Chapitre IV

Retour sur les machines de Turing

Dans ce chapitre, nous revenons sur le modèle de la machine de Turing. Il nécessite la compréhension :

- des automates finis, déterministe et non déterministe,
- des automates à pile

pour lesquels nous ferons quelques rappels.

Ces concepts vu en théorie du langage étant maintenant considérés comme acquis, nous allons donc étudier :

- le modèle abstrait de la machine de Turing,
- différentes variations,
- la simulation d'un ordinateur minimaliste sur une machine de Turing.

A la fin de ce chapitre, nous disposerons donc du détail du fonctionnement de la machine de Turing, ce qui nous permettra d'étudier avec la complexité avec la finesse nécessaire.

1 Modèles de machine étudiés en théorie des langages

Nous allons maintenant considérer des premiers modèles théoriques de machines :

- les automates déterministes
- les automates non déterministes finis
- les automates à pile
- les grammaires algébriques

qui sont des modèles élémentaires de machines utilisé en théorie des langages, et en informatiques pour de nombreuses tâches.

Nous verrons pourquoi ces langages ont des capacités bien trop limitées d'où la nécessité de disposer d'un modèle plus évolué.

Il convient néanmoins de ne pas négliger ces premiers modèles formels car leurs principes de bases sont réutilisés dans les machines de Turing.

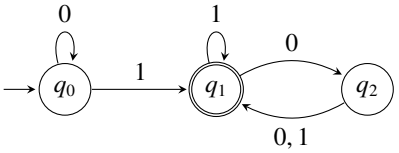
REMARQUES 21:

- Ces modèles de machine ne sont pas des machines programmables.
- Chaque machine n'est définie pour ne reconnaître qu'un seul langage.
- Ils sont donc l'équivalent d'un processeur spécialisé qui ne serait câblé que pour réaliser une seule tache.

1.1 Automate fini

Un automate déterministe fini (ADF) est une machine à état qui permet de résoudre un problème de décision pour un langage L.

Exemple 36:



Soit un alphabet $\Sigma = \{0, 1\}$. Pour décider un mot $w = s_1 \dots s_n \in \Sigma^*$ (où $n = |w|$ et $\forall i, s_i \in \Sigma$) avec cet ADF :

1. se placer sur le premier symbole s_1 de w ($i = 0$), et partir de l'état de départ q_1 (pointé par une flèche).
2. prendre la transition associée au symbole courant s_i et aller dans l'état correspond, puis passer au symbole suivant ($i = i + 1$).
3. recommencer (2) jusqu'à lire la totalité des symboles.
4. si l'état courant est un état acceptant (cercle double) alors accepter le mot, sinon le rejeter.

Remarque 22:

cet automate s'arrête toujours (longueur du mot d'entrée = nombre de transitions).

Il est déterministe :

- un symbole = une seule transition possible.
  pour chaque état, il y a une seule transition pour chaque symbole de l'alphabet.
- un mot = un seul chemin dans le graphe.

Définition IV.1 (formelle d'un automate déterministe fini)

Un automate déterministe fini est un 5-uple $(Q, \Sigma, \delta, q_0, F)$ où :

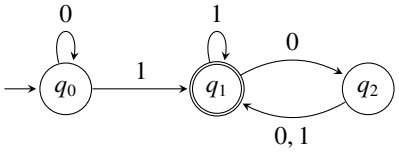
- Q est un ensemble fini d'états (ensemble des états).
- Σ est un ensemble fini de symboles (alphabet).
- $\delta : Q \times \Sigma \to Q$ est la fonction de transition.
- $q_0 \in Q$ est l'état de départ.
- $F \subseteq Q$ est l'ensemble des états acceptants.

Ce 5-uple est une description complète de l'ADF.

Remarques 23:

- La fonction de transition : $q' = \delta(q, s)$ est la fonction qui, lorsqu'on est dans l'état q et sur le symbole s, indique dans quel état q' on se retrouve.
- Si $F = \emptyset$ alors l'ADF n'accepte aucun mot ($L = \emptyset$). Si $F = Q$ alors l'ADF accepte tous les mots ($L = \Sigma^*$).

Exemple 37: de description d'un ADF



Cet ADF M a pour caractéristique :

- l'ensemble des états est $Q = \{q_1, q_2, q_3\}$
- l'alphabet est $\Sigma = \{0, 1\}$
- la fonction de transition $\delta : Q \times \Sigma \to Q$ (ou table de transition) est :

Q \ Σ	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

δ est une fonction de transition déterministe : pour chaque état $q \in Q$, il n'y a qu'une seule transition pour chaque symbole $s \in \Sigma$.

- q_1 est l'état de départ,
- l'ensemble des états acceptants est $F = \{q_2\}$.

Sa description formelle est $(Q, \Sigma, \delta, q_1, F)$.

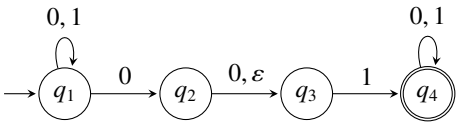
1.2 Automates non déterministes

Un automate non-déterministe fini (ANF) est un ADF modifié pour lequel, depuis un état, on modifie les transitions possibles de la manière suivante :

- un même symbole peut être associé à plusieurs transitions (voir aucun).
- un symbole supplémentaire (noté ϵ) permet une transition sans lecture de symbole sur la chaîne d'entrée.

L'automate n'est alors plus déterministe car plusieurs choix peuvent être possibles lors d'une transition, ou aucun !

Exemple 38:



- transitions supplémentaires : q_1 a deux transitions associées à 1.
- transition manquante : q_2 n'a pas de transition pour 1 (aussi q_3 pour 0).
- transition ϵ : la transition entre q_2 et q_3 peut se faire sans lecture de symbole.

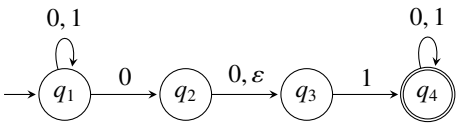
Un ANF s'exécute de la façon suivante :

- si plusieurs transitions sont possibles :
  - la machine se découple en des copies multiples.
  - chaque copie (= une branche d'exécution) suit une possibilité.
  - l'ensemble des branches continuent et suivent toutes les possibilités.
- une transition ϵ correspond à une transition sans lecture de symbole.
- si aucune transition n'est possible
  la branche d'exécution ne peut pas se poursuivre : elle s'arrête (=rejet)

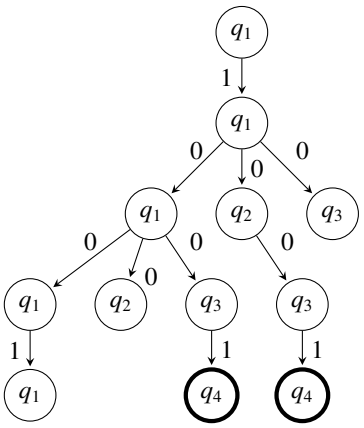
La chaîne d'entrée est :

- **acceptée** si au moins l'une de ses branches d'exécution accepte (= est dans un état acceptant à la fin de la lecture de la chaîne).
- **rejetée** si toutes les branches d'exécution ont rejeté ou se sont arrêtées avant la fin de la chaîne.

Exemple 39: ANF :



Exécution sur le mot 1001 **:**



Note : pour une transition ϵ, $q_1 \xrightarrow{0} q_2 \xrightarrow{\epsilon} q_3$ devient $q_1 \xrightarrow{0} q_3$.

Définition IV.2 (Fonction de transition)

La fonction de transition δ est définie de $Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$
où Q est l'ensemble des états, Σ l'alphabet du langage et $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$.

Exemple 40: On définit l'ensemble des états $Q = \{q_1, q_2, q_3\}$ et l'alphabet $\Sigma = \{0, 1\}$.

- $\mathcal{P}(Q)$ contient bien l'ensemble de toutes les transitions possibles entre un état et tous les autres (voir exemple ci-avant).
- Σ_ϵ est l'ensemble des symboles provoquant une transition (= symboles de l'alphabet + transition ϵ).
- $\delta(q, s)$ est définit pour tout $q \in \mathcal{P}(Q)$ et $s \in \Sigma_\epsilon$ (i.e. sur le produit cartésien $\mathcal{P}(Q) \times \Sigma_\epsilon$).

Exemples de transitions :

- $\delta(q_1, \epsilon) = \{q_2\}$: transition simple $q_1 \xrightarrow{\epsilon} q_2$
- $\delta(q_2, 0) = \{q_1, q_3\}$: transitions multiples $q_3 \xleftarrow{0} q_2 \xrightarrow{0} q_1$
- $\delta(q_1, 1) = \{\}$: pas de transition pour le symbole 1 depuis q_1

Définition IV.3 (description formelle d'un ANF)

Un automate non-déterministe fini est un 5-uple $(Q, \Sigma, \delta, q_0, F)$ où :

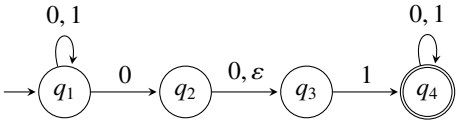
- Q est un ensemble fini d'états(ensemble des états).
- Σ est un ensemble fini de symboles (alphabet).
- $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ est la fonction de transition.
- $q_0 \in Q$ est l'état de départ.
- $F \subseteq Q$ est l'ensemble des états acceptants.

Le 5-uple $(Q, \Sigma, \delta, q_0, F)$ définit complètement l'automate non-déterministe fini.

Remarque : par rapport à la définition formelle d'un ADF seule la définition de la fonction de transition indique que l'on a affaire à un ANF.

Exemple 41: description formelle d'un ANF

Soit l'ANF suivant :



$N = (Q, \Sigma, \delta, q_1, F)$ où :

- états : $Q = \{q_1, q_2, q_3, q_4\}$
- langage : $\Sigma = \{0, 1\}$
- fonction de transition : $\delta(q, s)$

Q \ Σ	0	1	ϵ
q_1	$\{q_1, q_2\}$	$\{q_1\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

  δ est une fonction de transition non-déterministe : pour chaque état $q \in Q$, il n'y a qu'une ou plusieurs transitions pour chaque symbole $s \in \Sigma$, voire **aucune** ($\emptyset$ dans la table ci-contre).
- état de départ : q_1
- états acceptants : $F = \{q_4\}$

1.3 Automates à pile

Un Automate à Pile (AP) est un Automate Non-déterministe Fini (ANF) auquel a été ajouté pile LIFO [1] (=FILO) de taille infinie.

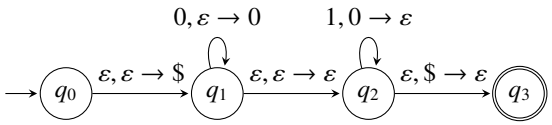
A chaque transition, un automate à pile :

- lit un caractère de la chaîne d'entrée (sauf s'il s'agit d'une transition ϵ)
- dépile (éventuellement) le symbole au sommet de la pile.
- empile (éventuellement) un symbole sur la pile.

La notation $a, b \to c$ signifie que la machine lit a depuis l'entrée, dépile b et pousse c sur la pile. ε signifie une absence d'opération.

1. Last In Firt Out.

Exemple 42:



Fonctionnement :

tant que l'on lit un 0 : le pousser sur la pile.

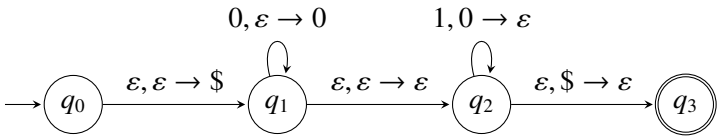
tant que l'on lit un 1 : dépiler un 0 de la pile, et s'il n'y en a pas, rejeter.

si la pile est vide, alors accepter, sinon rejeter.

Le langage reconnu est $L = \{0^n 1^n \mid n \geq 0\}$.

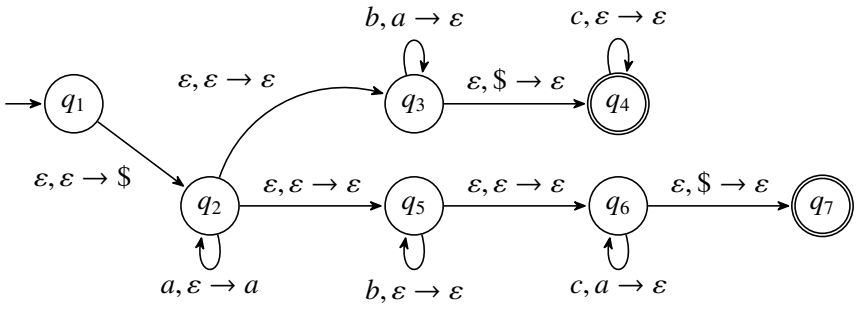
Remarques 24:

- on ne peut pas tester si la pile est vide.
  Une solution consiste à utiliser un symbole particulier (par exemple \$).
  On commence avant le début de la lecture par placer un \$ au sommet de la pile (transition de la forme $\epsilon, \epsilon \to \$$).
  Si on lit le symbole \$ sur le sommet de la pile, alors la pile est vide.
- on ne peut pas tester si la chaîne d'entrée a été traitée.
  Comme pour un ADF, la chaîne d'entrée est acceptée si après le dernier symbole lu, on se trouve dans un état acceptant.
- Dans l'exemple précédent (reproduit ci-dessous), le non-déterminisme est utilisé :
  — pour empiler autant de 0 à la lecture des 0s qu'il sera nécessaire à d'en dépiler à la lecture des 1,
  — pour s'assurer que la pile est vide lorsque l'on a fini de lire tous les symboles

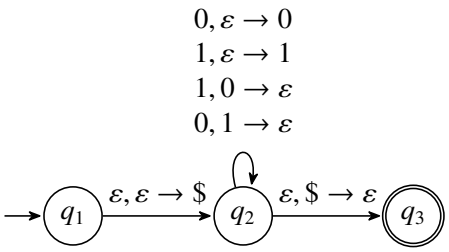


Exemple 43: $L = \left\{a^i b^j c^k \mid i, j, k \geq 0 \text{ et } i = j \text{ ou } i = k\right\}$
Comment construire l'automate à pile qui reconnaît ce langage ?

- pousser les a lus sur la pile.
- deviner si le nombre de a doit être comparé avec le nombre de b ou de c (fait par une transition non déterministe).



Exemple 44: $L = \{w \mid \#_0 w = \#_1 w\}$ ou $\#_i w$ = nombre de i dans la chaîne w.
Comment construire l'automate à pile qui reconnaît ce langage ?

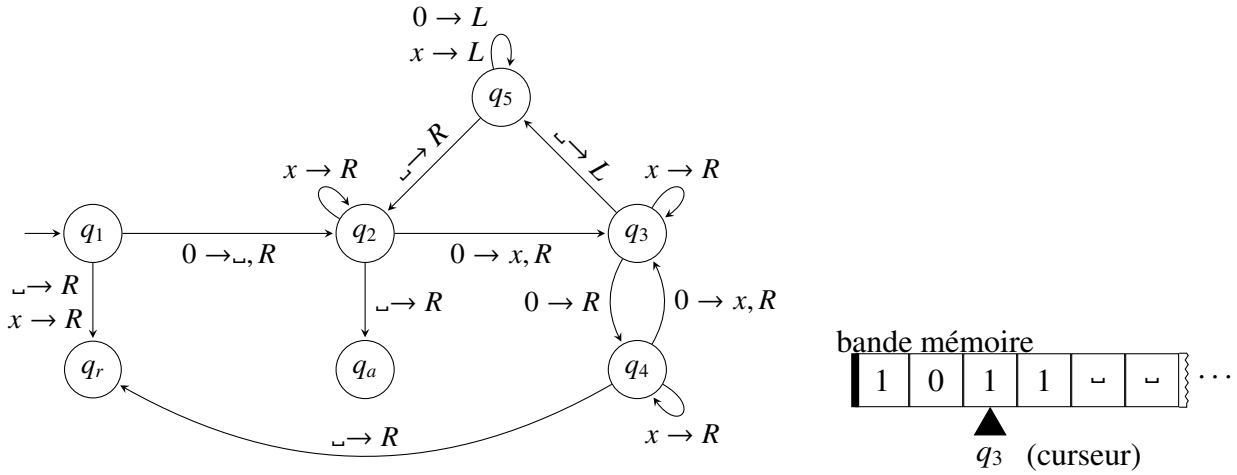


- si le sommet de la pile est 0 (= il n'y a que des 0 dans la pile)
  - si je lis un 1, alors je dépile le 0.
  - si je lis un 0, alors j'empile le 0.
- si le sommet de la pile est 1 (= il n'y a que des 1 dans la pile)
  - si je lis un 0, alors je dépile le 1.
  - si je lis un 1, alors j'empile le 1.
- à savoir, on dépile toujours autant de 0 (resp. 1) qu'on lit de 1 (resp. 0).
- si la pile est vide, le symbole suivant détermine celui qui sera empilé.
- le non déterminisme permet de trouver le bon chemin.

2 Machine de Turing

Une machine de Turing (MT) est un modèle abstrait de machine :

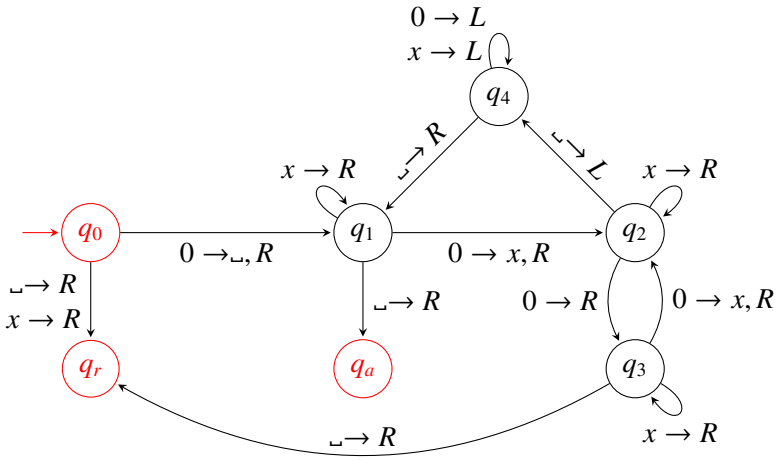
- qui prend en paramètre un mot d'entrée w,
- avec une bande mémoire (finie à gauche) permettant d'y stocker des symboles, et initialisée avec le mot d'entré,
- un curseur permet de se déplacer sur la bande (dans les deux sens), afin de lire ou écrire le symbole sous le curseur.
- le curseur est contrôlé par une machine à état déterministe,
- les transitions s'exécutent jusqu'à accepter ou à rejeter.



Une MT a donc pour but de résoudre des **problèmes de décision** (*i.e.* de décider des langages).

Une MT dispose de **trois états particuliers** :

- un **état de départ** (ci-dessous q_0) qui est le point de départ de l'exécution du graphe d'état, marquée par une petite flèche entrante.
- un **état acceptant** q_A : lorsqu'il est atteint, la machine s'arrête immédiatement en acceptant.
- un **état rejetant** q_R : lorsqu'il est atteint, la machine s'arrête immédiatement en rejetant.



Remarques 25:

- Les états q_A et q_R n'ont que des connexions entrantes (puisqu'elles provoquent l'arrêt de la machine),
- Si la machine n'entre jamais dans l'un des états q_A ou q_R, **elle boucle**.

Une MT dispose de **deux alphabets différents** :

- l'alphabet du langage Σ auquel appartient le mot d'entrée w (*i.e.* $w \in \Sigma^*$).
- l'alphabet de bande Γ contient l'ensemble des symboles qui peuvent être trouvé sur la bande.

Γ contient évidemment Σ mais également :

- le caractère blanc $\sqcup$ qui est le caractère spécifique avec lequel l'ensemble des cellules de la bande est initialisé.

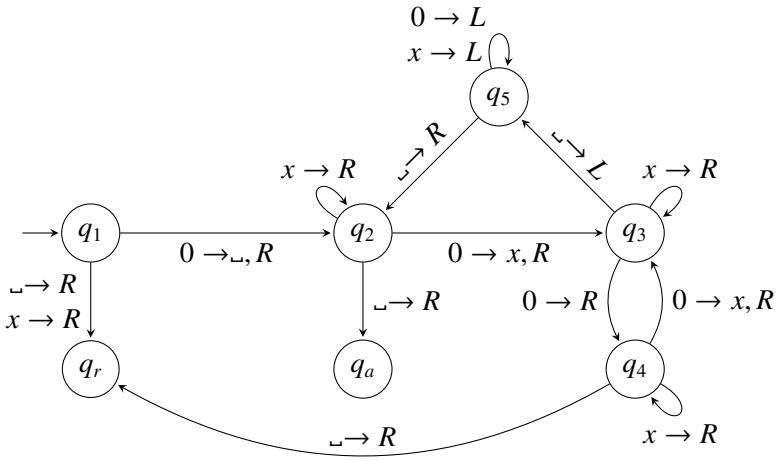
- tout autre caractères spécifiques auquel on affecte un rôle ou un sens particulier lors de l'exécution de la machine, par exemple :
  - des caractères spéciaux :
    # pour indiquer le début de la bande,
    `x` pour barrer un symbole déjà traité,
  - des marqueurs : pour un symbole 0, on peut le barrer Ø, le marquer de différente manière $\bar{0}$, $\dot{0}$, $\tilde{0}$, $\hat{0}$ afin d'indiquer un sens particulier tout en conservant le symbole, et éventuellement rétablir l'original.

Au minimum, on a $\Gamma = \Sigma \cup \{\sqcup\}$.
A une étape de l'exécution, l'état d'une MT est caractérisé par :

- l'état courant q_i dans lequel on se trouve sur le graphe d'état,
- l'endroit de la bande où est placé le curseur de bande,
- le symbole contenu dans la cellule pointée par le curseur de bande,

La transition $a \to b, L$ signifie : si la cellule pointée par le curseur contient le symbole a, alors écrire un b dans la cellule, et déplacer le curseur dans la direction L (gauche).

Une machine de Turing est **déterministe**, à savoir que **pour chaque état de la machine**, **pour chaque symbole de bande**, il n'y a qu'une seule transition possible depuis cet état.

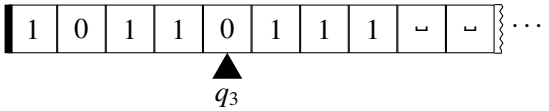


Dans l'exemple précédent, les symboles de l'alphabet sont $\Sigma = \{0\}$, les symboles de bande sont $\Gamma = \{0, x, \}$. Depuis chaque état, il y a une seule transition pour chaque état de Γ.

Définition IV.4 (Configuration d'une machine de Turing)

Notation qui contient l'**état**, la **position du pointeur**, le **contenu de la bande** d'une MT à un instant donné.

Exemple 45: Une configuration $1011q_30111$ signifie que la MT est dans l'état suivant :



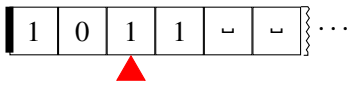
à savoir, la machine est dans l'état q_3, la bande contient 10110111, et le pointeur se trouve sur le cinquième caractère de la bande (= sur le symbole qui suit l'état dans la configuration).

Comme une machine de Turing est déterministe, une configuration décrit totalement l'état d'une MT ainsi que son évolution future.

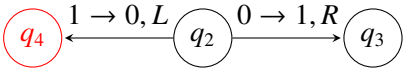
Exemple 46: On est dans l'état q_2 et le symbole dans la cellule sous le curseur de lecture est 1.

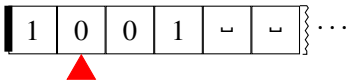
extrait du graphe de la machine de Turing

bande mémoire



On choisit donc la transition $1 \to 0, L$: on écrit 0, on déplace le curseur à gauche. et on passe dans l'état q_4.



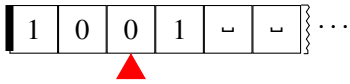


La machine passe donc de la configuration $10q_211$ à la configuration $1q_4001$.

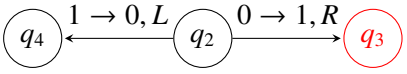
Exemple 47: On est dans l'état q_2 et le symbole dans la cellule sous le curseur de lecture est 0.

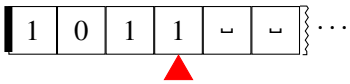
extrait du graphe de la machine de Turing

bande mémoire



On choisit donc la transition $0 \to 1, R$: on écrit 1, on déplace le curseur à droite. et on passe dans l'état q_3.





La machine passe donc de la configuration $10q_201$ à la configuration $101q_31$.

Plusieurs types de transitions sont possibles :

- Les transitions en lecture :
  $0 \to L$ = si le symbole courant est un 0, déplacer le curseur à gauche.
  $1 \to R$ = si le symbole courant est un 1, déplacer le curseur à droite.
- Les transitions en écriture :
  $0 \to 1, L$ = si le symbole courant est un 0, écrire un 1 à la place sur la bande, et déplacer le curseur à gauche.
  $1 \to 1, R$ = si le symbole courant est un 1, écrire un 1 à la place sur la bande (donc ne change rien), et déplacer le curseur à droite.

On utilise aussi les notations suivantes :

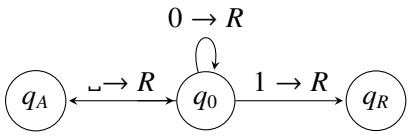
- $0, 1 \to L$ = si le symbole courant est 0 ou 1, déplacer le curseur à gauche.
- $\Sigma \to R$ = pour tout symbole de l'alphabet Σ, déplacer le curseur à droite.

Si on se déplace à gauche alors que le curseur est déjà au début de la bande, alors le curseur ne bouge pas.

Soit M une machine de Turing, et L le langage accepté par la machine. Le fonctionnement de M sur un mot d'entrée w est le suivant :

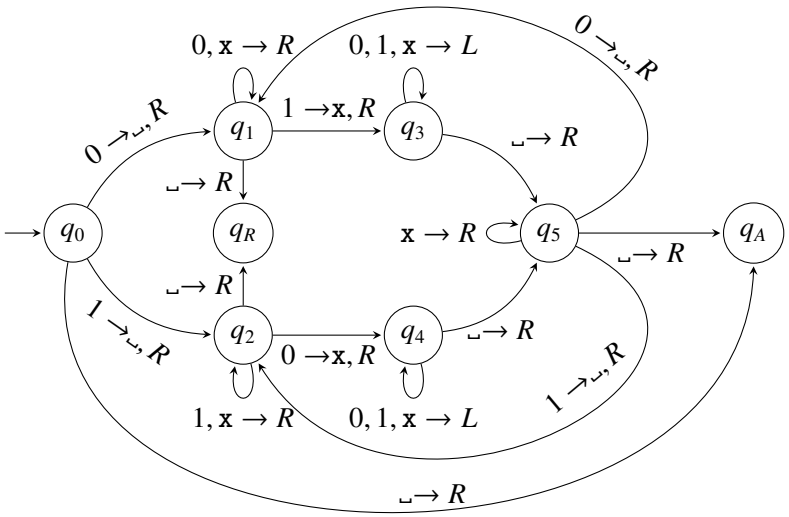
- **initialisation :**
  - placer le mot w au début de la bande mémoire,
  - placer le curseur sur la première cellule de la bande,
  - initialiser l'état courant à l'état de départ q_0 du graphe d'état.
- **exécution :**
  - en fonction de l'état courant et du symbole sous le curseur, effectuer la transition associée (écrire le nouveau symbole dans la cellule, déplacer le curseur, et passer à l'état suivant).
  - si le nouvel état est q_A, alors M s'arrête en acceptant ($w \in L$).
    si le nouvel état est q_R, alors M s'arrête en rejetant ($w \notin L$).
  - pour tous les autres états, recommencer avec ce nouvel état.

Exemple 48: langage qui ne contient que des 0



Remarquez que la machine s'arrête dès que le premier 1 est rencontré, sans lire le reste de la chaîne.

Exemple 49: langage qui contient autant de 0 que de 1

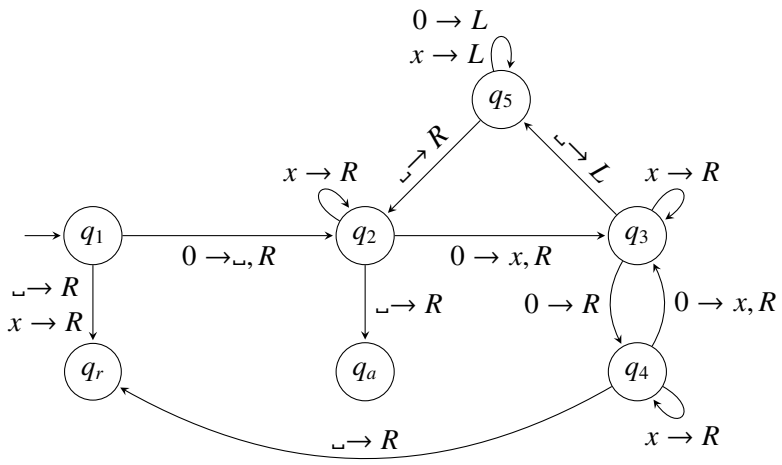


Exemple 50: Soit la machine de Turing M qui décide $L = \{0^{2^n} \mid n \geq 0\}$.

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.

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.



$q_1 0\,0\;\;0\,0 \sqcup$
$\sqcup q_2 0\;\;0\,0 \sqcup$
$\sqcup\, x q_3\, 0\,0 \sqcup$
$\sqcup\, x\, 0\, q_4 0 \sqcup$
$\sqcup\, x\, 0\;\; x q_3 \sqcup$
$\sqcup\, x\, 0\, q_5 x \sqcup$
$\sqcup\, x q_5\; 0\; x \sqcup$
$\sqcup q_5 x\;\; 0\; x \sqcup$
$q_5 \sqcup x\;\; 0\; x \sqcup$
$\sqcup q_2 x\;\; 0\; x \sqcup$
$\sqcup\, x q_2\; 0\; x \sqcup$
$\sqcup\, x\, x\, q_3 x \sqcup$
$\sqcup\, x\, x\;\; x q_3 \sqcup$
$\sqcup\, x\, x\, q_5 x \sqcup$
$\sqcup\, x q_5\; 0\; x \sqcup$
$\sqcup q_5 x\;\; 0\; x \sqcup$
$q_5 \sqcup x\;\; 0\; x \sqcup$
$\sqcup q_2 x\;\; 0\; x \sqcup$
$\sqcup\, x q_2\; 0\; x \sqcup$
$\sqcup\, x\, x\, q_3 x \sqcup$
$\sqcup\, x\, x\;\; x q_2 \sqcup$
$\sqcup\, x\, x\;\; x \sqcup q_a$

Trouver la fin de la chaîne est facile. C'est le premier caractère blanc rencontré lorsqu'on déplace le curseur de lecture à droite.
Il est plus difficile de trouver le début de la chaîne :

- soit il faut y placer un caractère spécial (par exemple # ou ␣)
  dans ce cas, il faut remplacer le premier caractère. On a donc deux possibilités :

  — si on n'a plus besoin du premier caractère car on n'a déjà pris en compte sa valeur, on peut le remplacer directement,
  — sinon, on insère le caractère au début de la bande et on décale la totalité de la chaîne d'entrée d'un caractère vers la droite.

  Trouver le début de la chaîne est alors facile : se déplacer à gauche jusqu'à trouver le caractère de début de chaîne.

- soit on utilise le fait que le curseur ne bouge pas lorsqu'on est en début de bande de la manière suivante :

  — on retient le caractère courant, on le remplace par #, et on va à droite.
  — si on relit le même symbole #, alors on est au début de la bande.
  — on rétablit le caractère courant et on poursuit l'exécution en fonction du cas dans lequel on se trouve.

  Le procédé est à recommencer à chaque fois qu'on veut tester si on est en début de chaîne.

Exercice 25 (détection du début de la chaîne)**.** *1. écrire la machine de Turing qui insère un caractère spécial dans la première cellule et décale les caractères de la chaîne d'entrée d'un caractère à droite.*

2. *écrire alors la machine de Turing qui permet de retourner en début de chaîne indépendamment de l'endroit où se trouve le curseur de bande.*
3. *écrire la machine de Turing qui teste si le curseur est en début de bande en utilisant le fait que le curseur de lecture ne bouge pas lorsqu'on va à gauche en début de bande.*

Exercice 26 (premiers langages)**.** *Écrire la machine de Turing qui reconnaît les mots :*

1. *de la forme* 0^n1^n *avec* $n \geq 0$.
2. *de la forme* $0^n1^n2^n$ *avec* $n \geq 0$.
3. *qui commencent et se terminent par les deux mêmes caractères pour* $\Sigma = \{0, 1\}$.
4. *de la forme* $\#w\#w$ *avec* $w = \{0, 1\}^*$.
5. *de la forme* $\#w\#w^R$ *avec* $w = \{0, 1\}^*$ *où* w^R *est le mot écrit à l'envers.*
6. *de la forme* ww^R *avec* $w = \{0, 1\}^*$.
7. w *tels que* $w = w^R$ *avec* $w = \{0, 1\}$.

Exercice 27 (Automate déterministe fini)**.** *Décrire une méthode permettant de transformer tout automate déterministe fini en sa machine de Turing équivalente (i.e. qui accepte le même langage).*

Exercice 28. *Montrer qu'une machine de Turing à bande unique qui ne peut pas écrire sur la partie de la bande contenant la chaîne d'entrée ne reconnaît que les langages réguliers.*

Autre façon de décrire une machine de Turing : décrire le "programme" que la machine exécute en décrivant les mouvements du curseur et les actions sur la bande.

Exemple 51: Construction de la MT qui décide $L = \{a^ib^jc^k \mid ij = k \text{ où } i, j, k \geq 1\}$

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.

Exercice 29 (premiers programmes)**.** *Écrire le code de la machine de Turing qui reconnaît les langages suivants :*

1. $L_1 = \{0^n1^n2^n \text{ où } n \geq 0\}$
2. $L_2 = \{ww \mid w \in \{0, 1\}^*\}$
3. $L_3 = \left\{0^i1^j2^k \mid i < j < k\right\}$
4. $L_4 = \left\{\#x_1\#x_2\#\ldots\#x_k \mid x_i \in \Sigma^* \text{ et } \forall i \neq j, x_i \neq x_j\right\}$ *avec* $\Sigma = \{0, 1\}$.

2.1 Définition

Notations :

- $Q = \{q_0, q_1, \ldots\}$ ensemble des états ; $\{q_A, q_R\}$ exclus.
- $Q' = Q \cup \{q_A, q_R\}$
- Γ alphabet de la bande = alphabet des symboles du langage + symboles de bande supplémentaires.
- $\{L, R\}$ mouvement du pointeur sur la bande (L = gauche, R = droite).

Fonction de transition $\delta : Q \times \Gamma \to Q' \times \Gamma \times \{L, R\}$
$\delta(q, a) = (r, b, L)$ 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 seule transition possible qui écrit sur la bande une valeur unique et se déplace dans la direction déterminée associée.

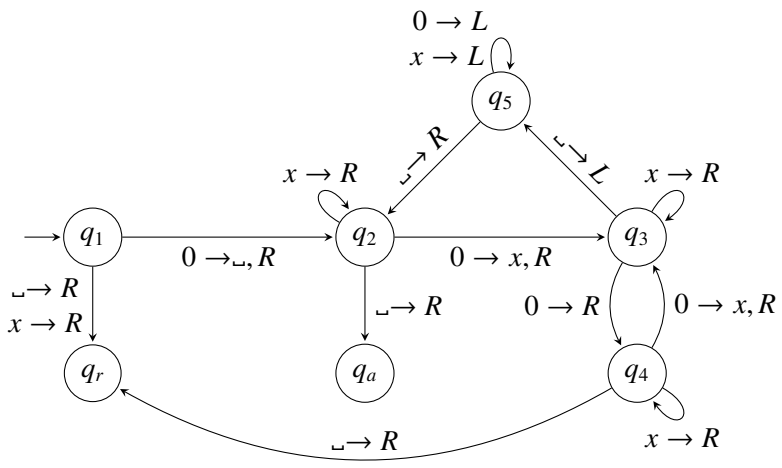
D'où la définition formelle d'une machine de Turing :

Définition IV.5

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

- q_a est l'état acceptant, et q_r est l'état rejetant.
- Q est l'ensemble fini des états (q_a et q_r exclus). On ote $Q' = Q \cup \{q_A, q_R\}$.
- $q_0 \in Q$ est l'état de départ.
- Σ est l'alphabet du langage
- Γ est l'alphabet de la bande
- $\delta : Q \times \Gamma \to Q' \times \Gamma \times \{L, R\}$ est la fonction de transition.

Exemple 52: de description formelle



Description formelle de la MT $M = (Q, \Sigma, \Gamma, \delta, q_1, q_a, q_r)$ associée avec $Q = \{q_1, q_2, q_3, q_4, q_5\}$, $\Sigma = \{0\}$, $\Gamma = \Sigma \cup \{\mathtt{x}, \sqcup\}$

$\delta(q, r)$	0	x	$\sqcup$
q_1	$(q_2, \sqcup, R)$	$(q_r, \mathtt{x}, R)$	(q_r, R)
q_2	$(q_3, \mathtt{x}, R)$	$(q_2, \mathtt{x}, R)$	$(q_a, \sqcup, R)$
q_3	$(q_4, 0, R)$	$(q_3, \mathtt{x}, R)$	$(q_5, \sqcup, L)$
q_4	$(q_3, \mathtt{x}, R)$	$(q_4, \mathtt{x}, R)$	$(q_r, \sqcup, R)$
q_5	$(q_5, 0, L)$	$(q_5, \mathtt{x}, L)$	$(q_2, \sqcup, R)$

Exercice 30. *Donner la description formelle de la machine de Turing qui reconnaît le langage contenant autant de 0 que de 1 (voir le graphe donné dans l'exemple 49)*

2.2 Modèle de calcul d'une machine de Turing

On rappelle que, pour une machine de Turing déterministe, une configuration définit complètement l'état de la machine.

Le déterminisme fait aussi qu'une configuration donnée ne peut conduire qu'à une seule autre configuration.

Il existe certaines configurations particulières :

- si w est le mot mis en entrée de la machine, et q_0 l'état de départ, alors $q_0 w$ est la configuration de départ.
- si C est une configuration qui contient l'état q_a, alors c'est une configuration acceptante,
- si C est une configuration qui contient l'état q_r, alors c'est une configuration rejetante.

Ainsi, toute configuration de départ ne produit qu'une seule exécution possible, et donc, qu'une seule suite de configurations.

Définition IV.6 (chaîne acceptée par une MT M)

M accepte un chaîne w en entrée s'il existe une suite de configurations $C_1, C_2, \cdots C_k$ 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 IV.7 (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 : $L = \mathcal{L}(M)$

3 Autres modèles de machine de Turing

Le modèle de la machine de Turing est simple, mais l'ensemble des langages reconnus les MTs change-t-il si on utilise les améliorations suivantes :

- une commande S (=stay) qui permet d'effectuer une transition sans bouger le pointeur sur la bande,
- un alphabet avec plus de symboles (#Σ dans sa définition formelle) ?
- une bande de travail infinie des deux cotés ?
- plusieurs bandes de travail ?
- en utilisant des transitions non déterministes ?

Afin de démontrer ces points, la ligne de démonstration sera toujours la même :

Est-il possible de simuler sur le modèle de MT le plus simple (avec un alphabet binaire) la MT améliorée ?

Par simuler, on entend : prendre les mêmes décisions (accepter, rejeter ou boucler) que la machine que l'on simule **en utilisant une machine de Turing classique**.

Si la réponse est oui, alors les deux modèles reconnaissent exactement le même ensemble de langages. 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 cotés,
- 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.

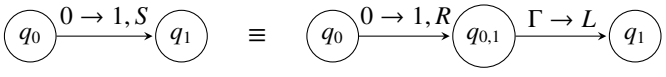
3.1 Ajout de commande

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 :

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

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 L puis R (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.

On montre ainsi l'équivalence entre les MT classiques et les MT LRS.

3.2 Restriction d'alphabet

La taille de l'alphabet a-t-il une influence sur l'ensemble des langages qu'il est possible de reconnaître ?

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

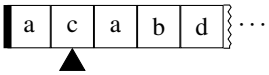
Évidemment, cette simulation ne peut s'effectuer qu'à travers l'encodage des symboles de Σ dans $\mathbb{B}$.

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

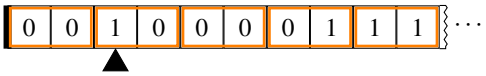
La bande est découpée en blocs de p bits dont chacun représente un symbole de Σ.

Exemple 53: Codage avec l'alphabet binaire $\Sigma_\mathbb{B}$ de l'alphabet $\Sigma = \{a, b, c, d\}$: $a = 00$, $b = 01$, $c = 10$, $d = 11$.

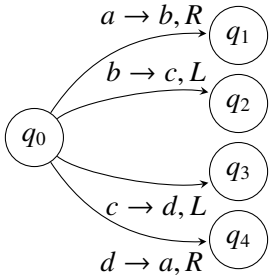
bande sur M_Σ



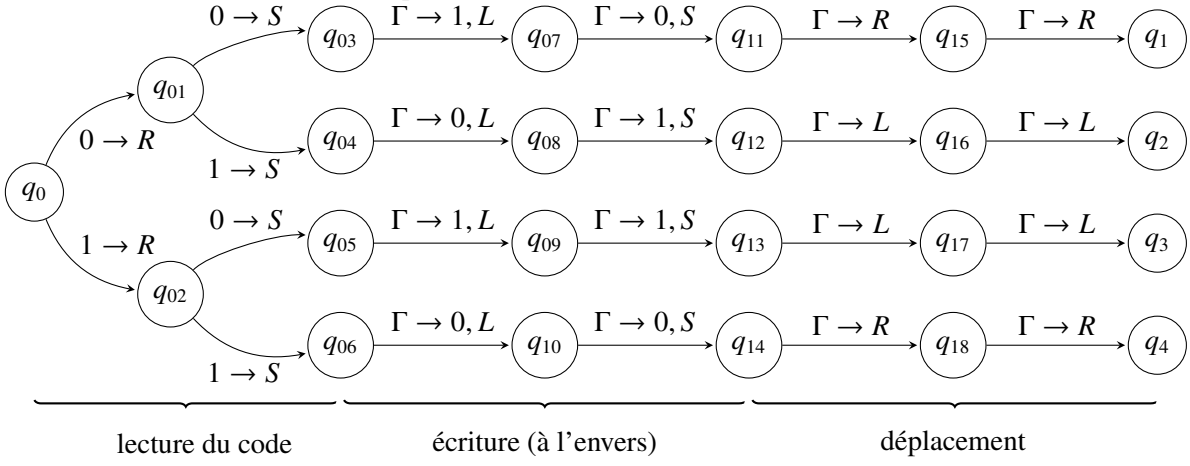
bande sur $M_\mathbb{B}$



extrait de machine sur M_Σ



extrait de la même machine sur $M_\mathbb{B}$



Montrons que le comportement de M_Σ sur tout mot d'entré peut être simulé sur une machine $M_\mathbb{B}$ sur ce même mot d'entré codé en binaire.

Sur $M_\mathbb{B}$, la simulation d'une transition $a \to b, L$ (resp. $a \to b, R$) 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.

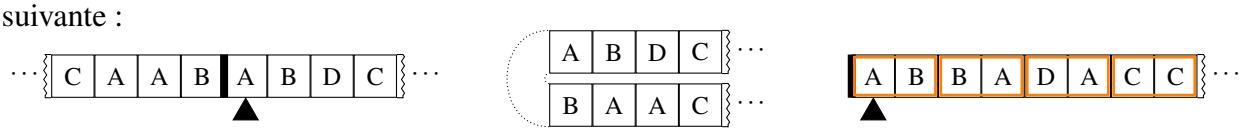
Conséquences : une MT M_Σ peut être simulée sur une MT $M_{\mathbb{B}}$.

- Elle n'est pas plus puissante qu'une $M_{\mathbb{B}}$,
- La simulation $M_{\mathbb{B}}$ augmente d'un facteur $3.\lceil \log_2 \#\Sigma \rceil$ le nombre de transitions nécessaires sur une entrée par rapport à M_Σ.

3.3 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 :

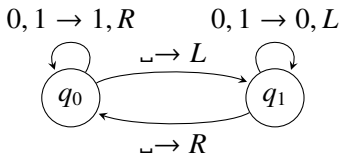


- "plier" la bande en deux à partir de son milieu (= position à laquelle le pointeur de lecteur est initialisé).
- chaque symbole de la bande est codé par un couple de symboles.
- pour un déplacement sur la portion droite (resp. gauche), considérer le symbole de droite (gauche) (= une version de M_b 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.

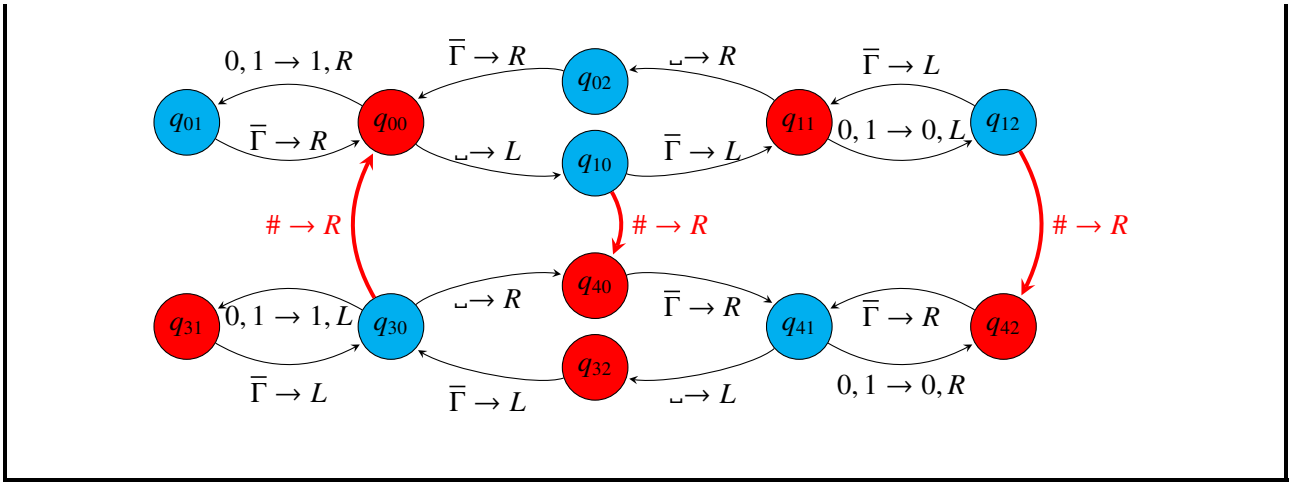
- Elle n'est pas plus puissante qu'une M,
- La simulation M_b augmente d'un facteur multiplicatif 2 le nombre de transitions nécessaires sur une entrée par rapport à M.

Exemple 54: Machine originale : va à droite en écrivant des 1, puis va à gauche en écrivant des 0, et recommence entre le début et la fin de la bande.



Machine avec bande doublement infinie :

- Ajouter des états intermédiaires pour passer un symbole sur deux.
- Dupliquer la machine en échangeant les sens d'exécutions
- Relier les deux machines lorsque le caractère de pliage de bande est rencontré en allant à gauche.



3.4 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 $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$
  $\delta(q_i, a_1, \cdots, a_k) = (q_j, b_1, \cdots, b_k, L, \cdots, R)$ signifie :

  si la machine est dans l'état q_i, et que les pointeurs $\{1, \cdots, k\}$ lisent $\{a_1, \cdots, a_k\}$ sur les k bandes,

  alors la machine passe à l'état q_j, les pointeurs $\{1, \cdots, k\}$ écrivent $\{b_1, \cdots, b_k\}$ sur les k bandes puis bougent respectivement dans les directions $\{L, \cdots, R\}$.

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).

Exercice 31. *Écrire une MT multibande (avec au moins deux bandes) qui reconnaît :*

1. *le langage* $0^n 1^n$.
2. *l'ensemble palindromes.*
3. *l'ensemble des mots ayant le même nombre de 0 et de 1.*
4. *le langage* $\left\{\#x_1\#x_2\#\ldots\#x_k \mid x_i \in \Sigma^* \text{ et } \forall i \neq j, x_i \neq x_j\right\}$ *avec* $\Sigma = \{0, 1\}$.

Exercice 32. *Expliquer comment simuler un automate à pile avec une machine de Turing multibande, avec l'une des bandes qui joue exactement le rôle de la pile.*

Théorème IV.1 (équivalence MT simple bande - MT multi-bande)

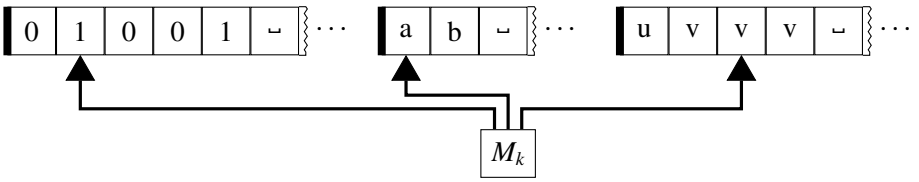
Toute MT M_k a k bandes a une MT M simple-bande équivalente.

DÉMONSTRATION: (constructive)

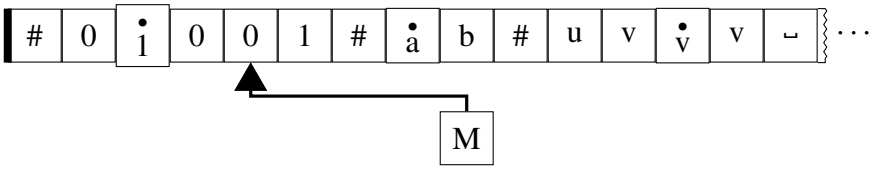
On stocke les k bandes de M_k sur la bande de M avec :

- 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.

Exemple de MT multibande :



Equivalent simple bande :



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

1. **initialisation de la bande** : $\#\dot{w}_1 w_2 \cdots w_n \#\dot{\sqcup}\#\dot{\sqcup} \cdots \#\dot{\sqcup}$
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. □

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.

3.5 Machine de Turing non déterministe

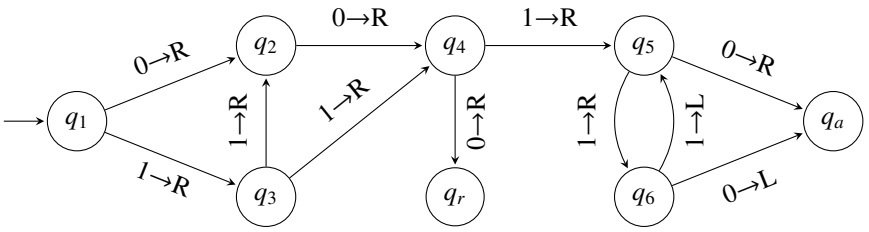
Une machine de Turing non déterministe (MTND) est une généralisation similaire à celle des ADFs en ANFs :

- 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 découple en autant de copies que nécessaires, chaque copie s'occupant d'une branche d'exécution.

Comportement possible d'une MTND :

- si une branche accepte, alors la MTND s'arrête en acceptant.
  peu importe le comportement des autres branches (rejet ou boucle).
- si toutes les branches rejettent, alors la MTND s'arrête en rejetant.
  = aucune branche de calcul n'a amené à accepter la chaîne d'entrée.
- si aucun branche n'accepte, et qu'**au moins une branche boucle**, alors la MTND boucle.
  branche qui boucle = calcul sur la branche non terminé ⇒ pas de décision.

Exemple : avec $\Sigma = \{0, 1\}$



- q_1,q_4, q_6 : états avec transition déterministe (une transition par symbole de Σ).
- q_3, q_5 : états avec une transition non déterministe.
- q_2 : n'a pas de transition pour 1 (=arrêt de la branche si cela est le cas).
  exemple : les mots commençant 01 ou 10 sont rejetés sans que la machine n'atteigne l'état q_r.
- la transition 1 pour q_5 et q_6 peut produire une boucle.
  exemple : pour le mot 001110, la machine boucle.

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.

On considère donc qu'un mot qui fait boucler une machine n'appartient pas au langage que cette dernière reconnaît.

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.

Exercice 33. *Toutes les MTNDs proposées devront utiliser activement le non-déterminisme.*

1. *écrire une MTND qui reconnaît les mots qui commencent et se terminent par les deux mêmes caractères. On prendra* $\Sigma = \{0, 1\}$.
2. *écrire une MTND qui reconnaît les mots contenant la chaîne* 011.
3. *écrire une MTND qui reconnaît les mots de la forme ww.*
4. *écrire une MTND qui reconnaît le langage* $0^i1^j2^k$.
5. *écrire le code d'une MTND qui trouve deux nombres binaires* w_1 *et* w_2 *de la même longueur que le mot d'entrée tels que* $w_1 \oplus w_2 = 0$ *(où* $\oplus$ *est l'opérateur* `XOR`*).*

La fonction de transition est de la forme :
$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$

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 IV.2 (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'.

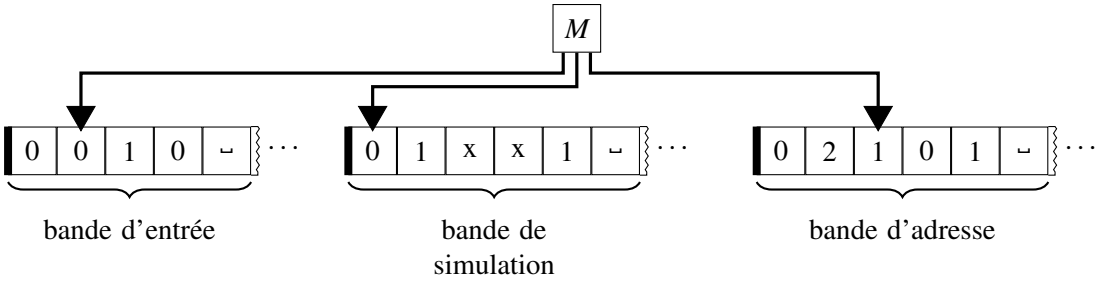
- M essaie toutes les branches possibles.
- si M accepte l'une de ces branches, alors M accepte.
- si toutes les branches sont rejetées, alors M rejette.
- si toutes les branches sont rejetées ou bouclent, alors M boucle.

L'exécution de M est un arbre :

- chaque branche de l'arbre est une branche de l'exécution non déterministe de M'.
- le nœud de l'arbre est la configuration initiale.
- chaque nœud de l'arbre est une configuration de M.
- le nombre d'enfants d'un nœud 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 nœud a au plus n enfants.
- à chaque nœud, on peut affecter une adresse qui est une chaîne dans l'alphabet $\Delta = \{1, \cdots, n\}$ chaque symbole correspondant à une configuration.
- **utilisation d'une adresse** :
  pour se rendre au nœud 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 IV.3 (simulation d'une MTND sur une MT (non démontré))

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

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

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.
  Ce point sera étudié dans le chapitre sur la complexité temporelle.

3.6 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 peut être simulées sur une MT classique et n'entraine qu'au plus un gain quadratique (proportionnel à $k.t^2$ 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 repréciserons.

4 Fonction calculable

Nous avons vu que les machines de Turing étaient une classe de machine permettant de résoudre (au moins) certains problèmes de décision.

Intéressons-nous à une modification d'une machine de Turing afin de résoudre (au moins) certains problèmes d'évaluations.

On parle alors de fonctions (partiellement) calculables.

Pour ce faire, nous allons aussi utiliser des machines de Turing dont la seule caractéristique supplémentaire sera de s'arrêter avec l'évaluation attendu sur sa bande.

4.1 Définitions

Définition IV.8 (fonction (totalement) calculable)

Une MT M calcule totalement une fonction $f : \Sigma^* \to \Sigma^*$ si M

- commence avec l'entrée w sur sa bande.
- s'arrête pour tout w et avec seulement $f(w)$ écrit sur sa bande.

Remarques 26:

- marche aussi pour les fonctions avec plus d'une variable.
  Encoder les paramètres sur la chaîne d'entrée intercalée de séparateurs.
  Chaque paramètre peut représenter n'importe quel objet.
- même remarque pour la sortie.
- une bande particulière peut être utilisée pour la sortie.

Notation : on note $\perp$ la sortie d'une fonction calculable si l'entrée n'est pas son ensemble de définition.

Définition IV.9 (fonction partiellement calculable)

Une MT M calcule partiellement une fonction $f : \Sigma \to \Sigma \cup \perp$ si M :

- commence avec l'entrée w sur sa bande.
- si $f(w)$ est défini, alors M s'arrête avec seulement $f(w)$ écrit sur sa bande.
- si $f(w)$ est indéfini, alors M retourne $\perp$ ou ne s'arrête pas.

Remarque 27:

Les fonctions calculables sont également appelées fonctions (totalement ou partiellement) récursives.

4.2 Exemples de fonctions calculables

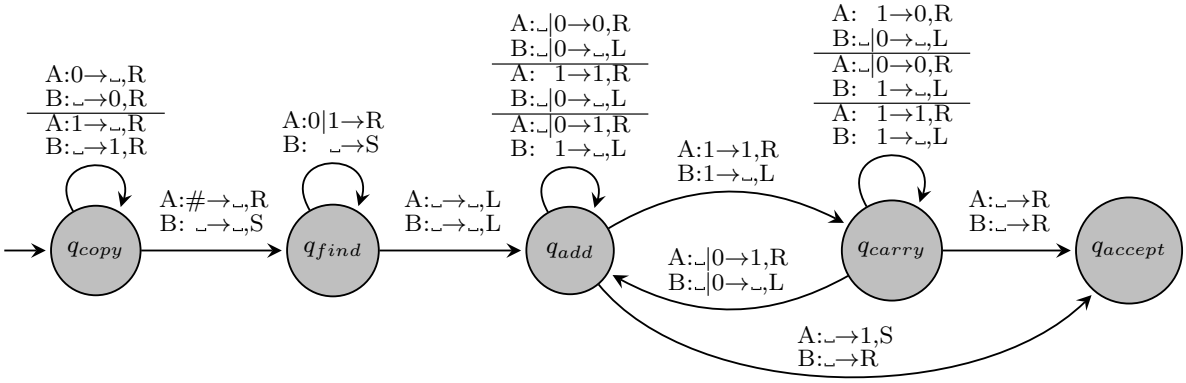
- Toutes les fonctions arithmétiques classiques sur les entiers sont totalement calculables : addition, soustraction, multiplication, division (quotient, reste), ...
- Toutes les fonctions non-arithmétiques sont également totalement calculables : trigonométrique, logarithmique, ... en utilisant les séries de Taylor (ou d'autres techniques d'approximation) à une précision spécifiée.

  voir "Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables" d'Abramowitz et Stegun sur la façon pratique de réaliser ces approximations à une précision donnée.

Nous donnons ici l'implémentation de l'addition binaire sur une MT à 2 bandes (notée A et B), infinie des deux côtés, initialisées avec des blancs, et avec un pointeur LSR (S =stay (ne pas bouger)).

L'entrée de la MT est la forme 1011#110 où # est le séparateur entre les deux mots binaires que l'on souhaite additionner.

- q_{copy} : copie le 1$^{\text{er}}$ mot binaire sur la deuxième bande (état de départ).
- q_{find} : place le pointeur sur le LSB du 2$^{\text{éme}}$ mot binaire de la 1$^{\text{ère}}$ bande.
- q_{add} : somme de 2 chiffres binaires sans retenue pour 0+0, 0+1, 1+0, avec retenue pour 1+1.
- q_{carry} : somme de 2 chiffres binaires avec retenue r=1 (1+0+r, 0+1+r, 1+1+r), sans retenue pour 0+0+r.
- q_{end} : fin du calcul.



Exercice 34. *Écrire la fonction calculable qui :*

1. *calcule un xor entre deux mots binaires. Supposera que l'entrée de la fonction est la suivante $f(\langle w_1\#w_2\rangle)$ où w_1 et w_2 sont deux mots binaires. La fonction pourra être multibande.*
2. *incrémente de 1 le nombre binaire en entrée.*
3. *calcule, en binaire, le nombre de* 1 *présent dans une chaîne binaire.*
4. *effectue la multiplication de deux nombres unaires. Par exemple $f(\langle x, y\rangle) = f(11\#111) =$ 111111.*
5. *effectue $f(\langle n\rangle) = \underbrace{1\ldots1}_{n\ fois}$.*
6. *triplique la chaine d'entrée (i.e. si la chaine d'entrée est $w =$ 11, la fonction retourne $www =$ 111111). On prendra l'alphabet $\Sigma = \{1\}$*
7. *retourne le caractère au centre d'une chaîne de caractères si sa longueur est impaire, et une erreur sinon.*

5 Machines Universelles

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 ?

5.1 Machine de Turing Universelle

Exercice 35. *Soit la description formelle d'ADF $A = \langle(Q, \Sigma, \delta, q_0, F)\rangle$.*

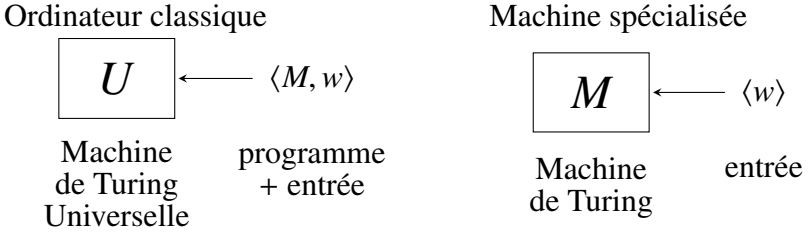
1. *Rappeler le sens de chacun des éléments de la description formelle d'un ADF.*
2. *Donner un codage de A sous forme d'une chaîne de symboles.*
3. *Écrire le code de la MT qui vérifie si une chaine w est bien le codage d'un ADF, à savoir d'une MT M telle que $M(\langle A\rangle)$ accepte si A représente la description formelle d'un ADF, et rejette sinon.*
4. *Écrire le code de la MT qui vérifie si un mot w est accepté par l'ADF A, à savoir écrire une MT M tel que $M(\langle A, w\rangle)$ accepte si A accepte w et rejette sinon.*

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

Exercice 36. *Donner un codage de la description formelle donnée à l'exercice 30.*

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

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



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

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.
- **Exécution 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.

Remarque : la MT universelle U

- a un alphabet de travail beaucoup plus grand
  pour faciliter les comparaisons, les copies, l'effaçage
- 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 IV.4 (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 $C.T \log T$ 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.

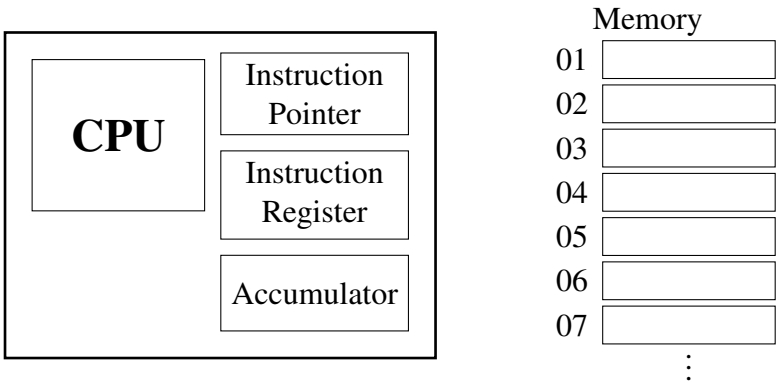
En conséquence,

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

Ces notions d'infini seront précisés plus tard.

5.2 Random Access Machine

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



Notons qu'une RAM est une machine programmable.

Comment définir une machine de Turing permettant de simuler une 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 (mémoire locale)
- d'un CPU qui fonctionne comme suit :
  1. `IR` ← `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
000	HALT	Arrêt
1xx	LOAD xx	A←MEM[xx]
2xx	LOADI xx	A←xx
3xx	STORE xx	MEM[xx]←A
4xx	ADD xx	A←A+MEM[xx]
5xx	ADDI xx	A←A+xx
6xx	SUB xx	A←A-MEM[xx]
7xx	SUBI xx	A←A-xx
8xx	JUMP xx	IP←xx
9xx	JZERO xx	IP←xx if A = 0

Exemple de programme : multiplication $n_r = n_1 \times n_2$

organisation mémoire :

adresses	50	51	52	53
contenu	n_1	n_2	n_r	compteur

Memory	Code	Assembleur
01	150	LOAD 50
02	353	STORE 53
03	200	LOADI 0
04	352	STORE 52
05	253	LOAD 53
06	912	JZERO 12
07	701	SUBI 1
08	353	STORE 53
09	152	LOAD 52
10	451	ADD 51
11	804	JUMP 04
12	000	HALT

Algorithme :

```
START  A←MEM[50]
       MEM[53]←A
       A←0
LOOP   MEM[52]←A
       A←MEM[53]
       IF A==0 GOTO END
       A←A-1
       MEM[53]←A
       A←MEM[52]
       A←A+MEM[51]
       JUMP LOOP
END    HALT
```

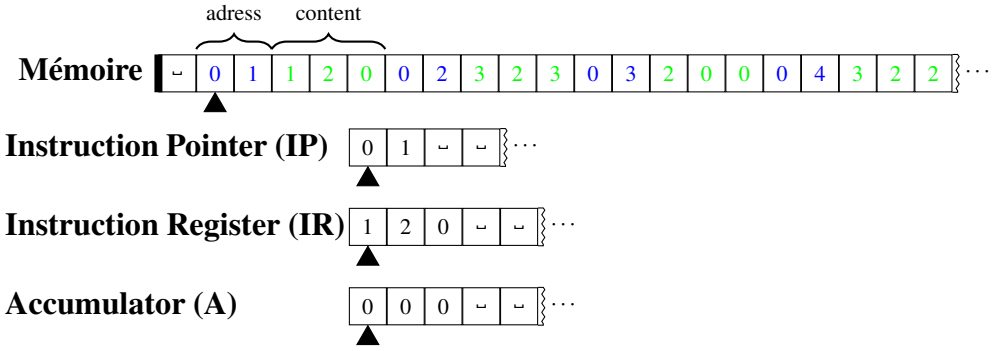
Théorème IV.5 (simulation d'une RAM)

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

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

Organisation des bandes :

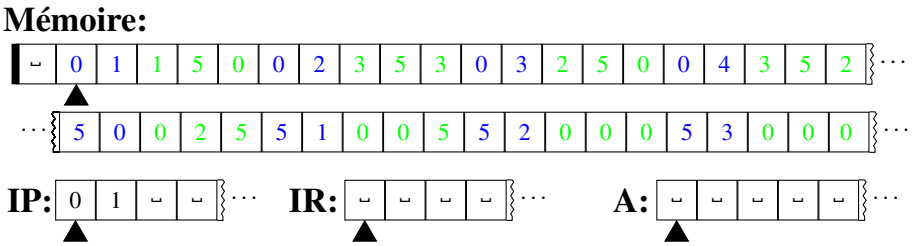
- utiliser une bande par registre (IR, IP et A).
- utiliser une bande pour la mémoire
- organisation de la mémoire :
  avec des entrées : <adresse> <contenu>
  utilisée pour stocker **le code** et **les données**



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. □

Des graphes simplifiés de la machine de Turing simulant une RAM seront donnés au cours de la simulation avec les notations suivantes :

- $* \to R$ signifie que la transition a lieu pour tout symbole sous le curseur.
- $B : 0 \to R$ signifie que la transition concerne la bande B.
- normalement, pour une machine multibande, toute transition entre deux états doit être spécifiée pour toutes les bandes.
- toute bande B non spécifiée dans une transition ne sera pas modifiée par cette dernière. Donc équivalent à $B : * \to S$ = le curseur ne bouge pas pour tout symbole sous le curseur pour cette bande).
- si une même variable x (i.e. x est un symbole quelconque) est utilisé sur plusieurs bandes, alors ce symbole doit être identique pour les deux bandes.

M:x→R IP:x→R	transition à droite sur les bandes M et IP si elles sont sur le même symbole x
M:x→ R IP:*→x,R	transition à droite sur les bandes M et IP, le symbole sur la bande M est écrit à la place du symbole courant sur la bande IP.

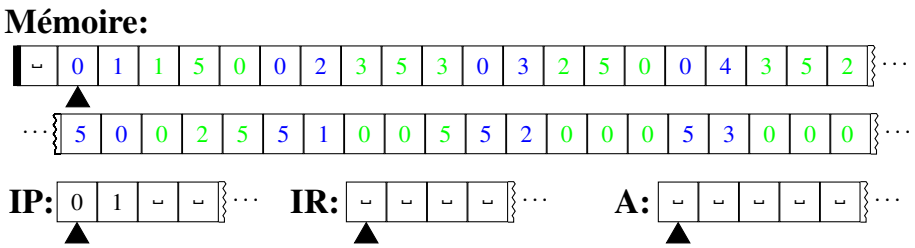
- si deux variable x et y sont utilisés sur plusieurs bandes, alors elles représentent des symboles différents.

M:x→R IP:y→R	transition à droite sur les bandes M et IP si elles sont sur des symboles différents

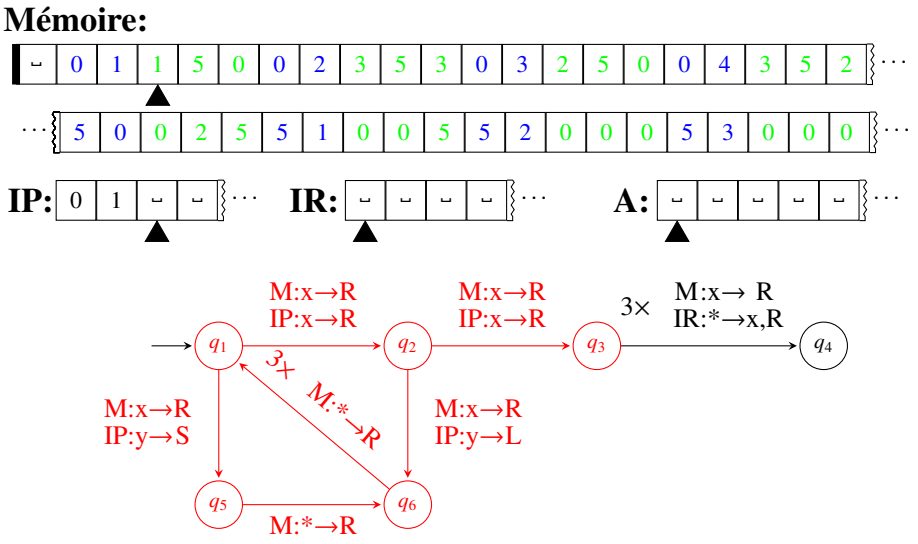
- l'écriture $2\times$ M:x→R signifie que la même transition est effectuée deux fois.

Exécution ! (multiplication)

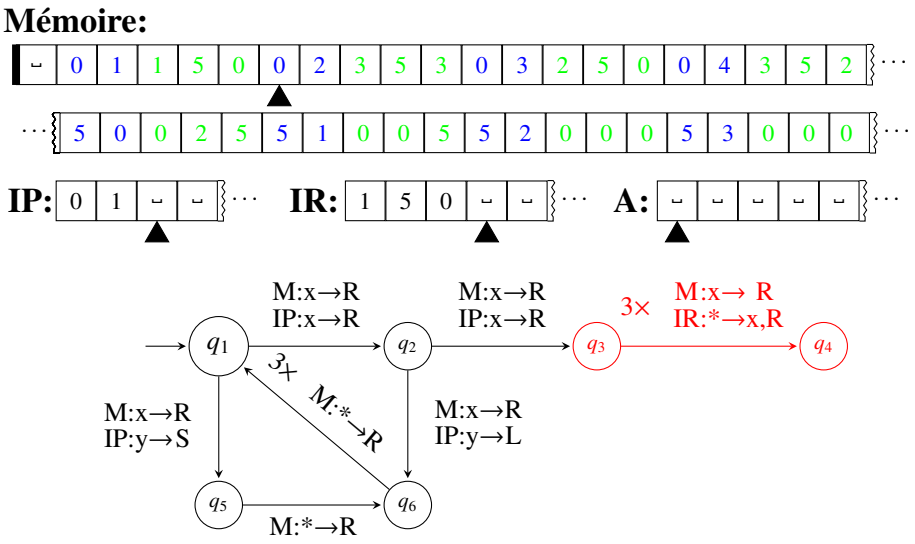
• Initialisation



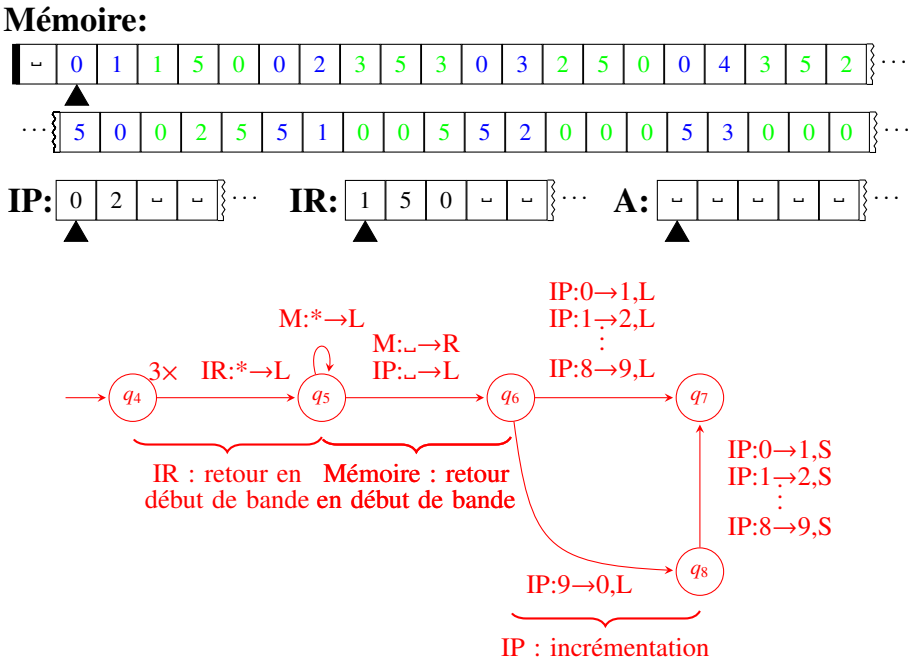
• Recherche de l'IP dans la mémoire (1ère instruction)



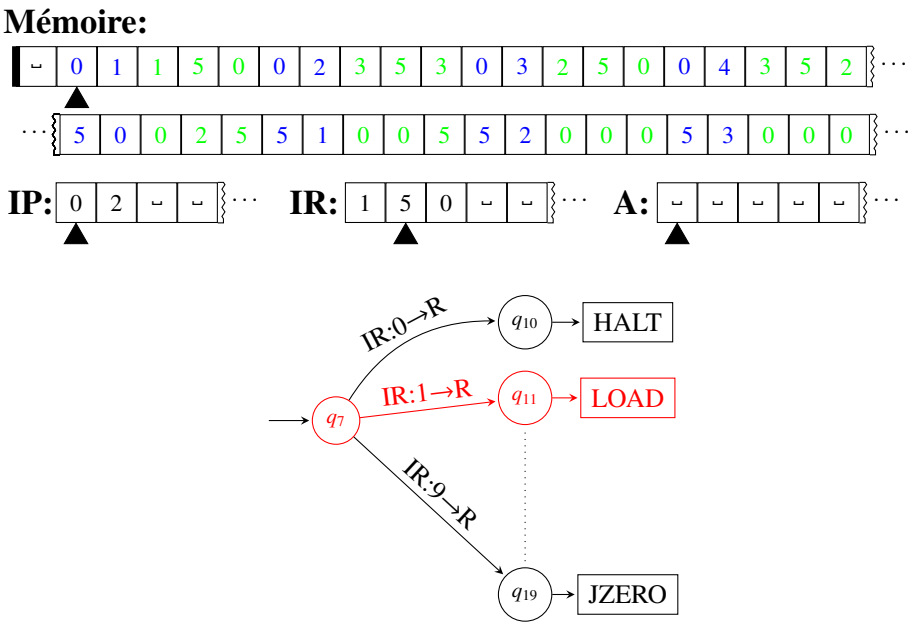
• Chargement du contenu à l'adresse de IP dans l'IR



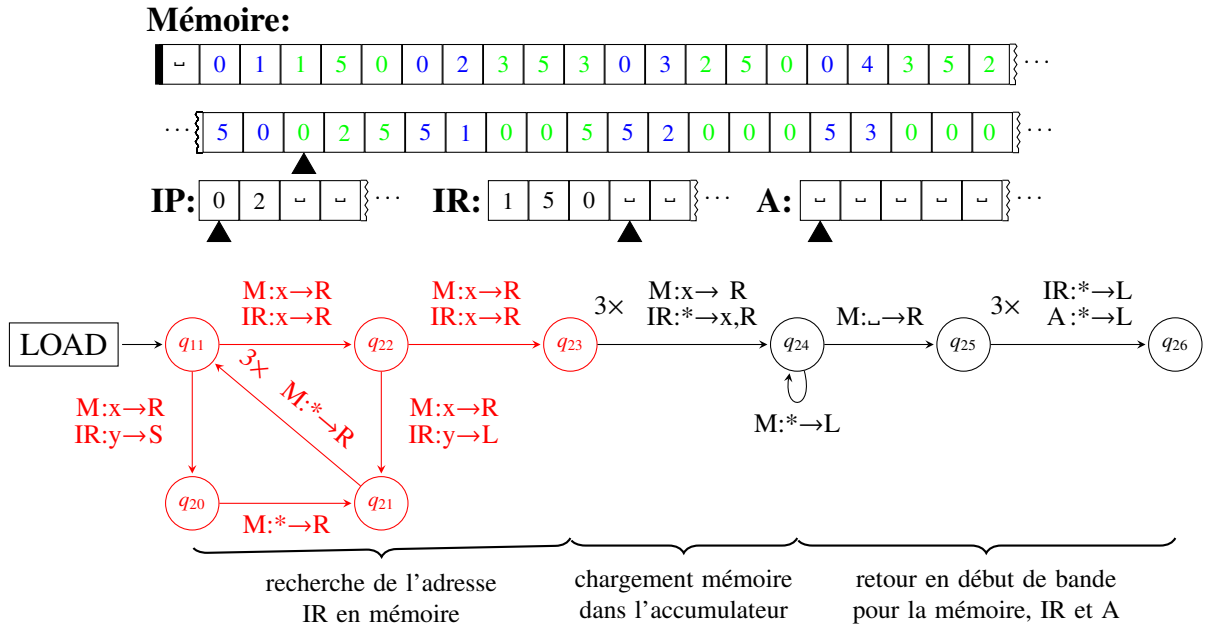
• Réinitialisation position des curseurs et incrémentation de l'IP.



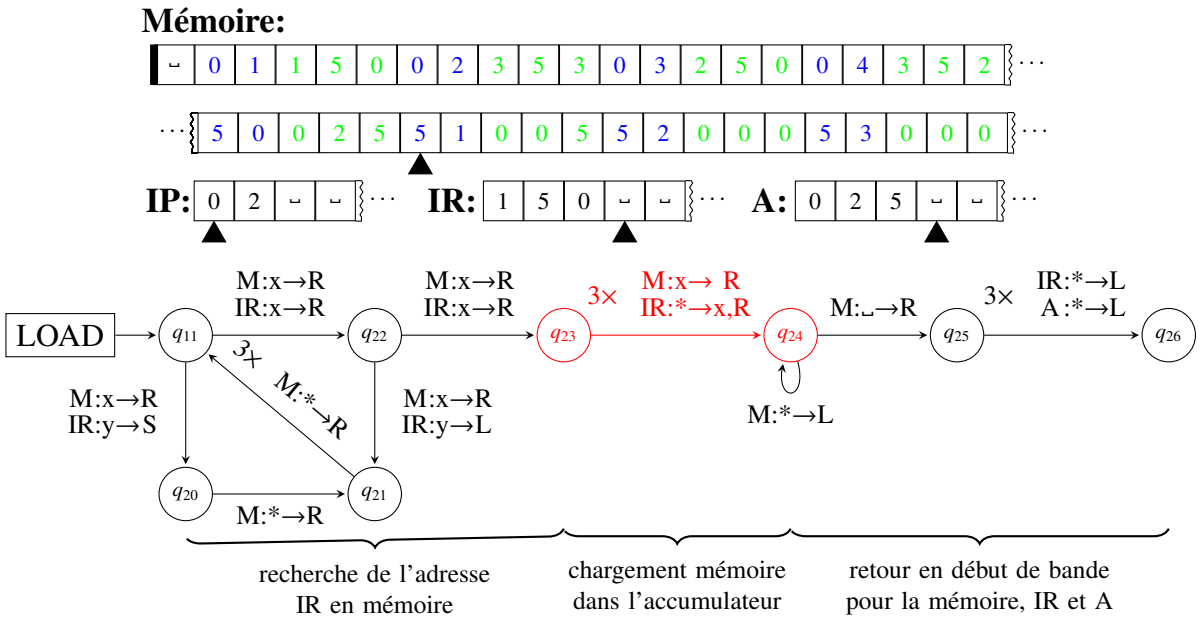
• Décodage de l'instruction à exécuter
code 1 dans la cellule 0 de l'IR = `LOAD`



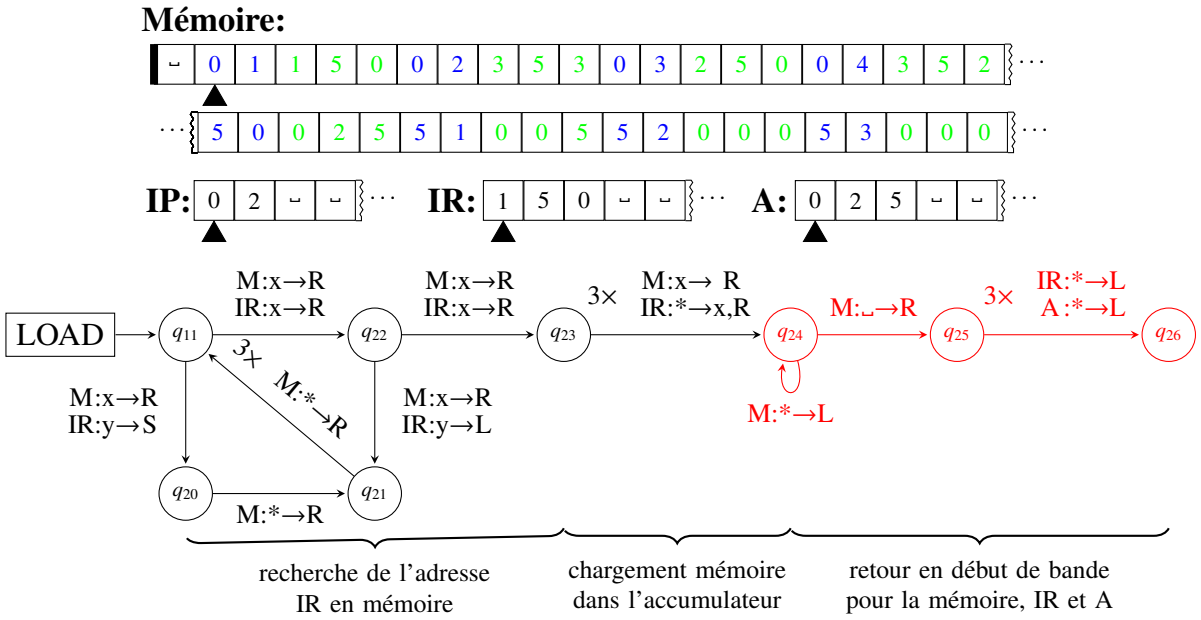
• Exécution de l'instruction décodée (`LOAD`)
recherche de l'adresse (cellule 1 et 2 = 50) dans IR = `LOAD`.



• Exécution de l'instruction décodée (ici `LOAD`)



• Exécution de l'instruction décodée (ici `LOAD`)



et on continue ainsi jusqu'à ce que la machine s'arrête.

6 Résumé

Nous avons vu dans ce cours que :

- la machine de Turing est modèle très puissant pour représenter les algorithmes,
- les langages récursivement énumérables sont les langages pouvant être reconnu par une MT ; les mots hors langages peuvent faire boucler la machine,
- toutes les variations des machines de Turing sont équivalentes.