U.F.R SCIENCES ET TECHNIQUES
Département d'Informatique
B.P. 1155
64013 PAU CEDEX
Téléphone secrétariat : 05.59.40.79.64
Télécopie : 05.59.40.76.54

I-Calculabilité et limitations des algorithmes: Machine de Turing Partie I

I-Introduction
II-La « thèse de Church-Turing »
III-La machine de Turing
IV-Calculabilité et Décidabilité

I-Introduction

La **théorie de la calculabilité** permet au développeur de comprendre :

- la **puissance** des programmes informatiques
- et leurs **limitations**.

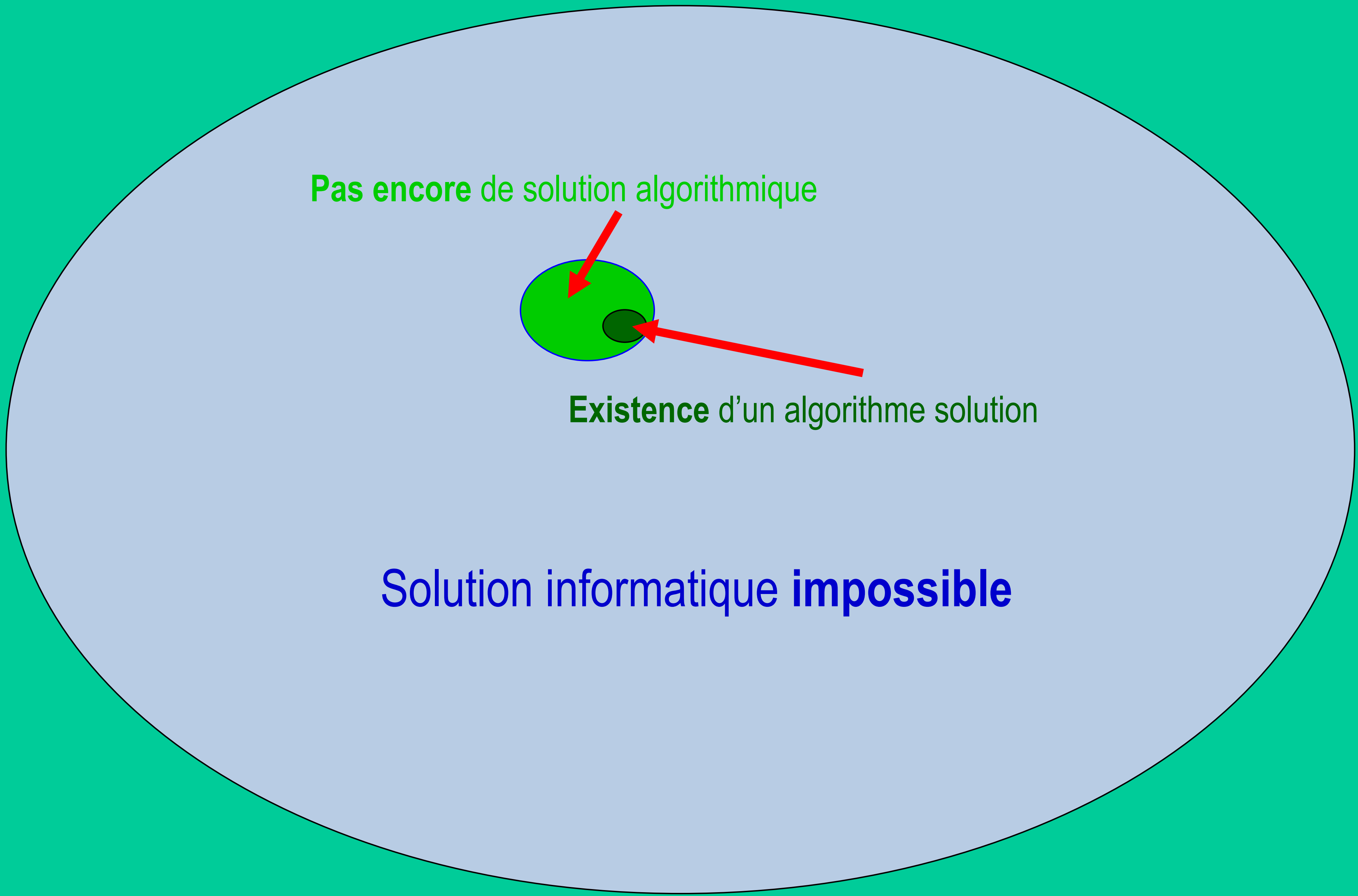
On y **prouve** (preuve) que certains problèmes:

- **peuvent se résoudre** informatiquement
- et que d'autres **ne peuvent pas l'être**.

L'objectif est d'**explorer les limites** de la programmation informatique.

En **pratique**, on peut se dire qu'en informatique, on s'intéresse à résoudre des **problèmes**, en les programmant à l'aide d'**algorithmes**.

Donc, on peut penser que s'intéresser aux problèmes que l'on ne **sait pas programmer** présente peu d'intérêt.


Pas encore de solution algorithmique
Existence d'un algorithme solution
Solution informatique **impossible**

Au sens de la **théorie de calculabilité**, c'est tout le contraire !

Les problèmes auxquels nous nous intéressons ne sont:

- ni des problèmes pour lesquels on ne connaît **pas encore** de solution.
- ni, a fortiori, les problèmes pour lesquels on **a proposé** un algorithme pour les résoudre.

Mais, et c’est encore beaucoup plus fort !

On s’intéresse aux problèmes pour lesquels :

-on sait qu’il est **impossible**

-de produire une **solution algorithmique**.

<u>Question</u>: pourquoi s’intéresser aux problèmes qui **ne peuvent pas** être résolus ?

1-**<u>Première réponse</u>**:

Comprendre qu'un problème ne peut pas être résolu est utile.

Cela signifie que le problème :

- doit être **simplifié**
- ou **modifié** (par réduction)

pour pouvoir être résolu.

2-**Deuxième réponse**:

Les résultats obtenus seront exploités pour **mettre en perspective**:

-les **limitations** des algorithmes,

-la **calculabilité** de certains problèmes **critiques**,

-l'**automatisation** de certaines tâches, comme par exemple en robotique.

Concept d’algorithme

Jusque-là, on a utilisé le concept d’**algorithme**, sans en avoir donné une **définition formelle**.

Intuitivement, on dit qu’un **algorithme** est « une suite finie d’étapes permettant d'obtenir un résultat »

En informatique, on peut dire qu'un **algorithme** est « une méthode **automatique:**

-pour **résoudre** un problème donné,

-qui peut être implémenté par un **programme. »**

Dans ce qui suit, nous allons définir **formellement** ce que nous entendons par **algorithme**.

I-La « thèse de Church-Turing »

Certains **mathématiciens** ont donné leur propre définition au concept d'**algorithme**.

Question : toutes ces définitions sont-elles **équivalentes** ?

1−Définition de Gödel

Un algorithme est une **suite de règles** mathématiques permettant:

- de construire des **fonctions complexes**
- à partir de **fonctions plus simples**,

2−Définition de Church

Un algorithme est un **ensemble de fonctions** exprimées dans un formalisme appelé **lambda-calcul**.

Le **lambda-calcul** de Church consiste à représenter les fonctions de la manière suivante:

$$\lambda \langle \text{ variables liées} \rangle \bullet \langle \text{corps} \rangle$$

En informatique:

-les **variables liées** correspondent aux **paramètres**,

-et le **corps** décrit la **procédure**: ce que fonction fait avec ces paramètres.

Exemple de fonctions λ

– Fonction **identité** :

id = λ x • x   (λ x . x est la fonction qui à x associe x)

– Fonction **Maximum** de deux nombres :

max = λ a b • **si** a > b **alors** a **sinon** b.

– Fonction **récursive de Fibonnaci** :

f = λ n • **si** n < 3 **alors** 1 **sinon** f(n-1) + f(n-2).

– Fonction **pgcd**:

pgcd= λ xy • **si** y =0 **alors** x **sinon** pgcd(y , x%y)

– Fonction **Tri par fusion**:

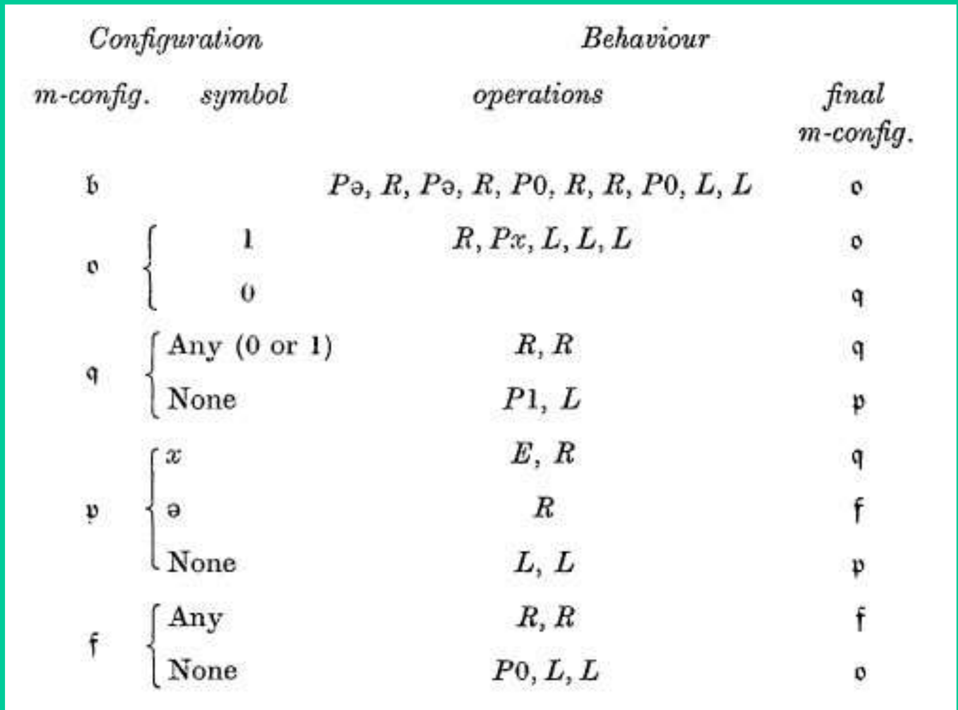
tri = λ t[i :j] • **si** i≥j **alors** t[i : j]

sinon inter(tri(t[i : (i+j)/2]),tri(t[(i+j)/2+1: j]))

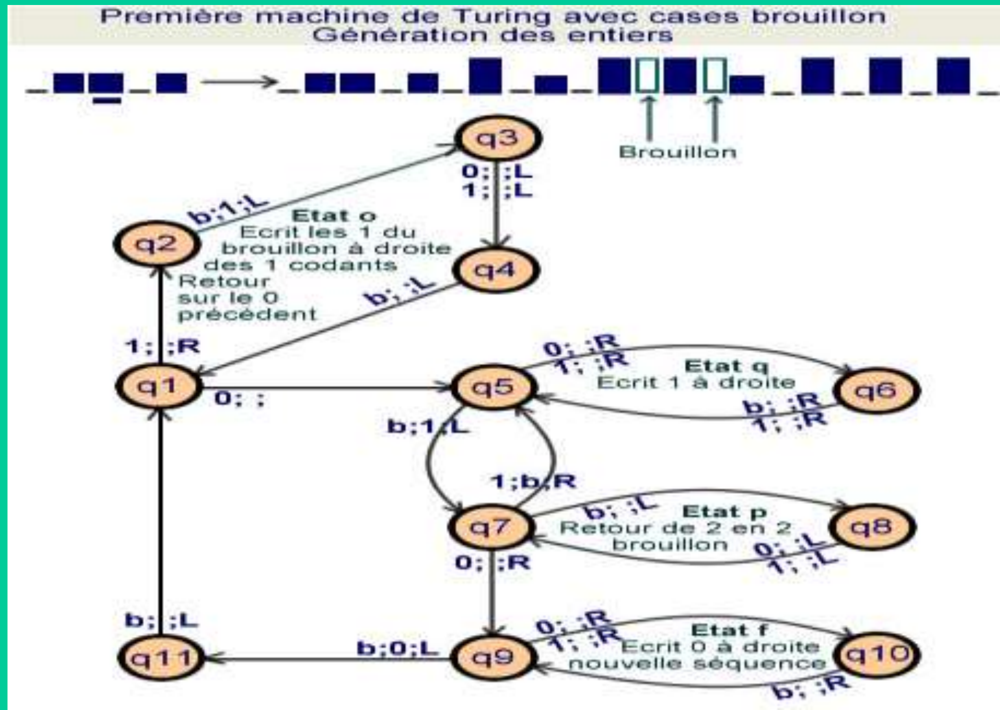
3−Définition de Turing

Un algorithme est un **ensemble d'instructions** pour une machine hypothétique: la **Machine de Turing**

La machine de Turing ou l'idée d'ordinateur



Configuration		Behaviour	
m-config.	*symbol*	*operations*	*final m-config.*
$\mathfrak{b}$		$P\text{ə}, R, P\text{ə}, R, P0, R, R, P0, L, L$	$\mathfrak{o}$
$\mathfrak{o}$	1	R, Px, L, L, L	$\mathfrak{o}$
	0		$\mathfrak{q}$
$\mathfrak{q}$	Any (0 or 1)	R, R	$\mathfrak{q}$
	None	$P1, L$	$\mathfrak{p}$
$\mathfrak{p}$	x	E, R	$\mathfrak{q}$
	ə	R	$\mathfrak{f}$
	None	L, L	$\mathfrak{p}$
$\mathfrak{f}$	Any	R, R	$\mathfrak{f}$
	None	$P0, L, L$	$\mathfrak{o}$



Publication d'Alan Turing en 1936

«Enoncé de la thèse de Church-Turing »

Les **deux propositions** suivantes furent alors admises :

1-toutes les définitions du terme algorithme **connues à ce jour** sont **équivalentes**,

2-toute définition du terme algorithme qui **sera établie par la suite** sera équivalente aux définitions existantes.

Toutes ces **définitions**, en apparence dissemblables, sont **équivalentes** signifie que :

-si on peut faire un calcul à partir d'un algorithme défini dans l'**une** de ces approches,

-alors on peut aussi le faire à partir **des autres**.

Ces idées sont désignées aujourd’hui par l’expression de **« thèse de Church-Turing »**.

Jusqu’à ce jour aucun fait n’est venu démontrer le contraire.

Cette thèse est désormais **universellement** reconnue.

En résumé

La «thèse de Church-Turing» affirme que nous disposons d'une bonne définition pour le concept d'**algorithme**.

Cette définition ne tient pas compte du caractère particulier :

- de tel ou tel **ordinateur**,
- ou de tel ou tel **langage** de programmation.

Tout **algorithme** qui a été exécuté sur un ordinateur **particulier** peut être exécuté sur n’importe quel **autre**.

L’**équivalence** entre tous les **ordinateurs** modernes est une **conséquence** de la confirmation de la «thèse de Church-Turing».

II-La machine de Turing

Est défini comme étant **calculable** tout **problème** qui peut être résolu par :

- un **algorithme**
- exécutable par la **Machine de Turing**

L’idée originale de Turing

L’idée de Turing en **1936**, a été de formaliser la **notion de calculabilité**:

- par une **machine théorique**
- qui incarne le concept d’**algorithme**

et ce avant l’existence même des **ordinateurs**.

Cette notion de calculabilité fait **abstraction des limitations** physiques que comportent nécessairement les machines.

Peu importe :

-le **temps** de calcul.

-et /ou la **taille mémoire**

que certains calculs peuvent exiger.

Indépendamment de ces **contraintes matérielles**, on pourra démontrer qu'un tel calcul permet:

- d'aboutir effectivement à un **résultat**
- après un **nombre fini d'étapes**

1-Motivation

L’objectif de Turing était de **formaliser** la notion de **calculabilité** moyennant une machine.

Un moyen classique pour l’atteindre aurait pu être l’utilisation d’une **machine à calculer**.

Pourquoi ?

La machine à calculer se limite à réaliser des opérations: processus purement **calculatoire**

Alors que **Turing** était plus ambitieux: il cherche à exprimer l'exécution d'un **algorithmique** sous une **forme mécanique**.

Toutefois, il fallait que cette machine soit **aussi simple que possible** afin de **simplifier** les démonstrations.

Le calculateur **idéalisé** qu'il a considéré à cette fin diffère profondément, par son agencement, des **calculateurs réels**.

En effet, la machine de Turing :

- a une mémoire **potentiellement infinie**.
- ne lit qu'**un symbole** à la fois

Cependant, si l'on reste sur le plan des concepts, la machine de Turing est beaucoup **moins irréaliste** qu'il ne paraît.

On peut avancer 3 raisons à cela :

1- Elle a une mémoire **potentiellement infinie**.

On retrouve une situation pratiquement analogue car on dispose sur un ordinateur de **disques** interchangeables **en quantité suffisante**, 2-Elle ne lit qu'**un symbole** à la fois

Rien n'empêche de donner à ce symbole peut avoir une **signification complexe**.

3-Elle a un nombre fini d'états

C'est le cas également de l'unité centrale d'un **calculateur réel** qui n'a qu'un **nombre fini d'états** internes.

Son comportement à une étape donnée dépend :

-de l'**état** où elle se trouve

-et de ce qu'elle **reçoit comme information**.

A la convention près :

-que l'apport d'information est idéalisé en la lecture d'un **symbole atomique**,

-le comportement est celui d'un ordinateur moderne.

A retenir absolument !

Les machines de Turing constituent une notion centrale en informatique, car:

1-<u>Puissance du formalisme</u>

-elles offrent une base théorique solide aux notions de **décidabilité** et de **complexité**

-elles offrent une **sémantique formelle** à la notion d'**algorithme**

2-<u>Puissance de calcul</u>

Tout problème **calculable** sur un ordinateur, aussi puissant soit-il, est calculable sur une machine de Turing.

Tout problème non calculable sur une machine de Turing est non calculable sur n'importe quel ordinateur aussi puissant soit-il

3-Puissance de calcul

La machine de Turing est conçue de façon **très simple** dans le seul but de **simplifier** les démonstrations et les calculs de preuves.

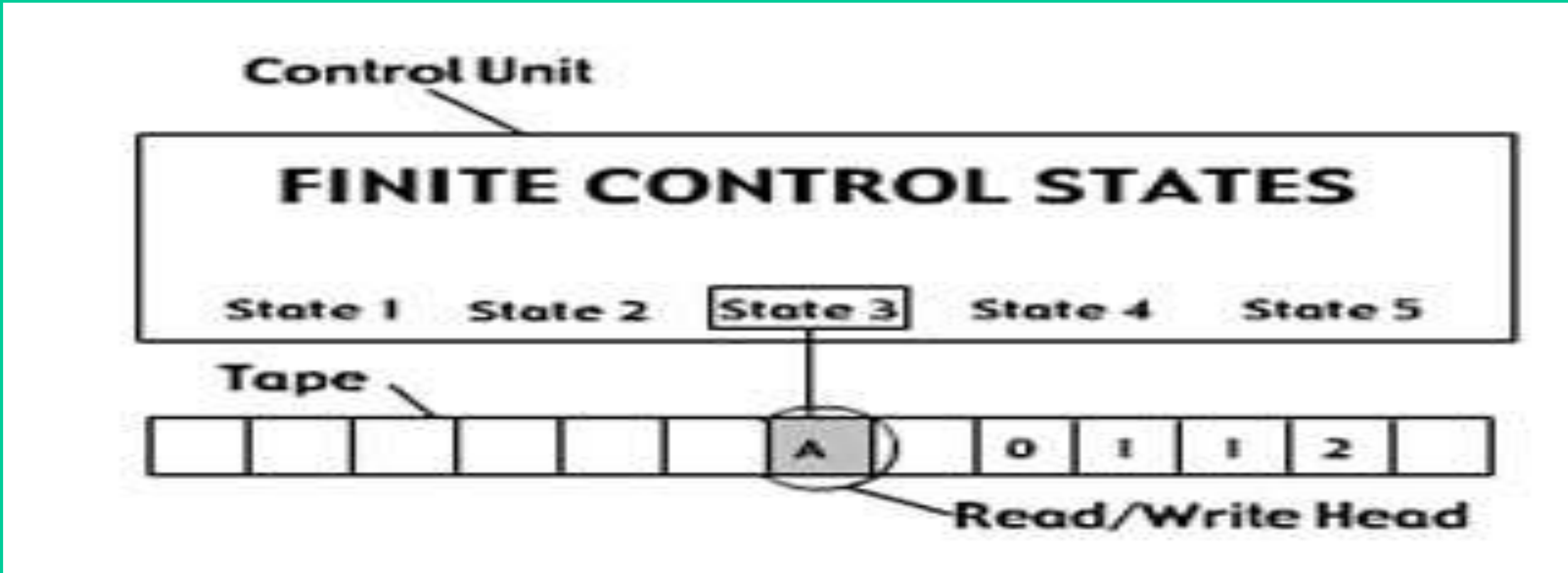
Ces démonstrations et ces calculs de preuves seront **très difficiles** à établir sur une machine à peine plus compliquée que la machine de Turing.

2-Description intuitive

Une **machine de Turing** se compose de quatre éléments:

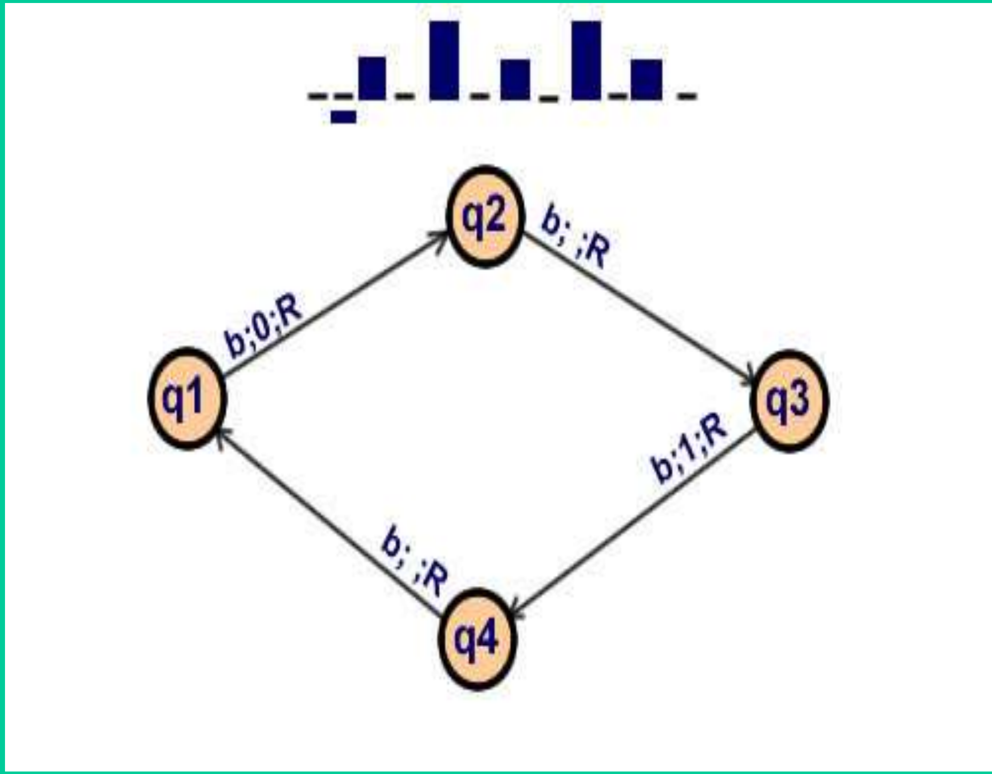
- -une **unité centrale**,
- -une **mémoire infinie** sous la forme d’une bande (ruban),
- -une **tête de lecture/écriture**,
- -un **programme**.

Schéma originale d’une MT

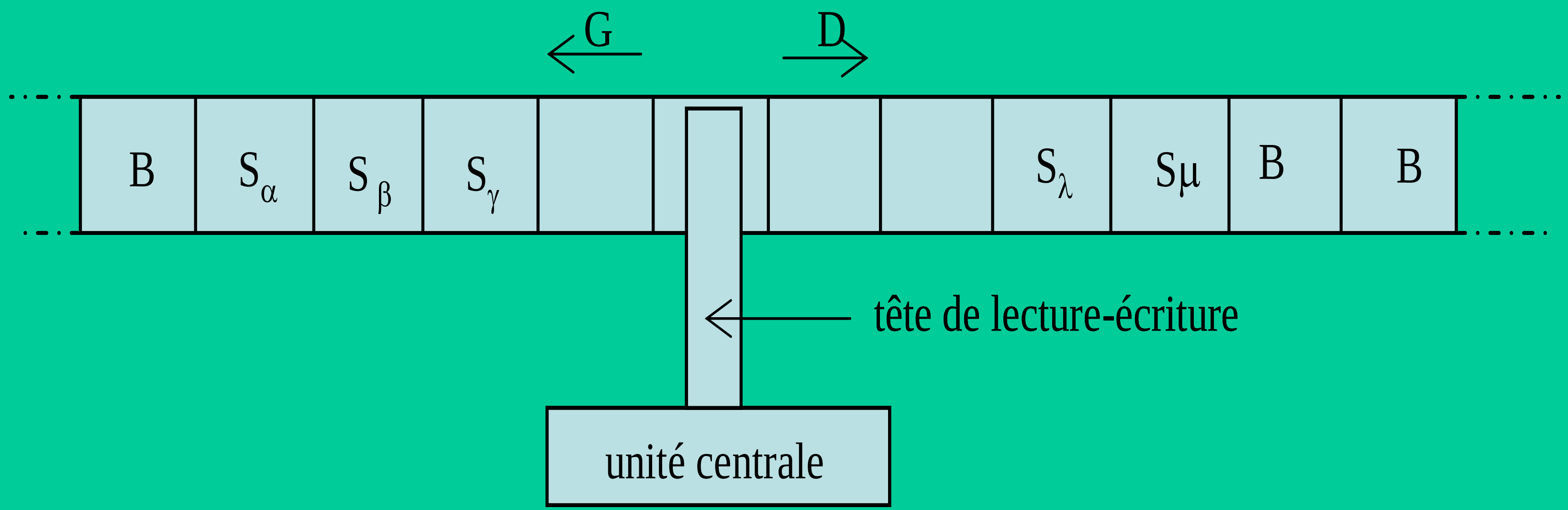


Voici le diagramme de la première machine décrite par Turing en **1936** pour construire la séquence "**0 1 0 1 0 1 0 1 0 1...**"

Table des transitions				
Etat	Lecture	Ecriture	Déplacement	Nouvel état
1	b	0	R	2
	0			
	1			
2	b		R	3
	0			
	1			
3	b	1	R	4
	0			
	1			
4	b		R	1
	0			
	1			

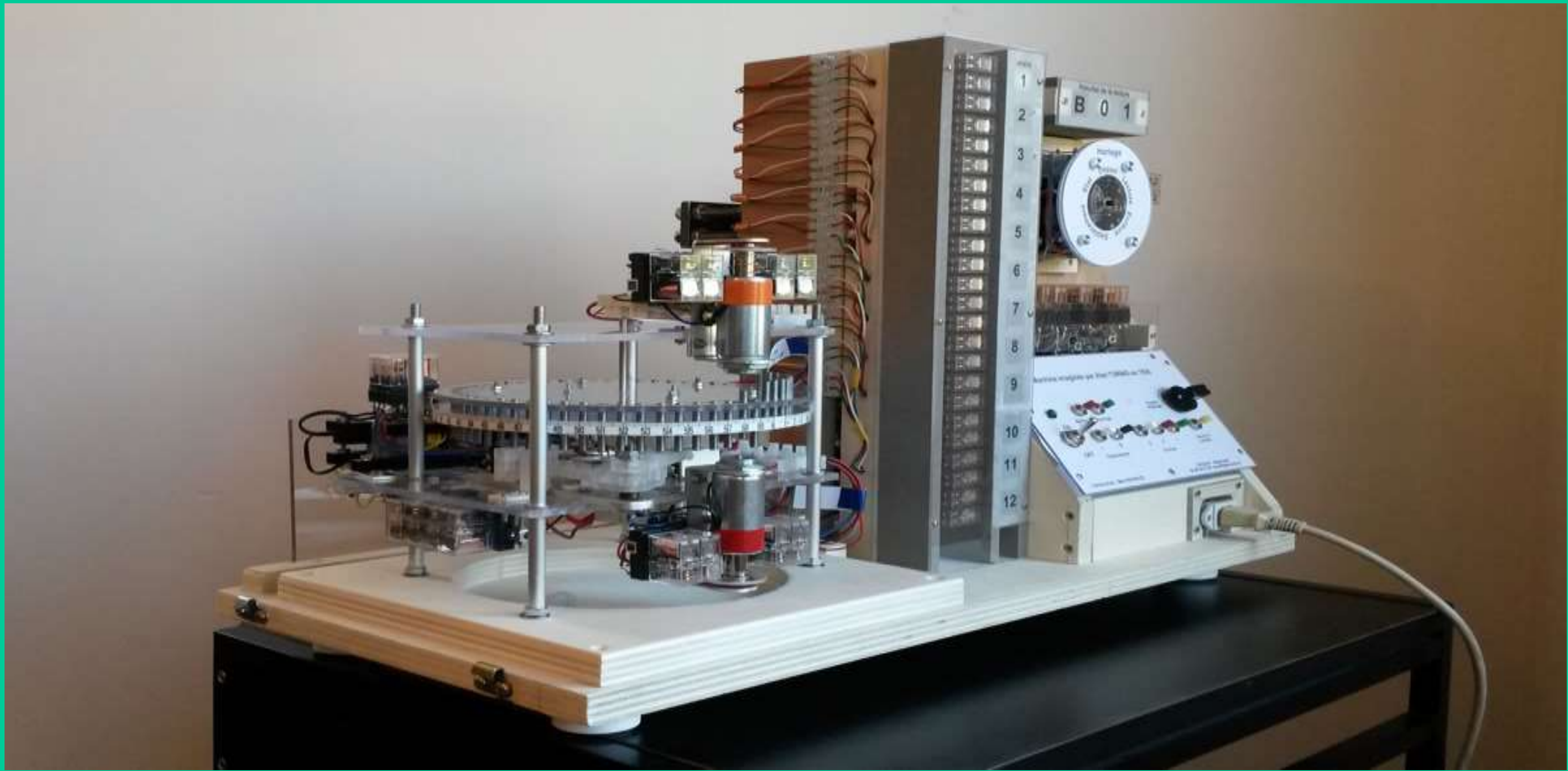


La structure d'une MT peut être schématisée comme suit:



Prototype de Machine de Turing

Professeur O. Raynaud -Université de Clermont Ferrand



1−Unité centrale

L'**unité centrale** est caractérisée par un ensemble **Q** fini d'**états internes.**

$$Q = \{q_0, q_1, \ldots, q_n\}$$

Ces états correspondent aux configurations prises par un ordinateur actuel: mémoires internes, circuits, etc…

2−Mémoire infinie

La **mémoire** est une bande (ou ruban) sur laquelle :

- -sont inscrites, au départ, les **données à traiter**
- -viennent s'inscrire, en cours d'exécution, les **calculs effectués**.

Les écritures se font au moyen des symboles d'un **alphabet** Σ fini.

La bande est divisée en **cases**: dans chaque case ne figure qu'un **seul symbole**.

…	B	B	a	b	a	a	b	B	B	B	B	…

On suppose que la machine dispose d'une bande **illimitée.**

Ainsi, **théoriquement**, on peut y inscrire **tous** les symboles nécessaires aux calculs.

3−Tête de lecture-écriture

Elle assure la **communication** entre :

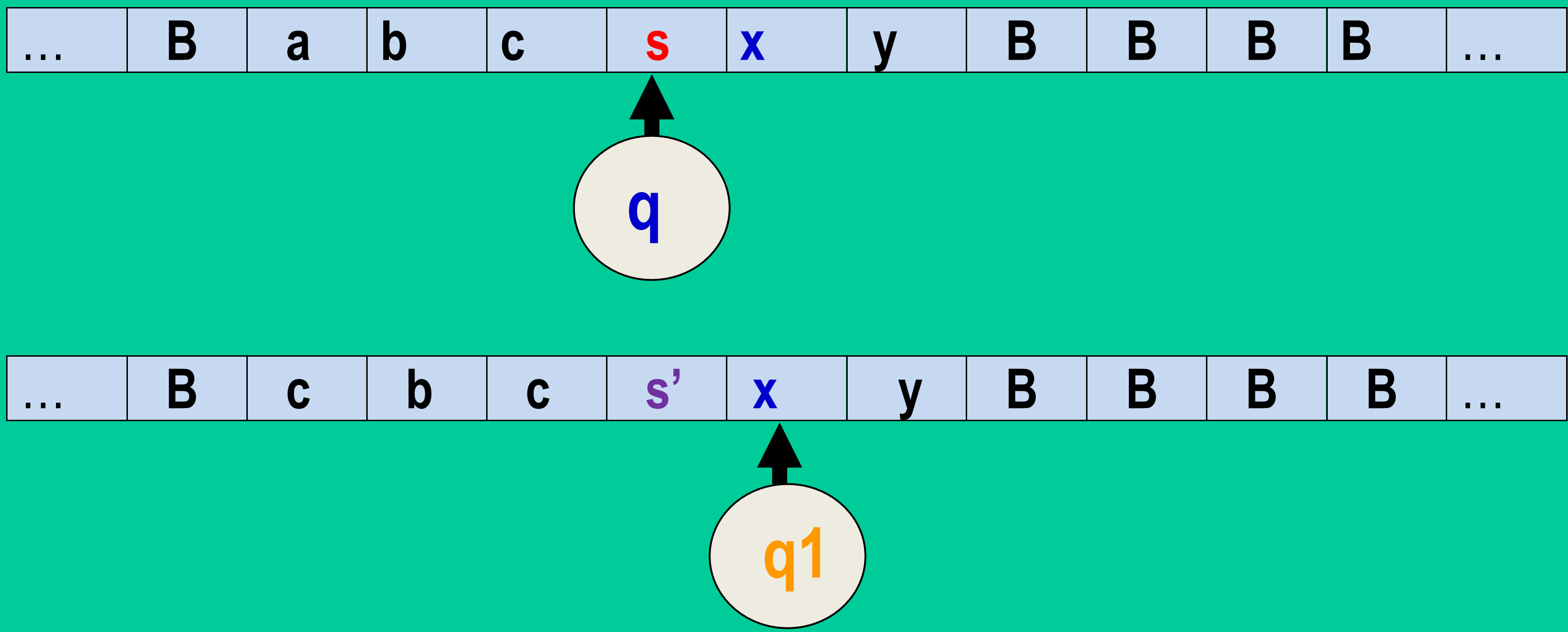
- l'**unité centrale**
- et la **bande**.

La tête n'opère que sur une **seule case à la fois**.

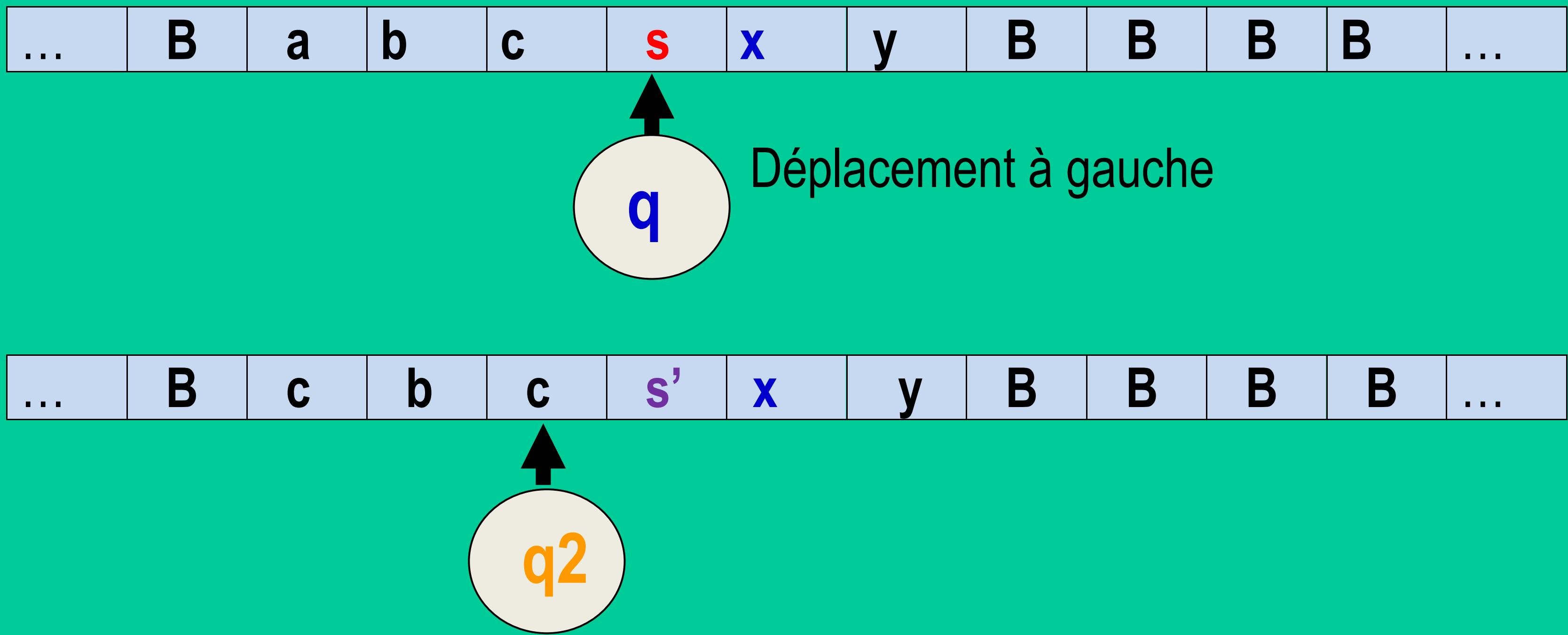
Elle peut:

- **lire** le symbole dans la case au-dessous : s
- **remplacer** le symbole **lu** par un **nouveau** symbole **s’**
- **se déplacer** éventuellement à **gauche** ou à **droite** de la bande.

Exemple de déplacement à **droite**



Exemple de déplacement à **gauche**

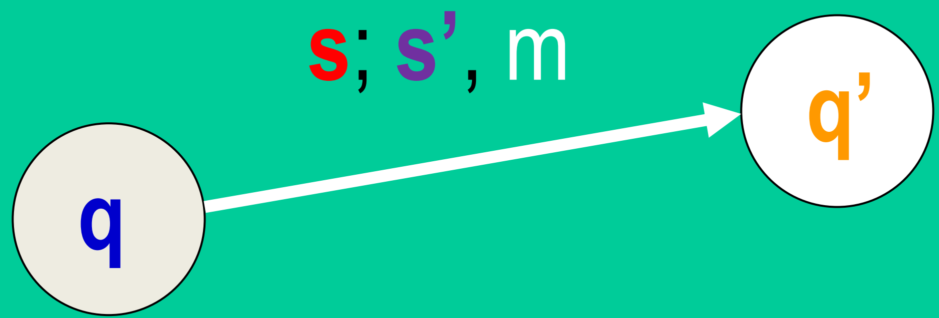


4-Un programme

Le programme est formalisé par une **fonction de transition**:

$$\delta\,(q,s) = (q', s', m)$$

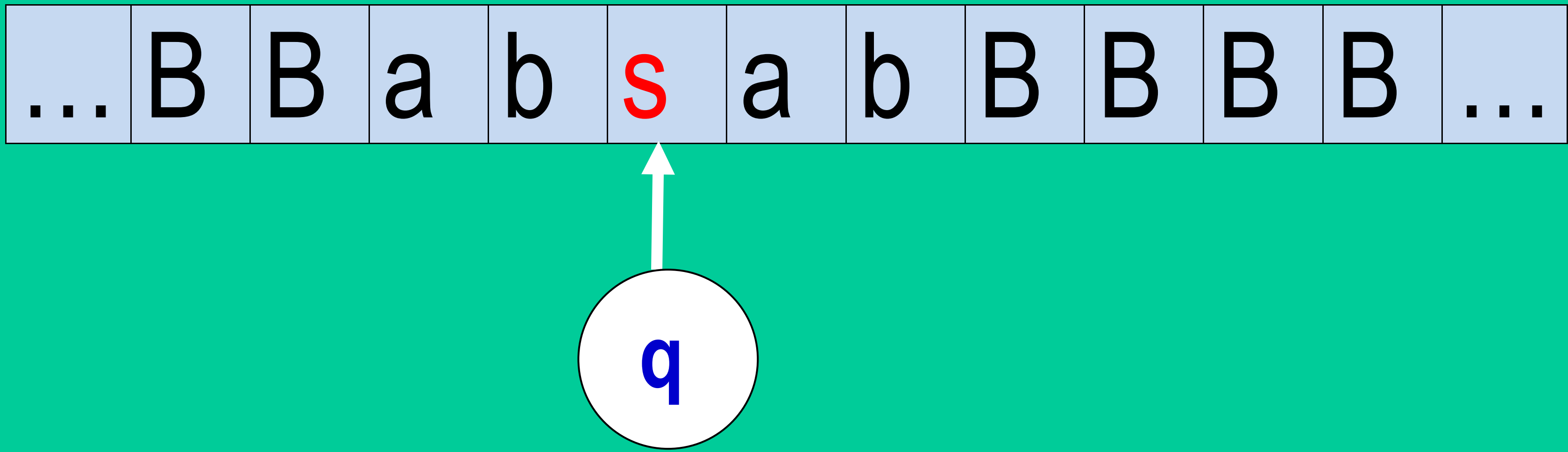
Symbolisée graphiquement par:



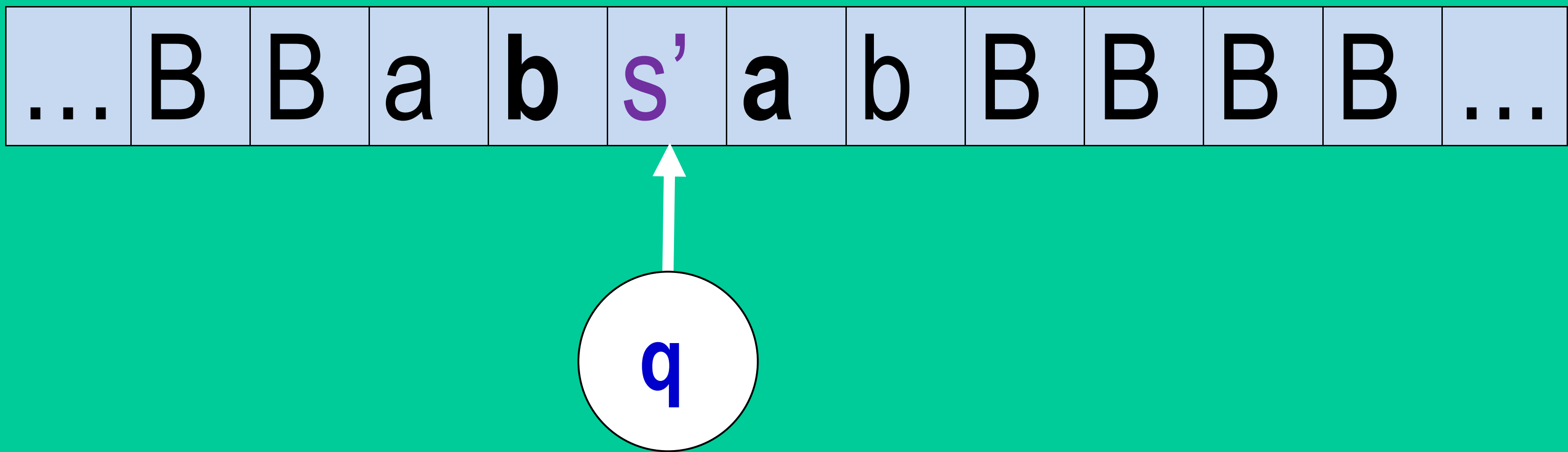
Comment exécuter une transition ?

Pour chaque état **q**, la fonction δ précise selon le symbole **s** sous la tête de lecture:

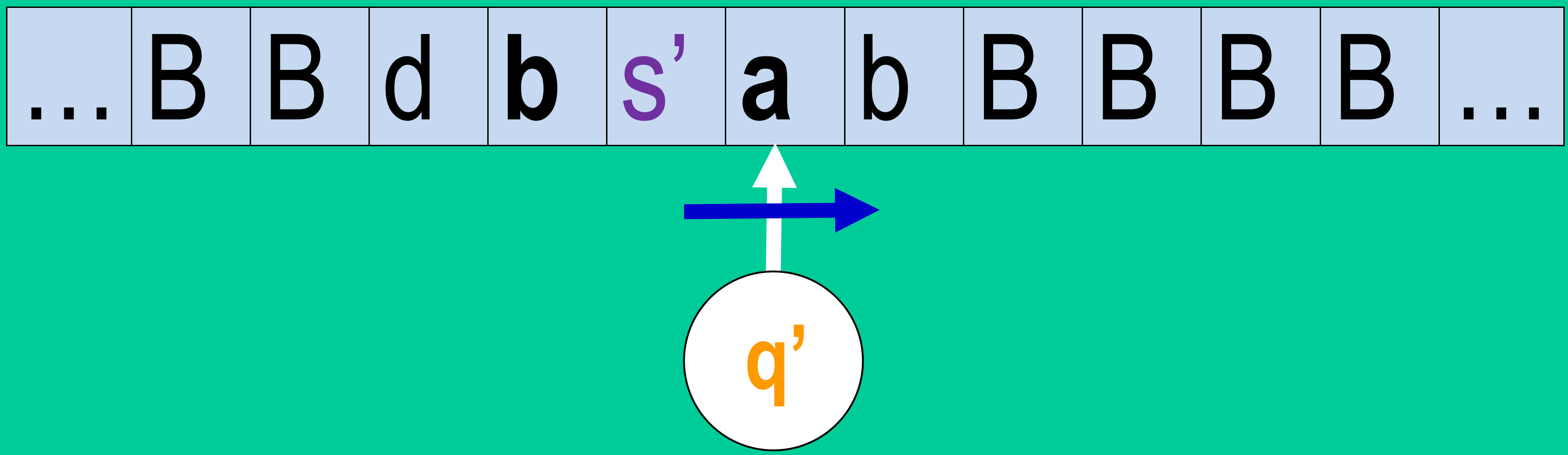
1- le **nouvel élément s'** $\in \Sigma$ à écrire à la place de **s**,
2- un **sens de déplacement** m pour la tête de lecture,
3- ensuite le **nouvel état q'** $\in Q$.



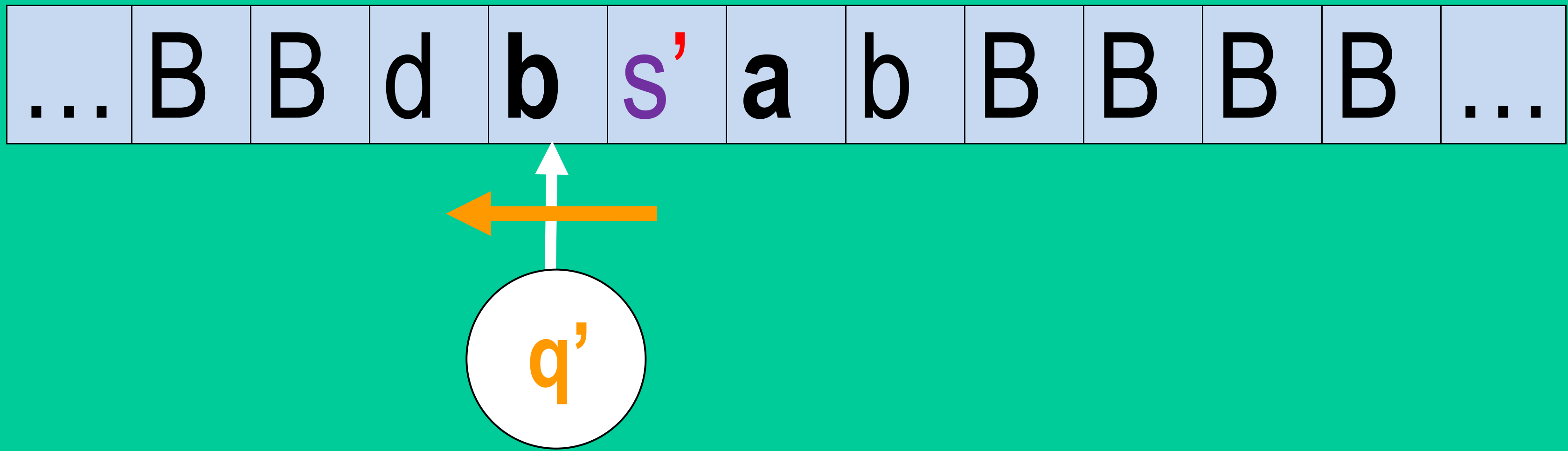
$$\delta\,(q,s) = (q', s', m)$$



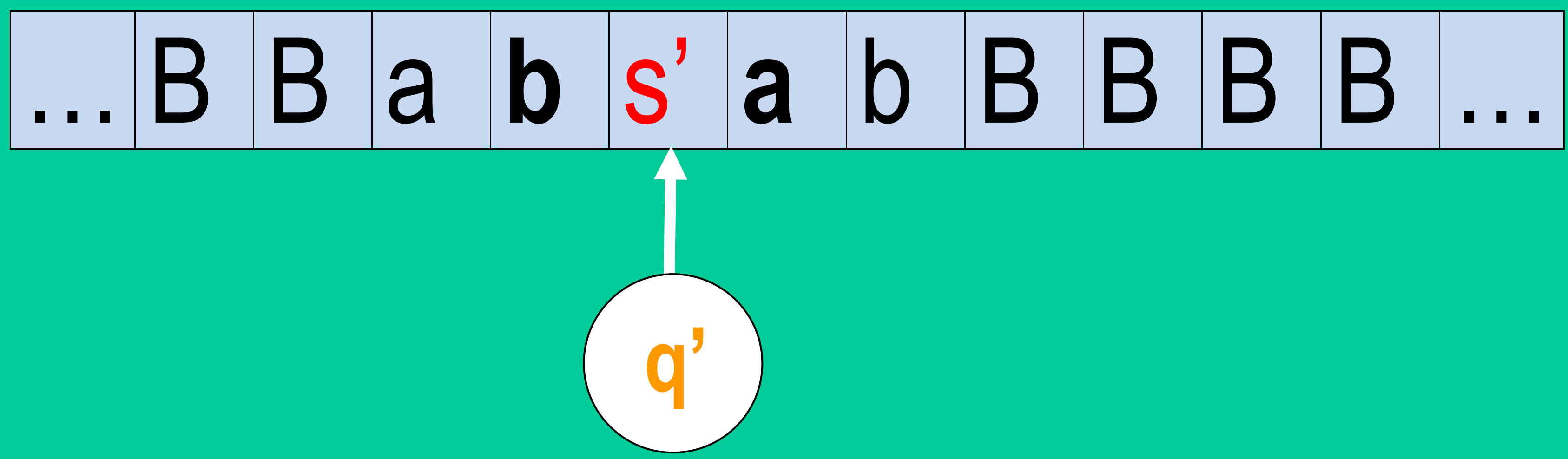
Si déplacement à droite : m = $\rightarrow$



Si déplacement à gauche : m = ←



Si pas de déplacement à gauche : m = l



Exécution d'une machine de Turing

L'exécution d'une machine de Turing sur un **mot** w de Σ^* noté :

$$W = w_1 w_2 w_3 \ldots w_p$$

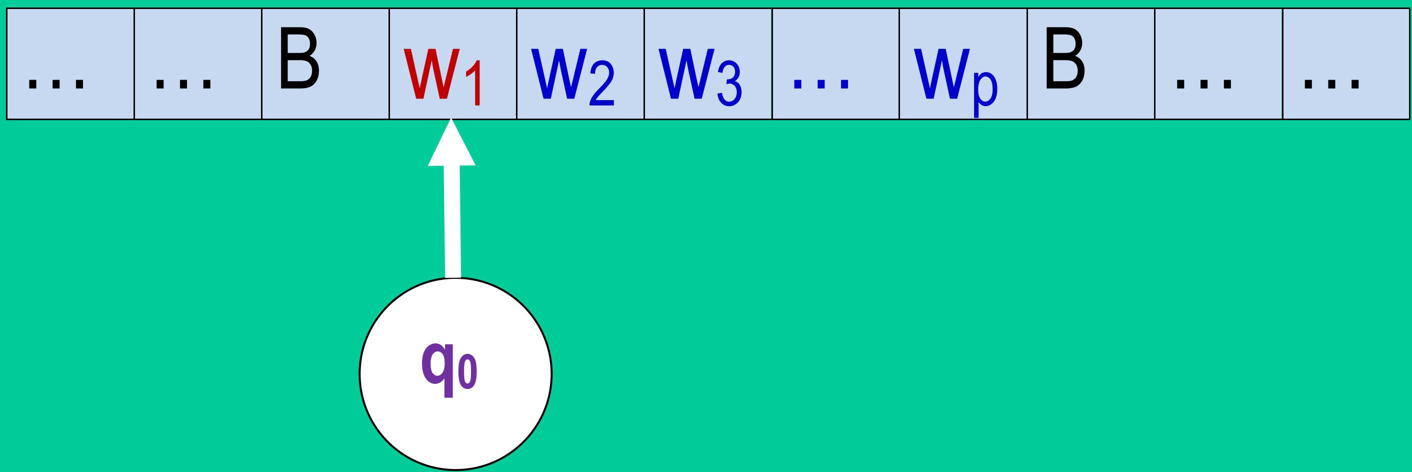
peut alors se décrire comme suit :

1-<u>initialement</u> :

a)-l'**entrée** $w_1\ w_2 \ldots w_p$ se trouve sur la bande,

b)-la tête de lecture est positionnée sur la **première lettre** w_1 du mot w.

c)-la machine est positionnée dans son état initial $\mathbf{q_0}$



2-<u>A chaque étape de l'exécution</u>

La machine **lit** le symbole se trouvant sous la tête de lecture ;

Et selon ce symbole, et selon son **état** actuel :

a)-elle le **remplace** par celui précisé par la fonction transition;

b)-**déplace**(ou non: symbole $|$) la tête de lecture d'une case vers la droite si $\rightarrow$, la gauche si $\leftarrow$ comme précisé par la **fonction de transition**;

c)-**change** d'état vers l'**état suivant**.

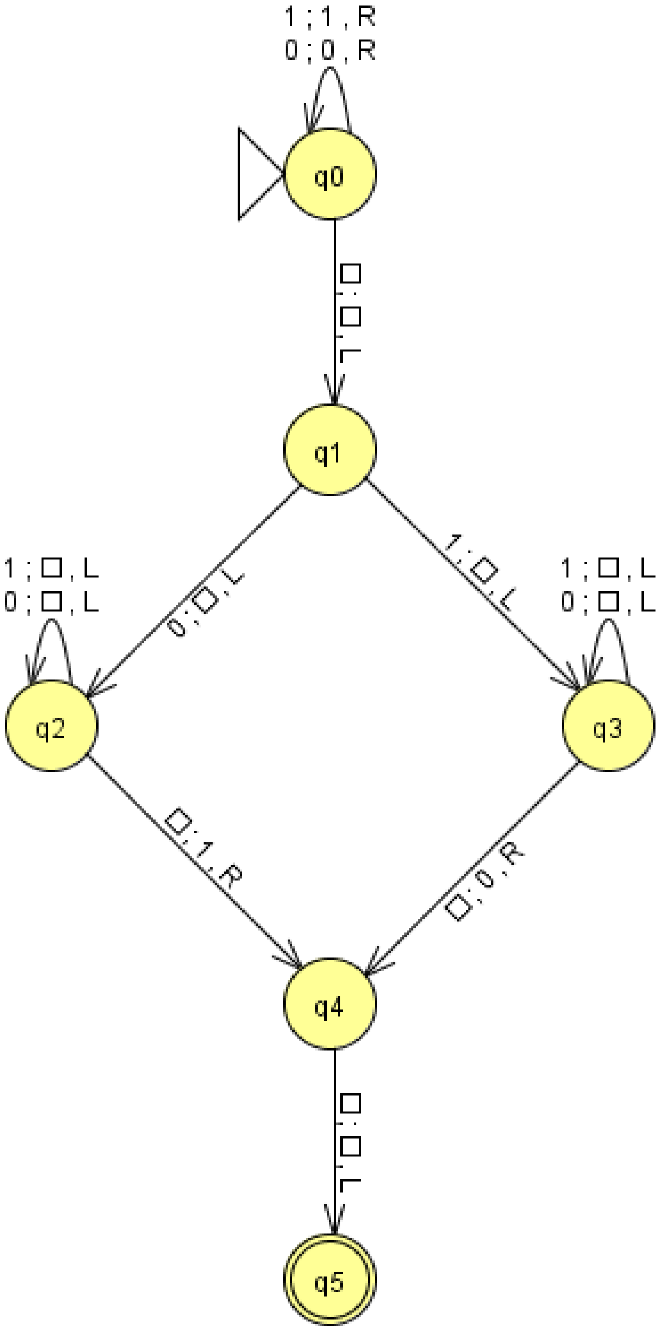
Exemple 1

Une machine de Turing déterminant si un nombre est pair

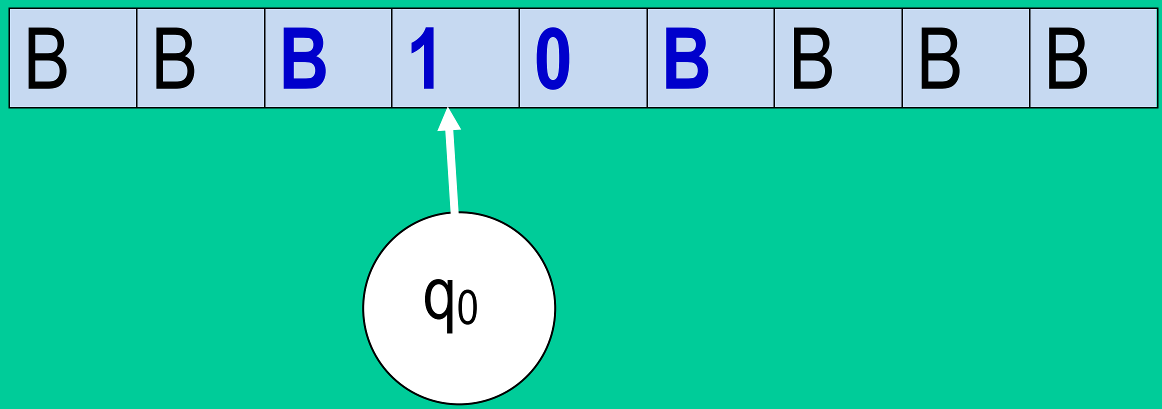
Entrée: le nombre à tester, sous forme binaire

Sortie: 1 si le nombre est pair,
0 sinon

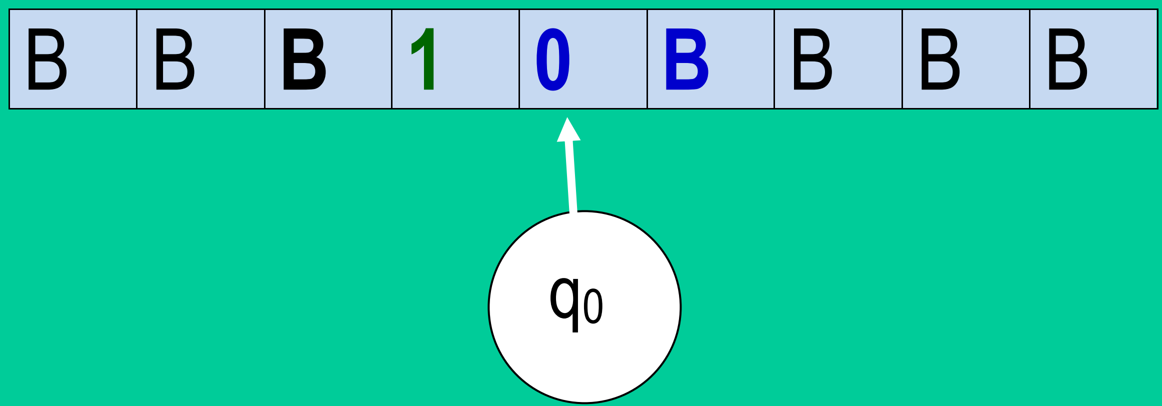
Etat actuel	Lit	Ecrit	Déplace	Etat suivant
q_0	0	0	→	q_0
	1	1	→	q_0
	B	B	←	q_1
q_1	0	B	←	q_2
	1	B	←	q_3
	B			
q_2	0	B	←	q_2
	1	B	←	q_2
	B	1	→	q_4
q_3	0	B	←	q_3
	1	B	←	q_3
	B	0	→	q_4
q_4	0	-	-	-
	1	-	-	-
	B	B	←	q_5


1 ; 1 , R
0 ; 0 , R
q0
□ ; □ , L
q1
0 ; □ , L
1 ; □ , L
1 ; □ , L
0 ; □ , L
1 ; □ , L
0 ; □ , L
q2
q3
□ ; 1 , R
□ ; 0 , R
q4
□ ; □ , L
q5

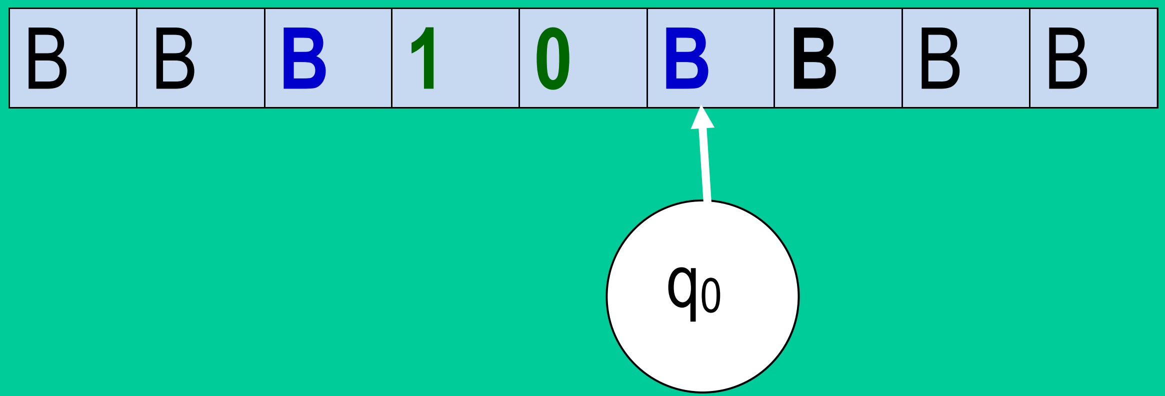
0-Etat initial



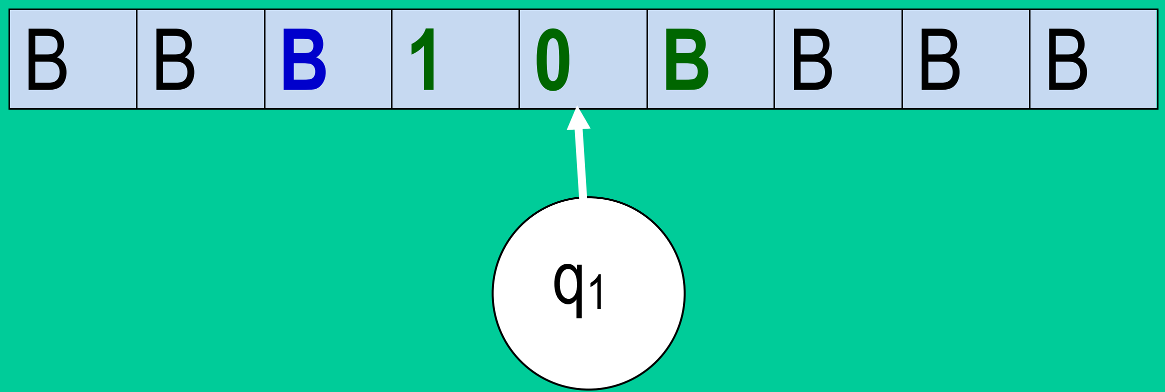
1-Ecrit 1 et va à droite



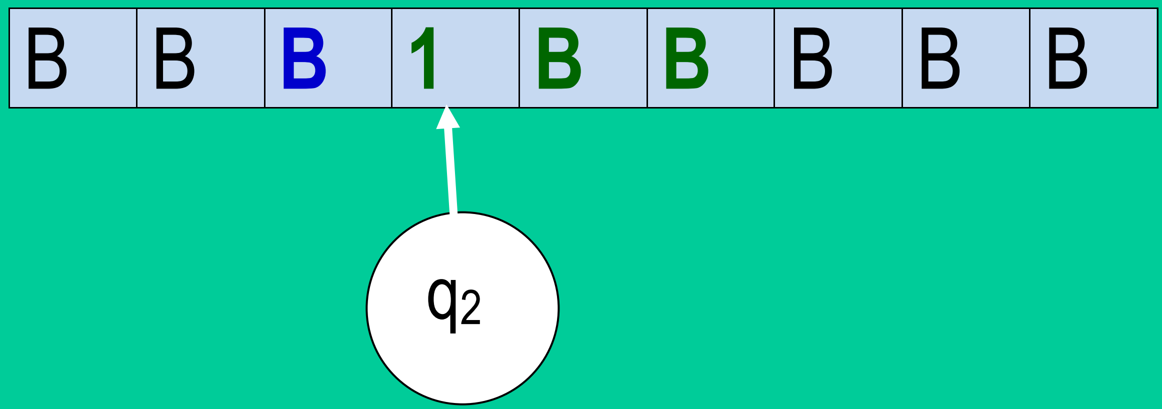
2-Ecrit 0 et va à droite



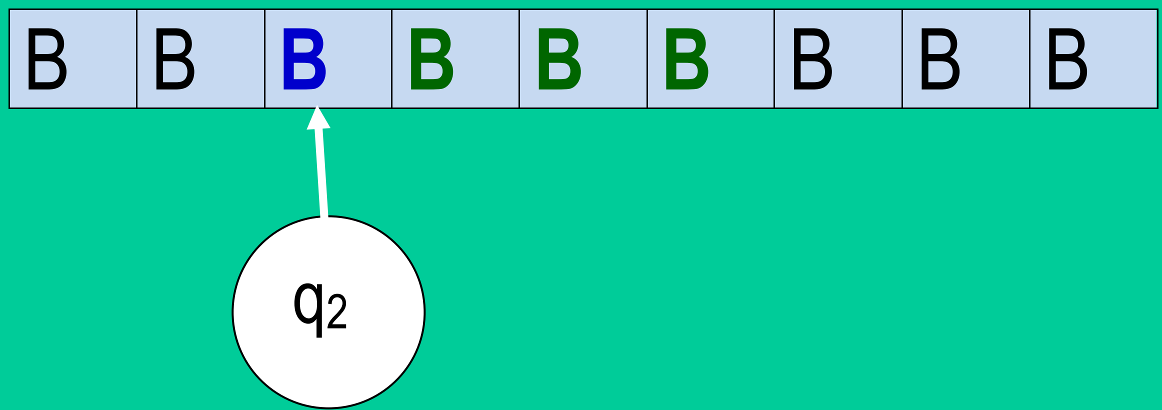
3-Ecrit B et revient à gauche: sur le dernier caractère



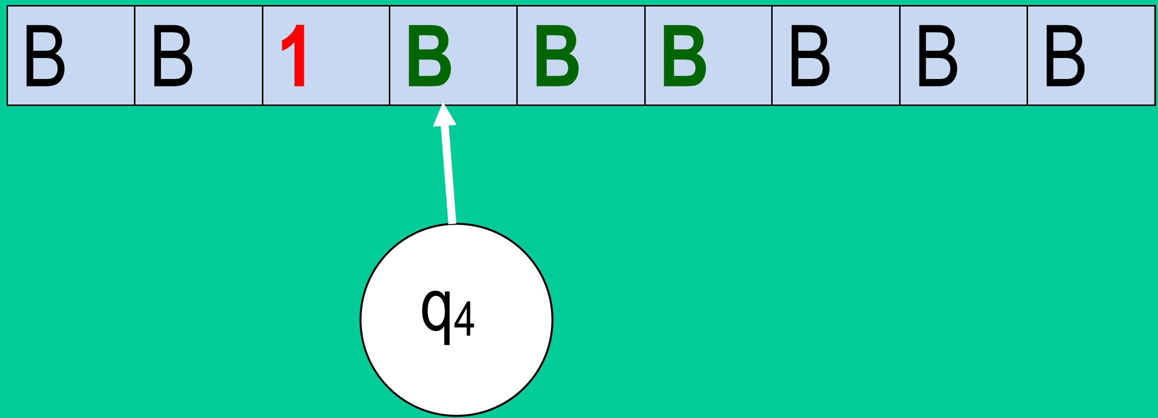
4-Efface et va à gauche :



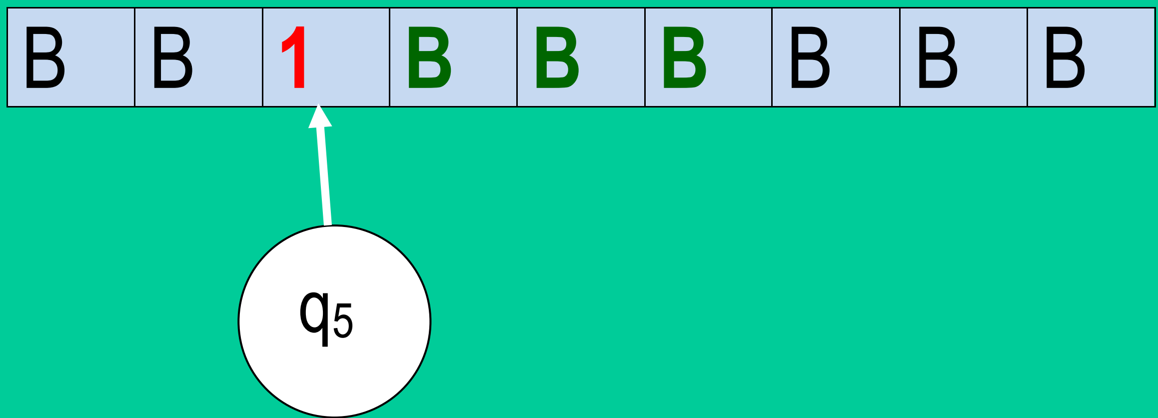
5-Efface et va à gauche :



6-Ecrit le résultat et va à droite :



7-Ecrit B et va à gauche: repositionne au début pour terminer:



3-Description formelle

1-Alphabet

Les symboles que la machine **lit** et **écrit** sur la bande constituent l'**alphabet** Σ de la machine.

L' alphabet Σ est fini.

Symbole spécial

Un symbole particulier noté **B** représente le **blanc.** Il figure dans les cases inoccupées.

B	B	B	a	b	**u**	a	b	B	B	B	B	B

Remplacer un **u** par **B** revient à effacer **u**.

B	B	B	a	b	**B**	a	b	B	B	B	B	B

Etats internes

Les **états internes** sont notés :

$q_1, q_2, q_3, \ldots$

La machine possède un **ensemble fini** d'états noté **Q**.

Parmi ces états, on distingue :

- un état **initial** : $q_0 \in Q$
- un état d'**acceptation** : $q_a \in Q$;
- un état d'**arrêt** : $q_r \in Q$;

Définition formelle

Une machine de Turing est un 8-uplet

$$M = (Q; \Sigma; \Gamma; B; \delta; q_0; q_a; q_r)$$

où :

Q est l'ensemble fini des **états** ;

Σ est un **alphabet fini**;

Γ est fini et constitue l'**alphabet de travail**: $\Sigma \subset \Gamma$;

B $\in \Gamma$ est le caractère **blanc** ;

δ est la fonction de **transition**, possiblement partielle:

$$\delta: \quad Q \times \Gamma \;\rightarrow\; Q \times \Gamma \times \{\leftarrow, \mid, \rightarrow\}.$$

Machine de Turing canonique

Si l'on impose les 2 hypothèses suivantes:

1- coder les entrées et les sorties sous forme binaire

$$\Sigma=\{0,1\}$$

2- indiquer l'absence de caractère par le caractère ε,

on obtient une représentation **canonique** de la machine de Turing.

D’autres hypothèses sont possibles pour la définition des machines de Turing :

- plusieurs bandes,
- autres opérations élémentaires,
- autres alphabets de caractères, ...

Mais on a montré que **toutes** les machines ainsi construites sont équivalentes à une **machine de Turing canonique.**

Les machines de Turing fournissent donc, par définition, une représentation possible pour les **algorithmes**.

De plus, l’expérience acquise dans le domaine de l’informatique invite à penser que les machines de Turing constituent en fait un **outil encore plus puissant**:

Thèse de Church:

« **tout** **algorithme** peut être représenté sous la forme d’une **table de transitions** d’une machine de Turing »

L'idée sous-jacente est la suivante:

-les machines de Turing, malgré leur apparente simplicité,

-possèdent une puissance suffisante pour permettre de résoudre **tout** problème pouvant l'être de façon automatique.

Une formulation savante de la thèse de Church est donc:

«tout problème calculable est Turing calculable»

Notion de configuration

Intuitivement, une Machine de Turing fonctionne selon un **cycle**.

Ce cycle consiste à passer successivement par trois phases:

-phase de **lecture**
-phase de **calcul**
-phase d'**action**

1-**<u>Phase de lecture</u>:**

-La machine **lit** le contenu de la case courante.

-Elle le transmet comme paramètre d'entrée à la fonction de transition.

2-**<u>Phase de calcul</u>:**

La fonction de transition est **calculée** en fonction de :

-l'état courant

-et du symbole contenu dans la case courante.

3-**<u>Phase d’action</u>:**

L’**action** déterminée par la fonction de transition est effectuée.

Elle comporte:

-l’**écriture** d’un symbole dans la case courante

-et un **déplacement** de la tête de lecture.

Description formelle

Le programme exécuté par une machine de Turing se définit à l'aide des notions:

- de **configurations**
- et de la **relation successeur** entre configurations d'une machine de Turing.

Qu’est-ce qu’une configuration ?

Une **configuration** correspond à toute l’information nécessaire pour:

- décrire l’**état** de la machine à un instant donné,

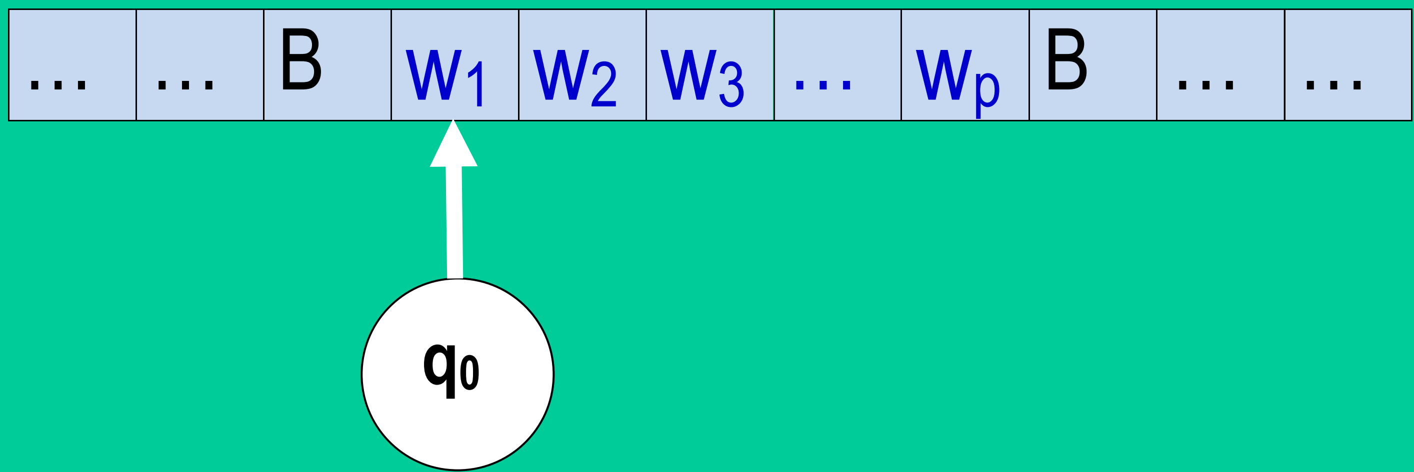
- décrire **la bande** et la **position** de la tête de lecture

- déterminer les **états ultérieurs** de la machine.

Formellement, une configuration est donnée par :

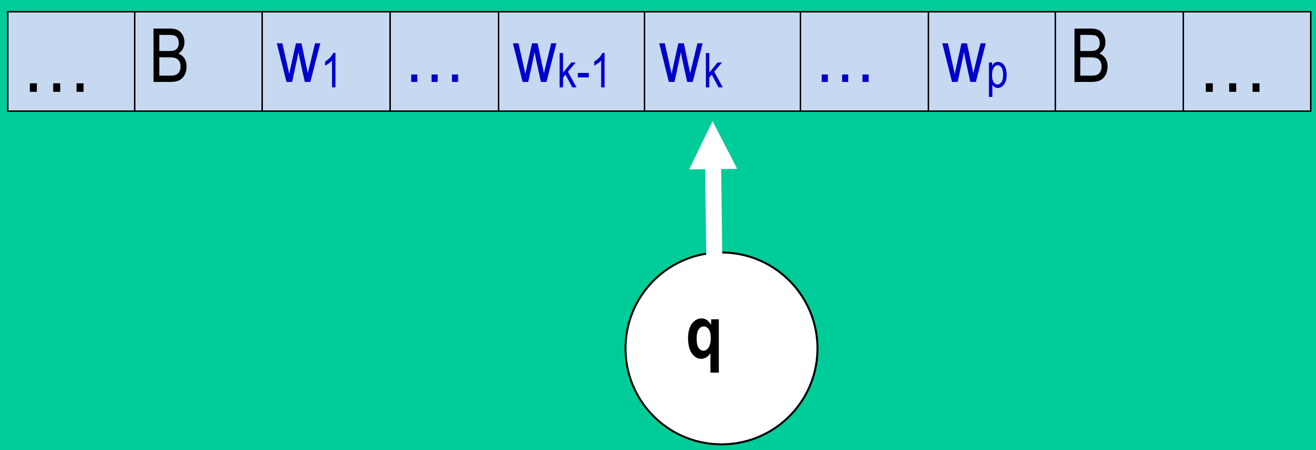
- la **description de la bande**,
- la **position de la tête** de lecture,
- et l'**état interne**.

Initialement la machine contient un mot en entrée de longueur finie et fixée.



La tête de lecture est positionnée au-dessus de la première lettre de w.

A chaque étape, la machine déplace la tête de lecture au plus d’**une seule** case.

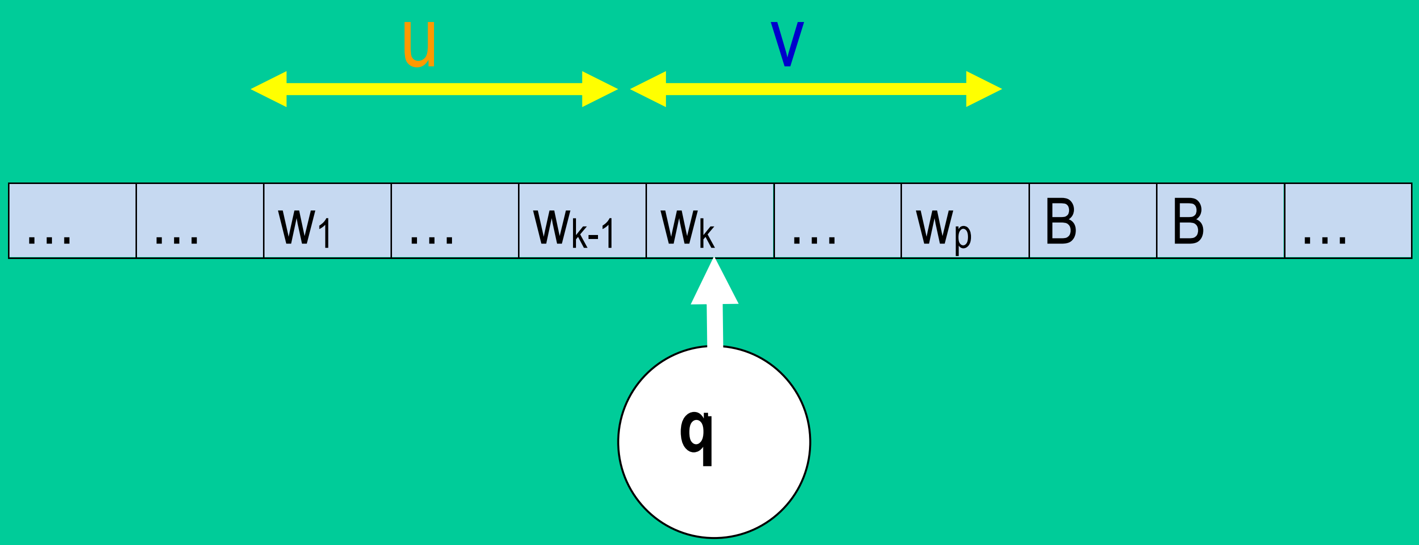


Après **k** étapes, la tête de lecture a parcouru :

-au plus, **k** cases (vers la droite et/ou vers la gauche)

-à partir de sa position initiale.

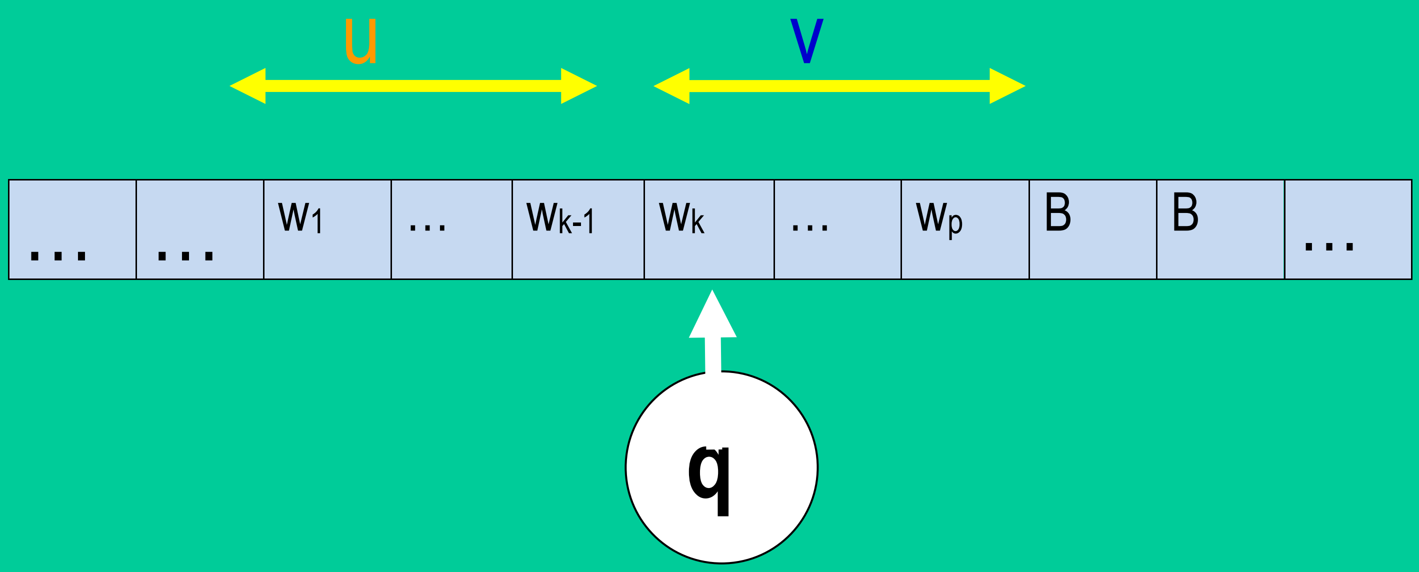
Ce parcours est représenté par **deux préfixes** finis.



Ces deux préfixes finis représentent :

- le contenu de ce qui est **à droite** de la tête de lecture : v

- le contenu de ce qui est **à gauche** de la tête de lecture : u



Une configuration sera donc un élément de :

$$Q \times \Gamma^* \times \Gamma^*:$$

Formellement une configuration se note:

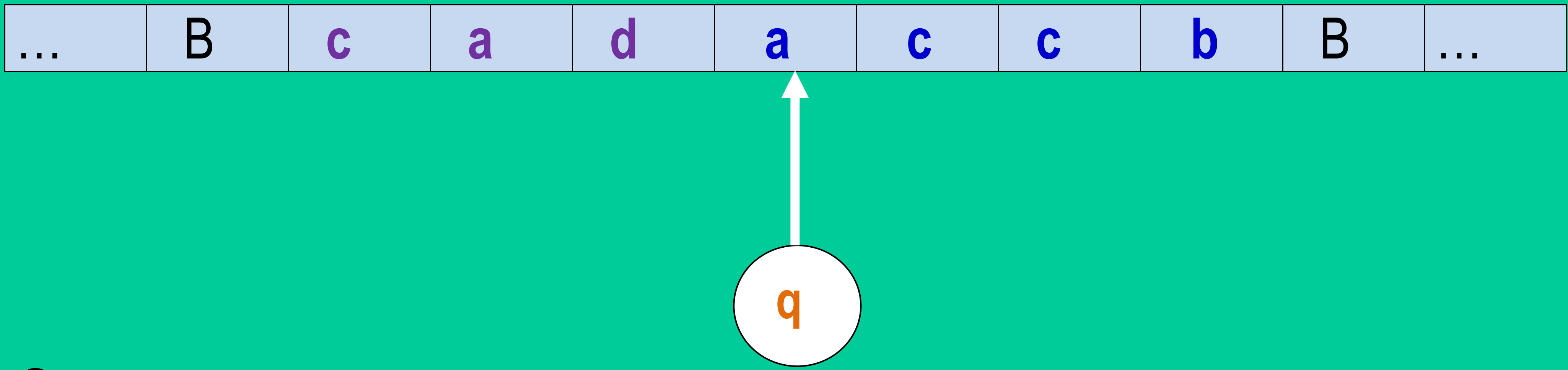
$$C = (q; u; v),$$

Avec :

$u; v \in \Gamma^*$

$q \in Q$

Ainsi, la configuration:



Se note:

$$(q;\ cad;\ accb)$$

avec u= cad et v=accb

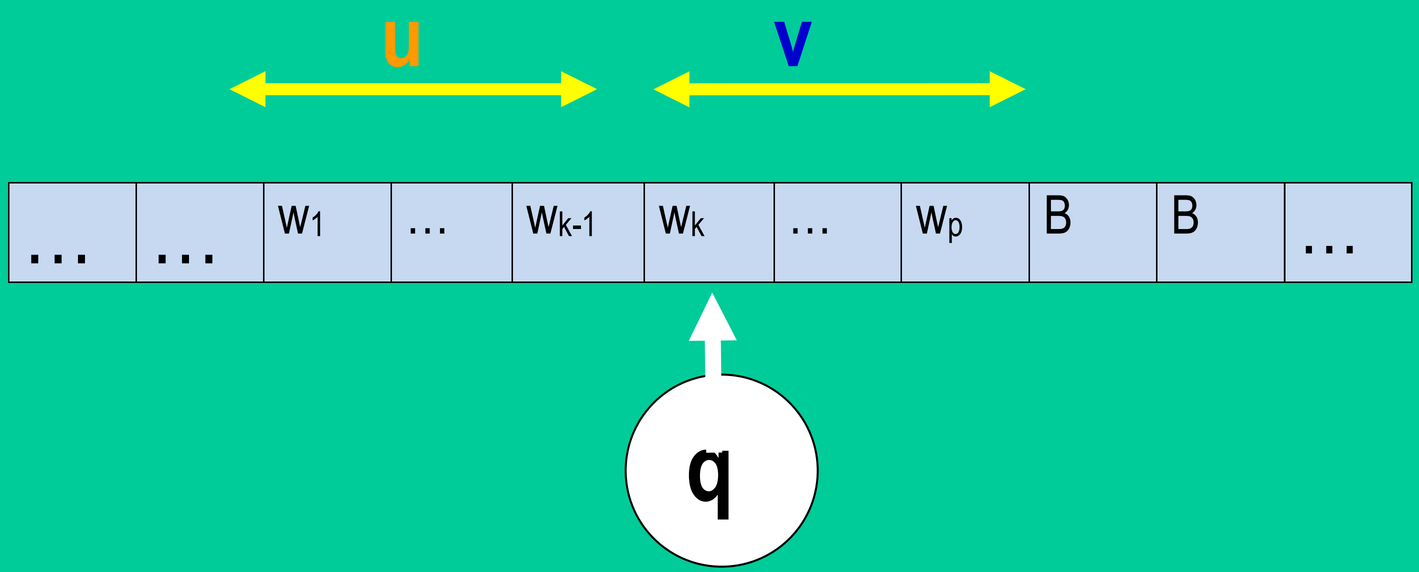
configuration initiale

Pour $W \in \Sigma^*$, la **configuration initiale** correspondante à w est la configuration :

$$C[W] = (q_0; \varepsilon; W).$$

Où ε représente la chaîne vide et $W = w_0\, w_1\, w_2 \ldots$

configuration courante



$$C_k = (\mathbf{q};\ \mathbf{u};\ \mathbf{v}).$$

configuration acceptante

Une configuration (q; u; v) est dite **acceptante** si:

$$q = \mathbf{q_a},$$

configuration refusante

Une configuration (q; u; v) est dite **refusante** si :

$$q = \mathbf{q_r}$$

Successeur direct d'une configuration

On note:

$$C \vdash C'$$

si la configuration C' est le **successeur direct** de la configuration C par le programme (donné par δ) .

Formellement, si :

$$C = (q; u; v)$$

et si **a** désigne la première lettre de v, et si :

$$\delta(q; \mathbf{a}) = (q'; a'; m')$$

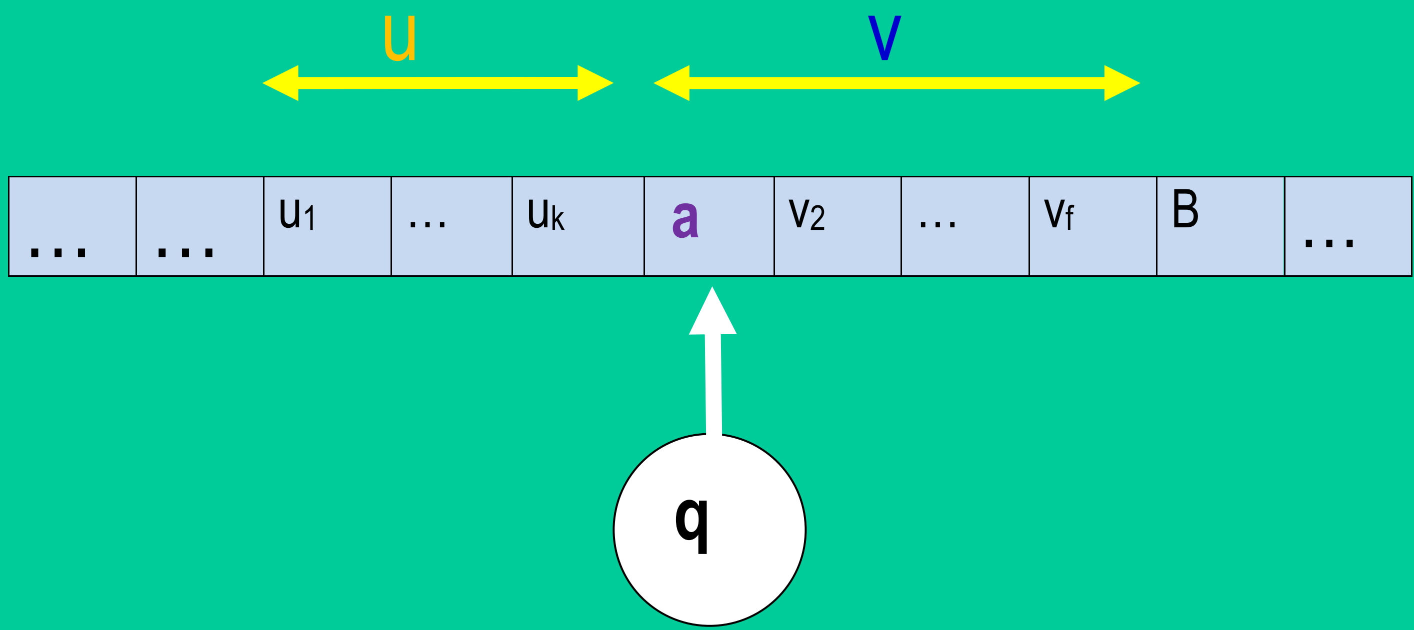
Alors :

$$C \vdash C'$$

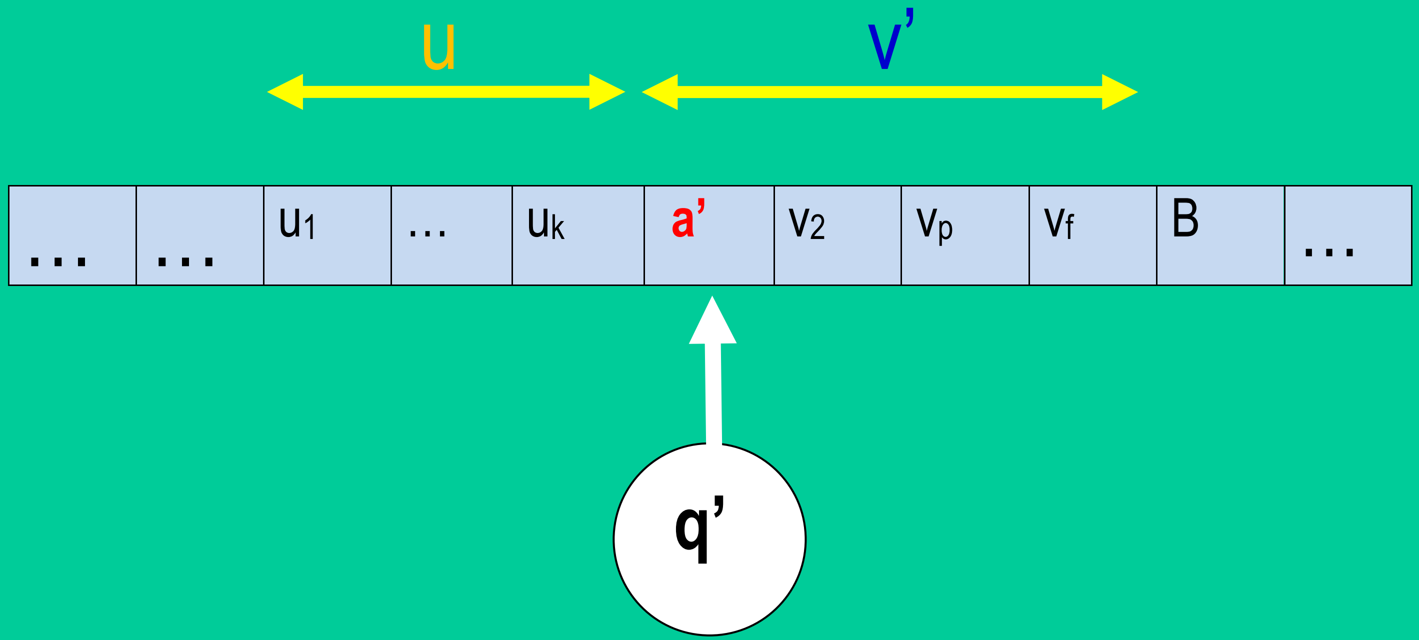
est définie par:

1- **si** m' = | , alors   C' = (q'; u; v')

a étant la première lettre de v :

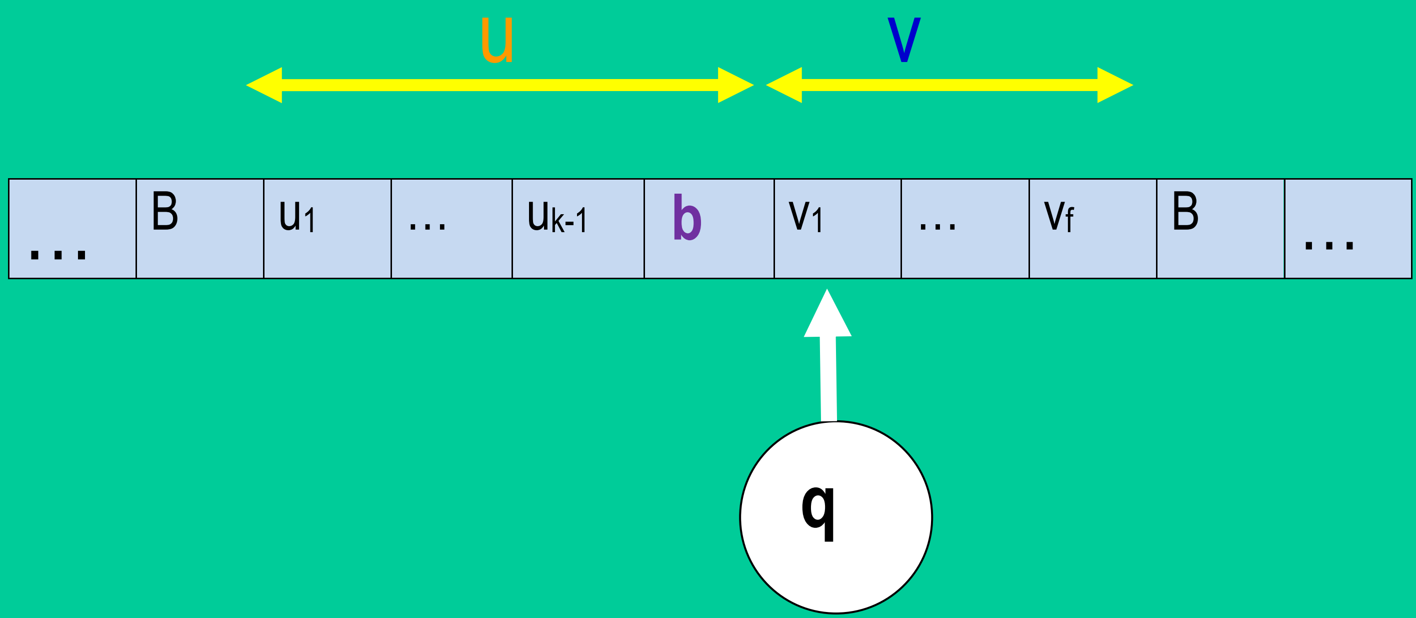


• **v'** est obtenu en **remplaçant la première** lettre **a** de v par **a'** ;



2- si m' = $\leftarrow$ alors C' = (q'; u'; v')

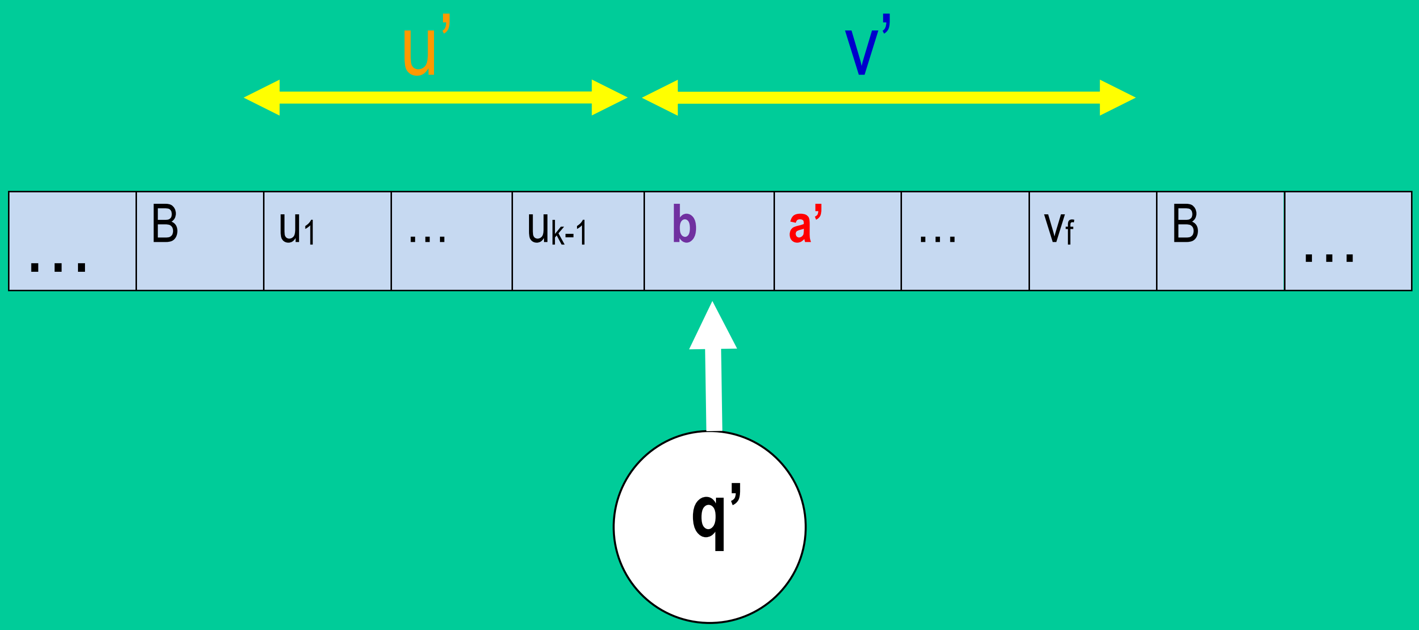
Si b est la **dernière** lettre de u:



- $v' = b\ a' v_2 \ldots v_f$

et u' est obtenu en **supprimant la dernière** lettre **b** de u ;

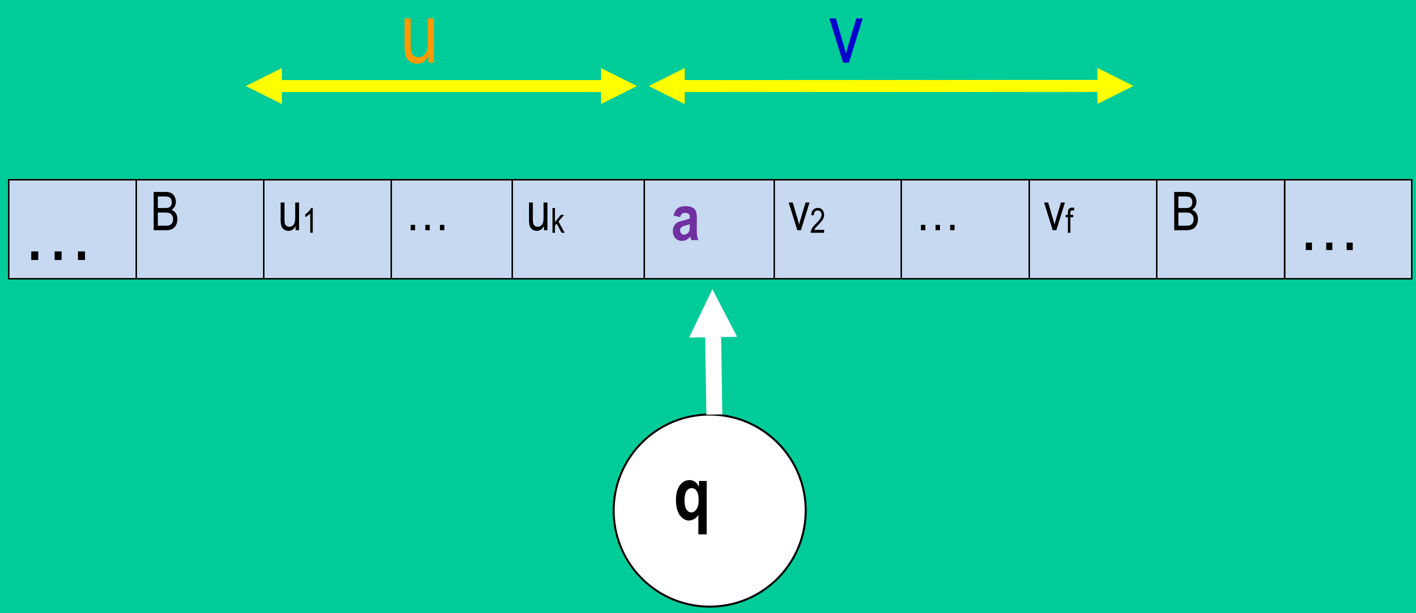
- $u = u'\ b$



3- si $m' = \rightarrow$ alors

$$C' = (q'; u' ; v')$$

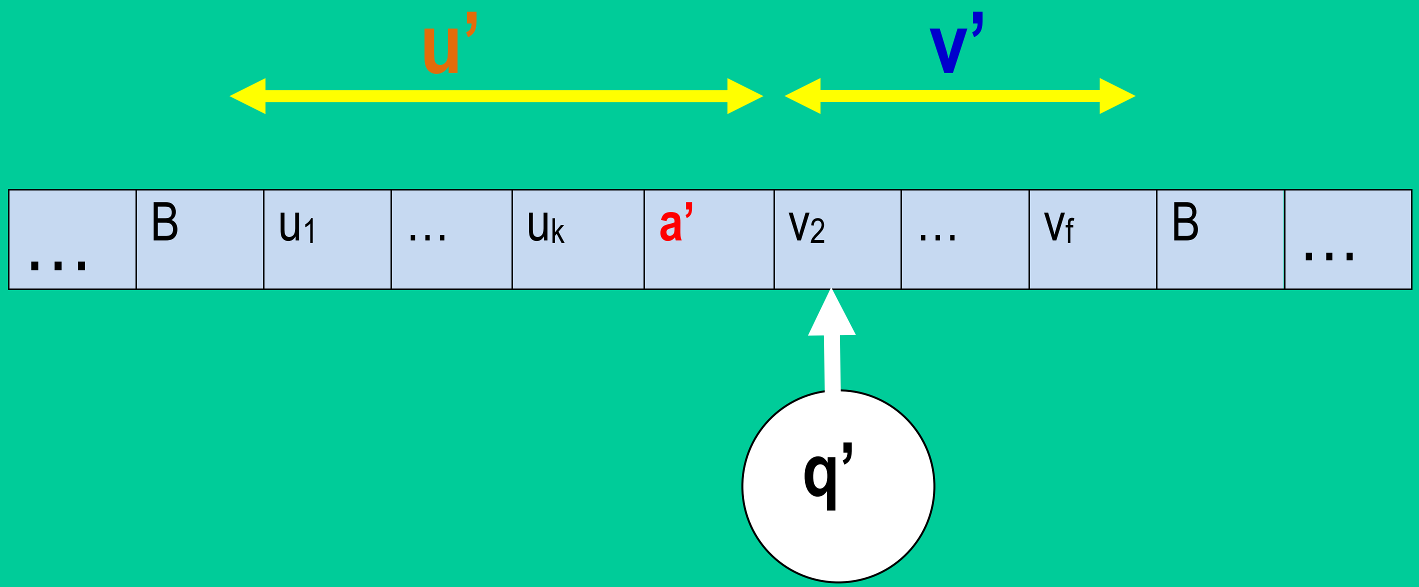
Si **a** est la **première** lettre de v.



- $u' = u\, a'$

et v' est obtenu en **supprimant la première** lettre de v.

- $v' = v_2\, v_3 \ldots v_f$



Mot accepté en temps t

Un mot $w \in \Sigma^*$ est dit **accepté** en temps **t** s'il existe une suite de configurations :

$$C_0, C_1, \ldots, C_i, C_{i+1}, \ldots, C_t$$

avec :

1. $C_0 = C[w]$;

2. $C_i \vdash C_{i+1}$ pour tout $i < \mathbf{t}$;

3. aucune configuration C_i pour $i < t$ n'est **acceptante** ou **refusante**.

4. $C_{\mathbf{t}}$ est **acceptante**.

Mot refusé en temps t

Un mot $w \in \Sigma^*$ est dit **refusé** en temps **t** s'il existe une suite de configurations

$$C_0, C_1, \ldots, C_i, C_{i+1}, \ldots, C_t$$

avec :

1. aucune configuration C_i pour $i < t$ n'est **acceptante** ou **refusante**.
2. C_t est **refusante**.

Machine qui boucle sur un mot

On dit que la machine de Turing **boucle** sur un mot w, si w n'est :

- ni **accepté**,
- et ni **refusé**.

Remarque:

La terminologie "boucle" signifie simplement que la machine ne s’**arrête pas** sur ce mot.

Cela ne veut pas dire nécessairement que l’on répète à l’infini les mêmes instructions.

Machine de Turing non déterministe

La définition d'une machine de Turing **non-déterministe** est exactement la même que celle de la machine de Turing.

Sauf sur un point : δ n'est plus une **fonction** mais une **relation** de la forme :

$$\delta \subset (Q \times \Gamma) \times (Q \times \Gamma \times \{\leftarrow, \mid, \rightarrow\})$$

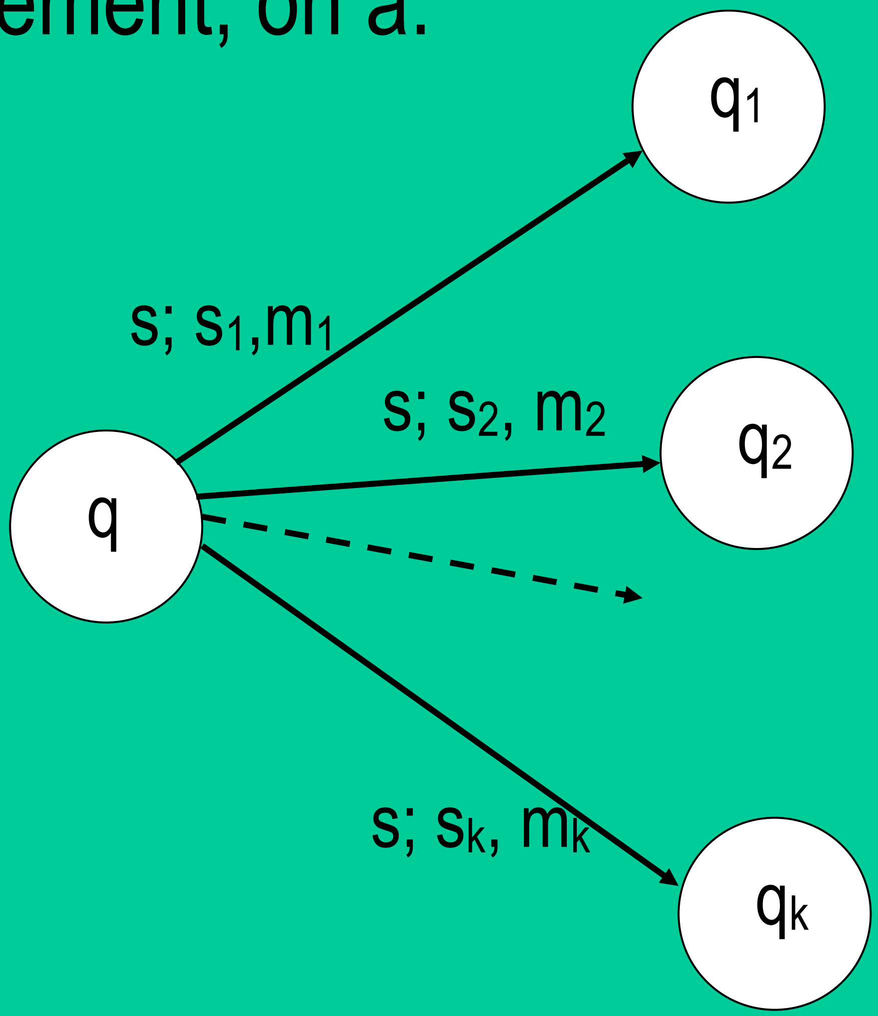
En d'autres termes, pour un état q et un symbole lu s , δ ne définit pas un **seul triplet** :

$$(q' , s' , m) \in Q \times \Gamma \times \{ \leftarrow , \mid , \rightarrow \}$$

mais un **ensemble de triplets** :

$$\{ (q_1 , s_1 , m_1) , (q_2 , s_2 , m_2) , \ \ldots (q_k , s_k , m_k) \}$$

Graphiquement, on a:



Intuitivement, lors d’une exécution, la machine a la possibilité de choisir n’**importe quel** triplet.

Formellement, cela s’exprime par le fait que l’on peut passer de la configuration C à la configuration successeur C’ :

$$\mathbf{C \vdash C'}$$

si et seulement si :

$$((q; a) \times (q'; a'; m')) \in \delta$$

La différence est qu'une machine de Turing non déterministe :

- n'a **pas une exécution unique** sur une entrée w,
- mais éventuellement **plusieurs**.

Machine de Turing universelle

Le **programme** de la machine de Turing correspond à sa **fonction de transition** δ.

Sur ce point, une Machine de Turing ne constitue donc pas encore un **modèle** pour l'**ordinateur.**

Pourquoi ?

- Pour l'ordinateur le **programme** est un élément **externe**: il fait partie des **données d'entrée**.

- Pour la **Machine de Turing**, le programme est un élément **constitutif** de la machine : c'est sa fonction de transition δ.

Si l'on désire qu'une machine de Turing constitue un **modèle pour l'ordinateur**, il faut réunir deux conditions:

1- que sa **fonction** de transitions soit **constante**,

2- que son **fonctionnement** soit **entièrement imposé par ses données d'entrées**.

On peut donc changer le **programme** de la machine de Turing.

Construction d'une machine de Turing universelle

Une telle machine est appelée **machine de Turing universelle.**

On peut construire :

- une machine de **Turing universelle** $\mathbf{T_0}$
- permettant de **simuler** le fonctionnement d'une machine de Turing quelconque **T**.

T_0 doit avoir le même comportement que T pour ce qui est des entrées et des sorties.

L'idée permettant de construire une telle machine **T_0** est la suivante:

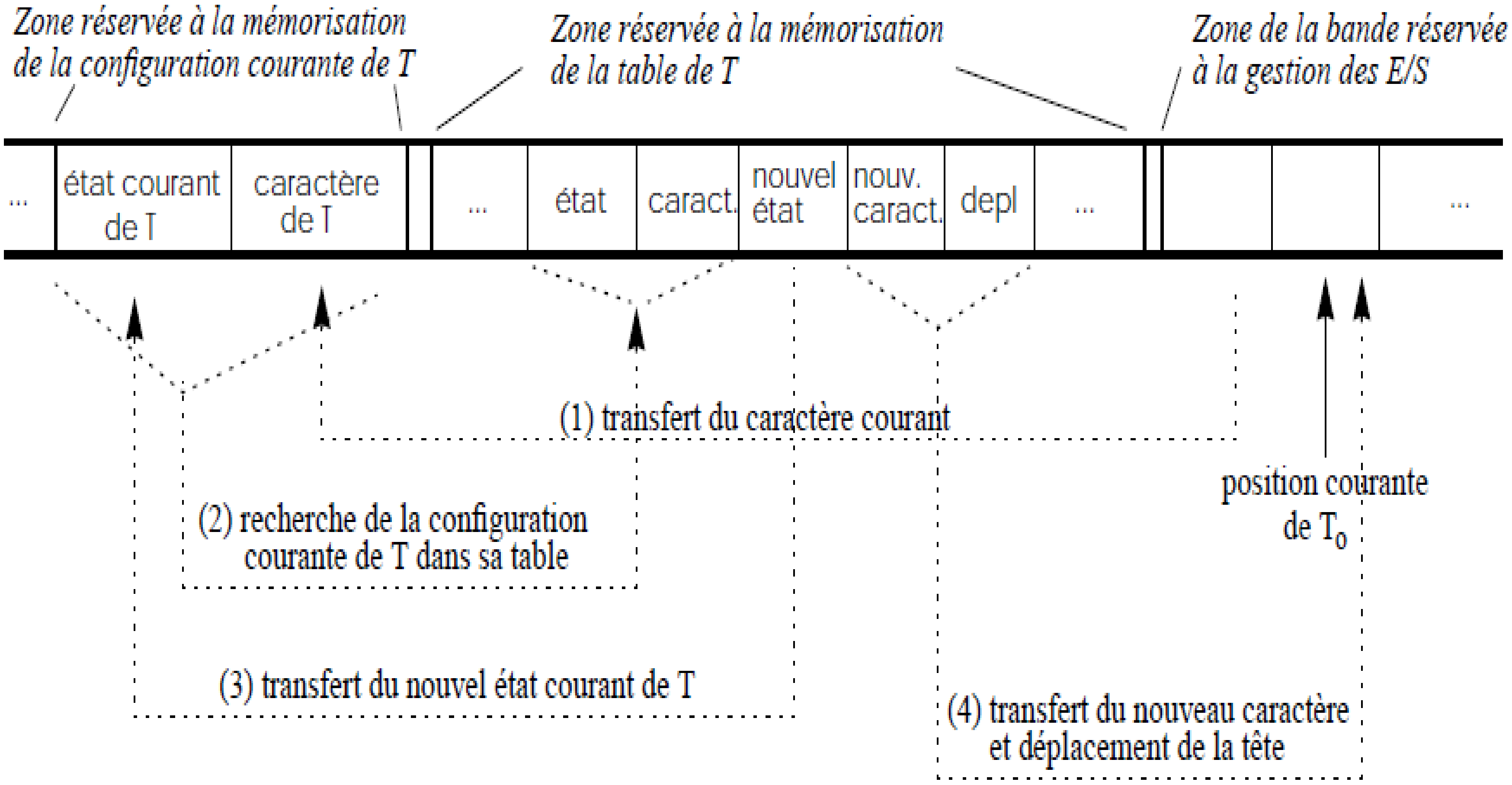
• la **table de transitions** de la machine T à simuler est codée sous la forme d'une séquence binaire W;

•la **bande** de la machine universelle est séparée en deux zones distinctes :

1-une zone pour stocker la séquence binaire représentant :

- la **fonction de transition δ de T**
- et sa **configuration** C.

2-l'autre zone permet de gérer les entrées (données) et les sorties (résultat)


Zone réservée à la mémorisation de la configuration courante de T
Zone réservée à la mémorisation de la table de T
Zone de la bande réservée à la gestion des E/S
...
état courant de T
caractère de T
...
état
caract.
nouvel état
nouv. caract.
depl
...
...
(1) transfert du caractère courant
(2) recherche de la configuration courante de T dans sa table
(3) transfert du nouvel état courant de T
(4) transfert du nouveau caractère et déplacement de la tête
position courante de T_0