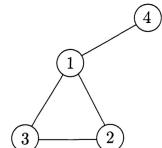
- on ne s'intéressera plus à la façon de coder/décoder
- seulement à la question de savoir si cela peut être fait.

Notations :

- si O est un objet, alors $\langle O \rangle$ est l'encodage de O dans la MT.
- si O_1, O_2, \dots, O_k sont des objets, alors $\langle O_1, O_2, \dots, O_k \rangle$ est l'encodage de ces objets dans une seule chaîne.

EXEMPLE 5: encodage d'un graphe

Graphe G :



Encodage :

$$\begin{aligned} \langle G \rangle &= (1, 2, 3, 4)((1, 2), (2, 3), (3, 1), (1, 4)) \\ &= (\text{liste des nœuds})(\text{liste de paire de nœuds}) \end{aligned}$$

Aisément exprimable comme une GLC.

$\langle G \rangle$ est un codage valide de G si on peut vérifier que :

- la syntaxe est correcte (GLC),
- l'absence de répétition d'un nœud dans la liste des nœuds
- la présence des nœuds de la liste des arêtes dans la liste des nœuds

Chacune de ces propriétés peut être vérifiée au cours de l'exécution d'une MT.

3.2 Algorithmes

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

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

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

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

Ce qui amène aux questions :

- qu'est-ce qu'un algorithme ?

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

Remarque : nous verrons plus tard qu'il aussi est possible de construire une MT qui produit une sortie (voir la partie sur les fonctions calculables).

Qu'est-ce qu'un algorithme ?

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

Historiquement :

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

La notion était informelle, mais suffisante jusque là.

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

3.3 Thèse de Church-Turing

D'où la thèse de Church-Turing :

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

Donc,

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

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

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

Attention, ceci n'est qu'une hypothèse.

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

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

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

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

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

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

- Il n'est pas clair que les implémentations physiques des ordinateurs quantiques actuels utilisent bien les propriétés quantiques (la superposition d'état) qui rendent les ordinateurs quantiques intéressants.

4 Complétude de Turing

De ce qui précède, nous avons utilisés les machines de Turing sous forme de «circuit spécialisé».

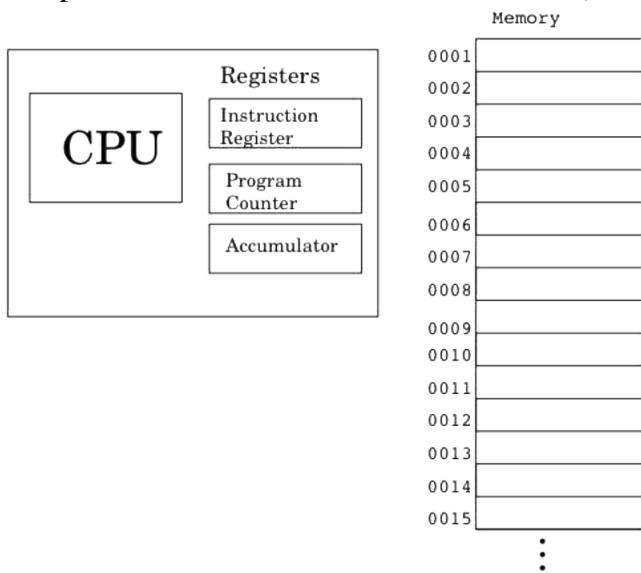
i.e. si nous avions construit une implémentation physique d'une MT M qui reconnaît un langage particulier. nous aurions été obligé de construire une autre machine différente pour reconnaître un autre langage.

On se demande alors s'il ne serait pas possible de construire une modèle de machine qui prenne en paramètre un programme et serait ainsi capable de simuler l'algorithme décrit dans ce programme ?

Etudions une première implémentation de ce principe sur un modèle d'ordinateur élémentaire sous la forme d'une machine à accès aléatoire (ou RAM).

4.1 Random Access Machine

Composants d'une machine à accès aléatoire (ou RAM) :



Une RAM est définie à partir :

- d'une mémoire (MEM).
 - MEM[i] représentant le contenu de la mémoire à l'adresse i.
 - de registres :
 - IP : instruction pointer (ligne du code en cours d'exécution)
connu aussi sous le nom de Program Counter
 - IR : instruction register (code de l'instruction à exécuter)
 - A : accumulateur (memoire locale)
 - d'un CPU qui fonctionne comme suit :
 1. $IR \leftarrow MEM[IP]$ (= charger l'instruction IR)
 2. incrémenter IP
 3. exécuter l'instruction dans IR
- voir ci-après pour le type d'instructions exécutable

Jeux d'instructions du CPU :

Code	Instruction	Signification
000000	HALT	Arrêt
01xxxx	LOAD xxxx	$A \leftarrow \text{MEM}[xxxx]$
02xxxx	LOADI xxxx	$A \leftarrow xxxx$
03xxxx	STORE xxxx	$\text{MEM}[xxxx] \leftarrow A$
04xxxx	ADD xxxx	$A \leftarrow A + \text{MEM}[xxxx]$
05xxxx	ADDI xxxx	$A \leftarrow A + xxxx$
06xxxx	SUB xxxx	$A \leftarrow A - \text{MEM}[xxxx]$
07xxxx	SUBI xxxx	$A \leftarrow A - xxxx$
08xxxx	JUMP xxxx	$IP \leftarrow xxxx$
09xxxx	JZERO xxxx	$IP \leftarrow xxxx \text{ if } A = 0$
10xxxx	JGT xxxx	$IP \leftarrow xxxx \text{ if } A > 0$

Exemple de programme : multiplication

multiplie 2 nombres (stockés aux adresses 1000 et 1001)

stocke le résultat à l'adresse 1002.

Memory	Code	Assembleur	Algorithm
0001	011000	LOAD 1000	START $A \leftarrow \text{MEM}[1000]$
0002	031003	STORE 1003	$\text{MEM}[1003] \leftarrow A$
0003	020000	LOADI 0	$A \leftarrow 0$
0004	031002	STORE 1002	LOOP $\text{MEM}[1002] \leftarrow A$
0005	021003	LOAD 1003	$A \leftarrow \text{MEM}[1003]$
0006	090012	JZERO 0012	IF $A == 0$ GOTO END
0007	070001	SUBI 1	$A \leftarrow A - 1$
0008	031003	STORE 1003	$\text{MEM}[1003] \leftarrow A$
0009	011002	LOAD 1002	$A \leftarrow \text{MEM}[1002]$
0010	041001	ADD 1001	$A \leftarrow A + \text{MEM}[1001]$
0011	080004	JUMP 0004	JUMP LOOP
0012	000000	HALT	END HALT

THÉORÈME 5 : simulation d'une RAM

Toute RAM peut être simulée par une MT multi-bande.

DÉMONSTRATION: Très long ! Juste une idée de la preuve.

Organisation des bandes :

- utiliser une bande par registre (IR, IP et A).
- utiliser une bande pour la mémoire
- organisation de la mémoire :
 - avec des entrées : <adresse> <contenu>
 - utilisée pour stocker le code et les données

Initialisation : (de la MT)

Placer les données et le code dans la mémoire.

Initialiser le contenu de l'IP à **0001**

Exécution : (de la MT comme RAM)

1. parcourir la mémoire à la recherche de l'adresse correspondant à l'IP.
2. copier le contenu de cette adresse dans l'IR.
3. incrémenter l'IP.
4. en fonction de la valeur de l'IR
 - copier une valeur dans l'IP (jump)
 - copier une valeur / l'accumulateur dans la mémoire (store)
 - copier une valeur / le contenu mémoire dans l'accumulateur (load)
 - faire une addition/soustraction
5. recommencer au 1.

Note : montrer aussi que chaque instruction peut se simuler sur la MT.

□

Exécution ! (multiplication)

- Etat initial



- Recherche de l'IP dans la mémoire (1^{ère} instruction)



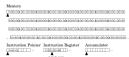
└

┘

- Chargement du contenu à l'adresse de IP dans l'IR



- Instruction LOAD : pointeur au début de la mémoire



- Instruction LOAD : recherche de l'adresse



- Instruction LOAD : chargement de la valeur dans l'accumulateur



- Recherche de l'IP dans la mémoire (2^{ème} instruction)



- Chargement du contenu à l'adresse de IP dans l'IR



4.2 Machine de Turing Universelle

Il est évident que la description d'une MT M peut elle aussi être encodée (*i.e.* $\langle M \rangle = \langle (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r) \rangle$).

On définit alors :

MT universelle $\stackrel{\Delta}{=}$ MT qui peut simuler une MT M arbitraire sur une entrée arbitraire w .

Comment réaliser cela ?

La MT universelle U fonctionne de la façon suivante :

- Mettre $\langle M, w \rangle$ en entrée de la MT.
- Vérifier que $\langle M, w \rangle$ est un encodage correct d'une MT, suivi par une chaîne de Σ^* .



- Simuler M sur w .
voir ci-après le détail.
- A l'entrée w ,
 - si M entre dans un état acceptant, alors U accepte.
 - si M entre dans un état rejetant, alors U rejette.
 - si M boucle, alors U boucle aussi.

Comment simuler M sur U ?

— Organisation :

Pour une MT M à une bande et alphabet de travail Γ , la MT universelle U a :

- 5 bandes :

1. **bande d'entrée** : contient $\langle M, w \rangle$
2. **bande du programme** : contient $\langle M \rangle$
3. **bande de simulation** : contient w
4. **bande d'état** : contient l'état de la MT M
5. **bande de travail** : stockage intermédiaire.

- un alphabet de travail Γ' étendu
marquage de symboles : pointé (position de lecture), barré, ...

— Initialisations

- copies : $\langle M \rangle$ sur la bande de programme, w sur la bande de simulation.
- placer la tête de la bande de simulation au début de w (pointer son premier symbole).
- placer l'état de départ sur la bande d'état.

— Execution de U

1. placer une copie de l'état q_i (depuis la bande d'état) et une copie du caractère courant a (depuis la bande de simulation) sur la bande de travail.
2. comparer (q_i, a) aux entrées de table de transition de la bande du programme.
3. lorsque la correspondance est trouvée (ex : $\delta(q_i, a) = (q_j, b, L)$)
 - mettre à jour la bande d'état avec q_j .
 - écrire la lettre b et déplacer le pointeur sur le caractère dans la direction L sur la bande de simulation.
4. test de l'état q écrit sur la bande d'état.
 - si $q = q_{\text{accept}}$ alors U accepte $\langle M, w \rangle$
 - si $q = q_{\text{reject}}$ alors U rejette $\langle M, w \rangle$
 - sinon on recommence au 1.

REMARQUES: la MT universelle U

- a un alphabet de travail beaucoup plus grand pour faciliter les comparaisons, les copies, l'effacement
- a un nombre d'états **fini** (environ une centaine)
 - le nombre d'états est indépendant de la machine à simuler.
 - U peut donc simuler une MT M avec beaucoup plus d'états.

THÉORÈME 6 : MT universelle efficace

Soit une MT $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ où $T(w)$ est le nombre de transitions avant son arrêt sur l'entrée w .

Il existe une MT universelle capable de simuler une machine M en $C \cdot T \log T$ transitions où C dépend de $|w|$, $\#\Sigma$, $\#Q$ et du nombre de bandes.

Démonstration :

voir "Two-tape simulation of multitape Turing machines", Hennie et Stearns, 1966.

La méthode présentée ci-avant est en T^2 . □

Intéressons-nous aux descriptions de MT possibles :

- il est clair que l'ensemble des descriptions MTs possibles est Σ^* .
- on peut faire en sorte que la MT universelle rejette toutes les entrées si la description de la machine $\langle M \rangle$ n'est pas valide (de façon à ce que toute chaîne de Σ^* soit associée à une MT).
- on peut faire en sorte que la MT universelle ignore tous les symboles qui suivent une description correcte d'une MT correcte (*i.e.* $\langle M \rangle \equiv \langle M \rangle \Sigma^*$).
de la même façon que l'ajout de commentaires à un programme ne change pas son sens.

En conséquence,

- il y a un nombre d'algorithmes infini (autant que $\#\Sigma^*$),
- il y a un nombre infini d'algorithmes qui acceptent le même langage.

Attendez !

- Une MT universelle U est une MT qui accepte [*une MT M qui accepte tout w ∈ L*].
- On vient donc de démontrer que l'ensemble des MTs est énumérable.
- Plus formellement, soit $A_{\text{MT}} = \{ \langle M, w \rangle \mid M \text{ est une MT qui accepte } w \}$.
Alors A_{MT} est récursivement énumérable.
- Mais A_{MT} est-il décidable ?
La MT universelle ne permet pas de le déterminer : si M boucle, U aussi.
A suivre ...

Historique : C'est la MT universelle qui a inspiré les premiers ordinateurs programmables des années 40 et 50.

4.3 Complétude de Turing

Sur une MT universelle, le programme à exécuter est donné sous forme de la description d'une MT (= langage de programmation).

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

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

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

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

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

DÉFINITION 8 : Complétude de Turing

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

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

5 Résumé

Nous avons vu dans ce cours que :

- la machine de Turing est modèle très puissant pour représenter les algorithmes,
- la classe des langages récursivement énumérables est la classe des langages pouvant être reconnus par une MT ; les mots hors langages peuvent faire boucler la machine,

- la classe des langages décidables est la classe des langages reconnus par une MT et qui s'arrête sur toutes les entrées,
- d'après la thèse de Church-Turing, un algorithme s'exécutant sur une machine de Turing est équivalent à un modèle raisonnable de calcul informatique,
- tout modèle raisonnable de calcul peut être simulé sur une machine de Turing,
- cette simulation peut être effectuée avec un facteur multiplicatif de pénalité au plus quadratique, exception faite des MTs quantiques.
- un modèle de machine est Turing-complet s'il permet de simuler tout algorithme (=simuler une machine de Turing),
- le modèle de la machine de Turing a servi de base pour concevoir le principe des ordinateurs modernes.

