Jean Goubault-Larrecq

Calculabilité

1. Machines de Turing

Aujourd'hui

- Machines de Turing
- Fonctions calculables, problèmes décidables
- Fonctions semi-calculables, problèmes récursivement énumérables
- Variantes
- Basé sur le poly d'Hubert Comon:
  http://www.lsv.fr/~comon/Calculabilite1/cours20.1.pdf

Qu'est-ce qu'une fonction calculable?

Das Entscheidungsproblem

- « Calculable » a un sens intuitif depuis longtemps
- En 1928, David Hilbert and Wilhelm Ackermann:
  y a-t-il un **algorithme** prenant une formule *F* de logique du premier ordre en entrée,
  et **décidant** si *F* est valide ou pas?
- **Problème**: aucune définition connue à l'époque de la notion d'**algorithme**, de **calcul**

D. Hilbert in 1912
By Unknown author - Possibly Reid, Constance (1970) Hilbert, Berlin, Heidelberg: Springer Berlin Heidelberg Imprint Springer, p. 230 ISBN: 978-3-662-27132-2., Public Domain, https://commons.wikimedia.org/w/index.php?curid=36302

W. Ackermann, circa 1935
By Unknown author - Steinfurt, Public Domain, https://commons.wikimedia.org/w/index.php?curid=9379449

La réponse

- Non! Il n'y en a pas: la validité des formules de la logique du premier ordre est **indécidable**
- Dû indépendamment à Alonzo Church et Alan Turing (1936)
- … qui ont au passage défini ce qui est **calculable**

(de deux façons très différentes, mais équivalentes)

A. Church
By Princeton University, Fair use, https://en.wikipedia.org/w/index.php?curid=6082269

AN UNSOLVABLE PROBLEM OF ELEMENTARY NUMBER THEORY.[1]

By Alonzo Church.

1. **Introduction.** There is a class of problems of elementary number theory which can be stated in the form that it is required to find an effectively calculable function f of n positive integers, such that $f(x_1, x_2, \cdots, x_n) = 2$ [2] is a necessary and sufficient condition for the truth of a certain proposition of elementary number theory involving $x_1, x_2, \cdots, x_n$ as free variables.

An example of such a problem is the problem to find a means of determining of any given positive integer n whether or not there exist positive integers x, y, z, such that $x^n + y^n = z^n$. For this may be interpreted, required to find an effectively calculable function f, such that $f(n)$ is equal to 2 if and only if there exist positive integers x, y, z, such that $x^n + y^n = z^n$. Clearly the condition that the function f be effectively calculable is an essential part of the problem, since without it the problem becomes trivial.

Another example of a problem of this class is, for instance, the problem of topology, to find a complete set of effectively calculable invariants of closed three-dimensional simplicial manifolds under homeomorphisms. This problem can be interpreted as a problem of elementary number theory in view of the fact that topological complexes are representable by matrices of incidence. In fact, as is well known, the property of a set of incidence matrices that it represent a closed three-dimensional manifold, and the property of two sets

A. Turing en 1938
Par Auteur inconnu. Public Domain according to https://www.theguardian.com/science/2012/mar/06/alan-turing-exhibition-enigma-codebreaker. — https://berichtenuithetverleden.wordpress.com/2013/01/29/alan-turing-1912-1954/, Domaine public, https://commons.wikimedia.org/w/index.php?curid=39218619

230 A. M. Turing ,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions

Fonctions calculables

- Plusieurs modèles:
  - Machines de Turing

Par Sethwoodworth sur Wikipédia anglais, taken by Bcjo
Transféré de en.wikipedia à Commons par Mardetanha u
CommonsHelper., CC BY 3.0, https://
commons.wikimedia.org/w/index.php?curid=55

  - Machines à compteurs (Marvin Minsky)
  - Fonctions récursives (Kurt Gödel)—

  - λ-calcul (Alonzo Church)

Par Auteur inconnu — Familienalbum der Familie Gödel,
Scan from Gianbruno Guerrerio, Kurt Gödel - Logische Paradoxien
und mathematische Wahrheit, S.24, Domaine public, https://
commons.wikimedia.org/w/index.php?curid=10595692

- Ils sont tous équivalents (« thèse de Church »)

Machines de Turing

Modèles de calcul

- Haut niveau: langages de programmation (Caml, Prolog, Python, Lisp, etc.)
- Bas niveau: assembleur, langage machine
- Encore plus bas niveau: machines de Turing, machines à compteurs (de Minsky), RAM

Mots, et notations

- Une machine de Turing travaille sur des **mots finis**, sur un **alphabet** Σ (un ens. fini non vide, dit de **lettres**)
- On note Σ^* l'ensemble des mots (finis) sur Σ
- a, b, … : lettres,
  promues silencieusement en mots de longueur 1
- ε : mot **vide**
- ww' : **concaténation** des deux mots w et w'
  aussi aw, wb, etc.

Langages

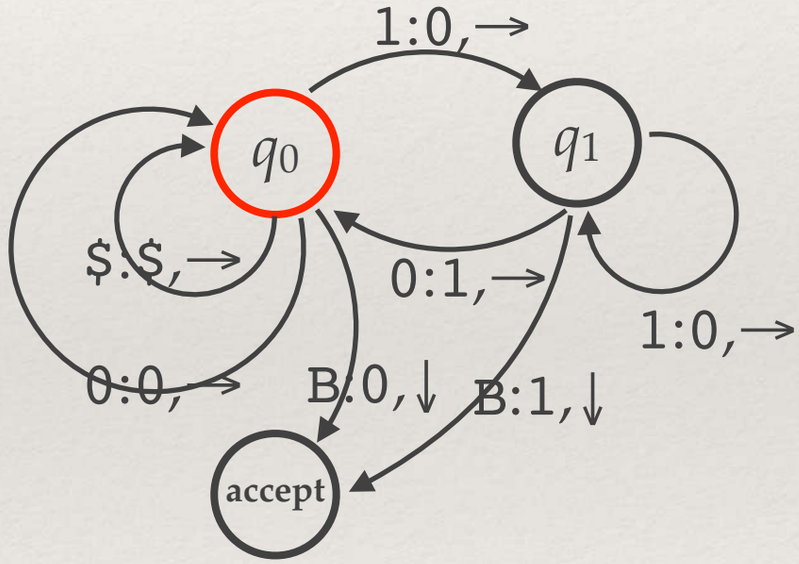
- Un **langage** (sur Σ) est une partie L de Σ^*
- Pour chaque mot w, on note encore w le langage $\{w\}$
- Note: ε est le langage $\{\varepsilon\}$ $\neq$ langage vide $\emptyset$
- **Concaténation** de langages: $LL' \stackrel{\text{def}}{=} \{ww' \mid w \in L, w' \in L'\}$
- **Union** de langages: $L+L' \stackrel{\text{def}}{=} L \cup L'$
- Exemple: qu'est $\varepsilon+\$(\Sigma-\{\$\})^*$ ($ lettre)?

Machines de Turing

- Une **bande** (ou ruban)
  ≈ mémoire d'un ordinateur

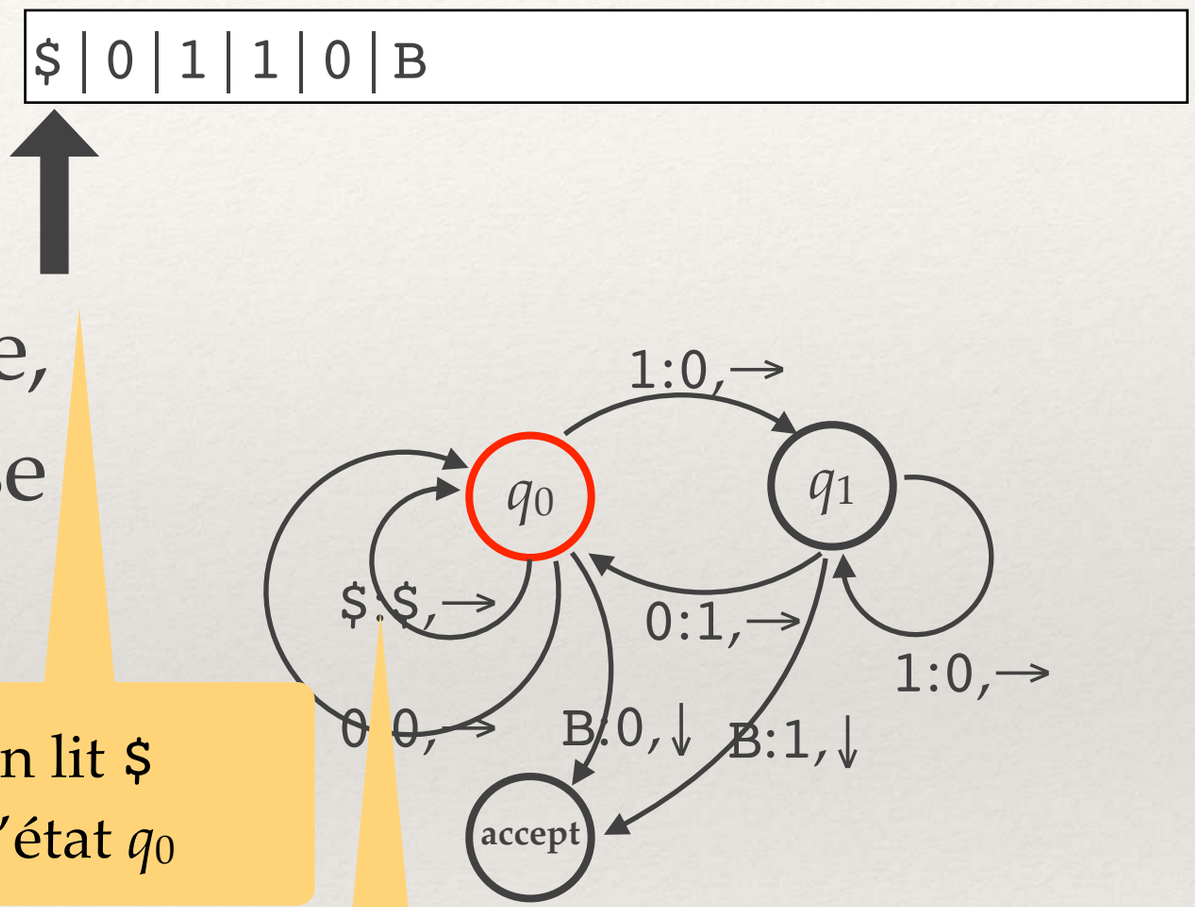
- Une **tête** de lecture/écriture, voyageant d'au plus un case à chaque étape de calcul

- Un **contrôle** fini
  ≈ programme



Machines de Turing

- Une **bande** (ou ruban)
  ≈ mémoire d'un ordinateur
- Une **tête** de lecture/écriture,
  voyageant d'au plus un case
  à chaque étape de calcul
- Un **contrôle** fini
  ≈ programme

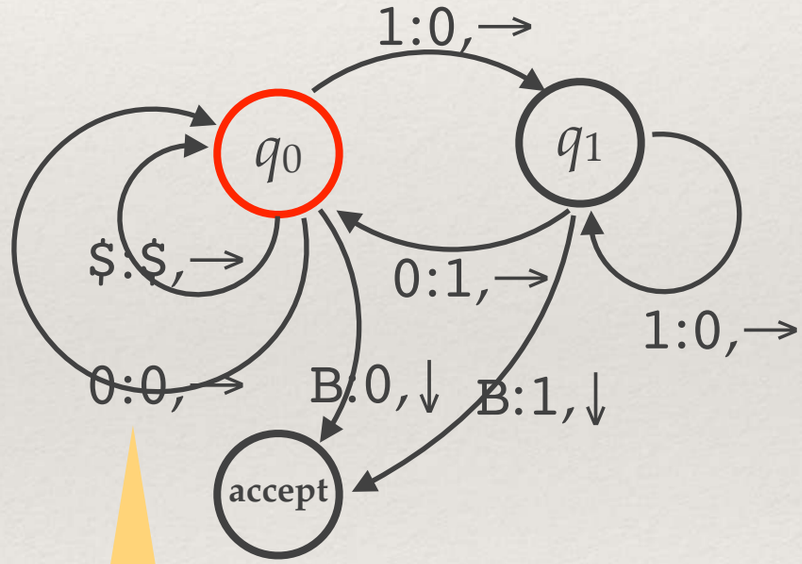


Machines de Turing

- Une **bande** (ou ruban)
  ≈ mémoire d'un ordinateur

- Une **tête** de lecture/écriture, voyageant d'au plus un case à chaque étape de calcul

- Un **contrôle** fini
  ≈ programme



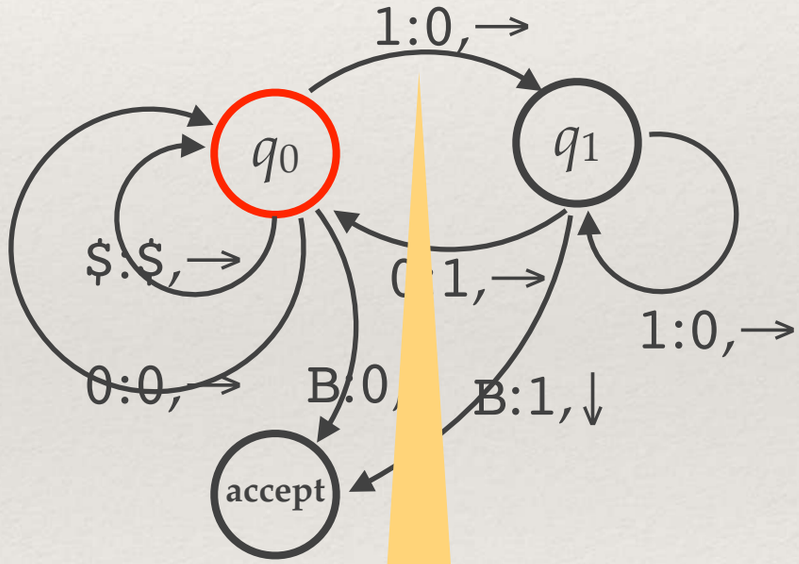
Le contrôle nous dit:
écrire 0 et
aller à droite (→),
rester en q_0

Machines de Turing

- Une **bande** (ou ruban)
  ≈ mémoire d'un ordinateur

- Une **tête** de lecture/écriture, voyageant d'au plus un case à chaque étape de calcul

- Un **contrôle** fini
  ≈ programme



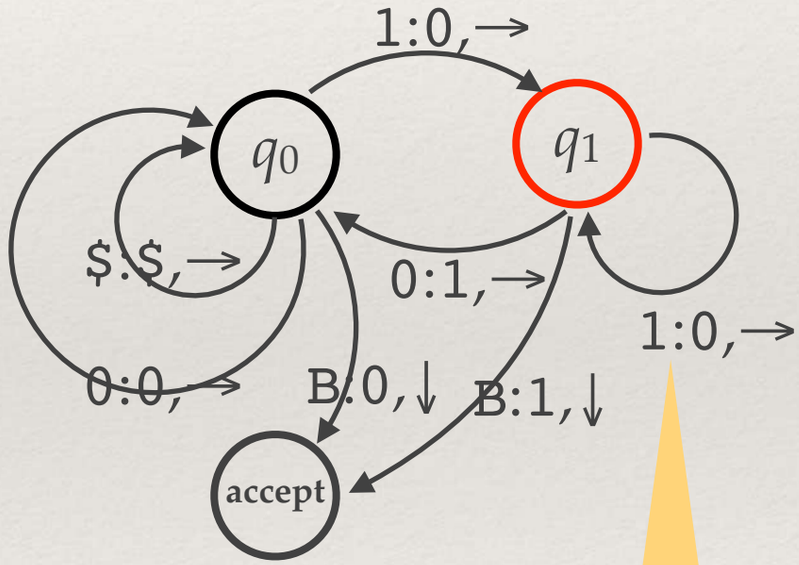
Le contrôle nous dit:
écrire 0 et
aller à droite (→),
passer en q_1

Machines de Turing

- Une **bande** (ou ruban)
  ≈ mémoire d'un ordinateur

- Une **tête** de lecture/écriture, voyageant d'au plus un case à chaque étape de calcul

- Un **contrôle** fini
  ≈ programme



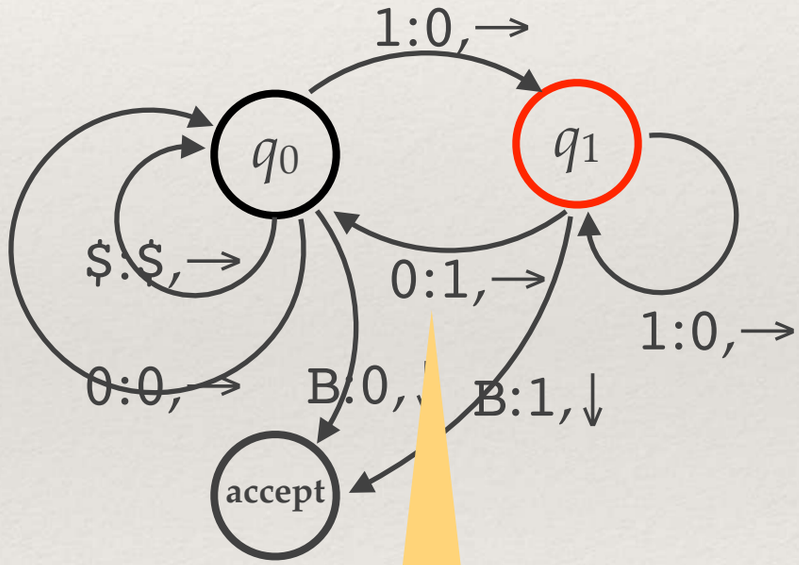
Le contrôle nous dit:
écrire 0 et
aller à droite (→),
rester en q_1

Machines de Turing

- Une **bande** (ou ruban)
  ≈ mémoire d'un ordinateur

- Une **tête** de lecture/écriture, voyageant d'au plus un case à chaque étape de calcul

- Un **contrôle** fini
  ≈ programme



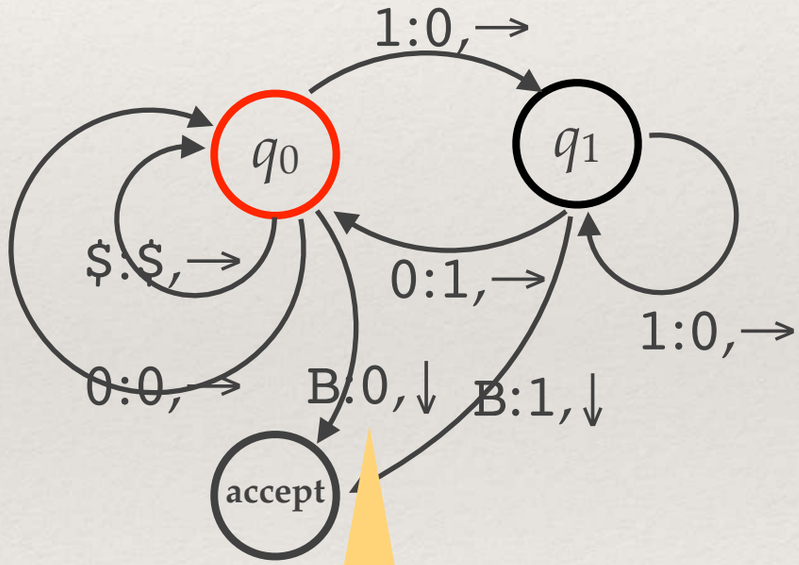
Le contrôle nous dit:
écrire 1 et
aller à droite (→),
passer en q_0

Machines de Turing

- Une **bande** (ou ruban)
  ≈ mémoire d'un ordinateur

- Une **tête** de lecture/écriture, voyageant d'au plus un case à chaque étape de calcul

- Un **contrôle** fini
  ≈ programme



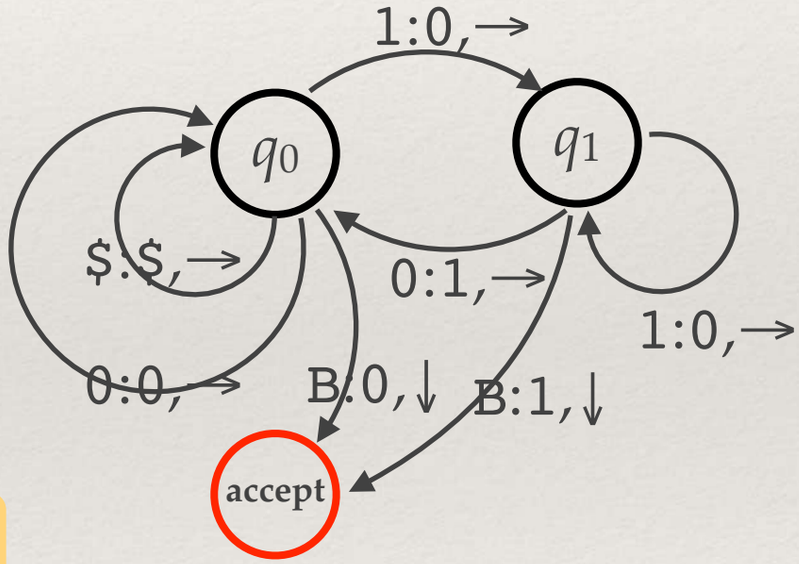
Le contrôle nous dit:
écrire 0 et
rester sur place (↓),
passer en **accept**

Machines de Turing

- Une **bande** (ou ruban)
  ≈ mémoire d'un ordinateur

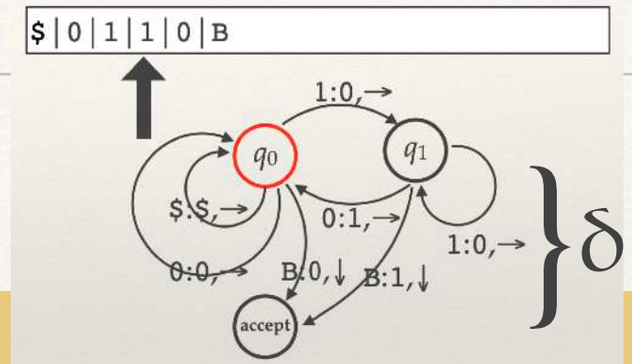
- Une **tête** de lecture/écriture, voyageant d'au plus un case à chaque étape de calcul

- Un **contrôle** fini
  ≈ programme



D'après vous, que calcule cette machine de Turing?

Machines de Turing: définition



- Formellement:

- **Définition.** Une *machine de Turing* est un sextuplet

$$M \stackrel{\text{def}}{=} (Q, q_0, \Sigma, \delta, \mathtt{B}, \mathtt{\$}) \quad \text{où:}$$

Q est un ensemble fini dit d'**états** [de contrôle]

$q_0 \in Q$
[l'état **initial**]

Σ = **alphabet** fini (card ≥ 3)

B, $ ∈ Σ lettres distinctes
— B = **blanc**
— $ = marqueur de **début**

Fonction **de transition** $\delta : Q \times \Sigma \to Q^+ \times \Sigma \times \mathrm{Dir}$
— $Q^+ \stackrel{\text{def}}{=} Q \uplus \{\textbf{accept}, \textbf{reject}\}$, **accept**≠**reject**
— $\mathrm{Dir} \stackrel{\text{def}}{=} \{\leftarrow, \downarrow, \rightarrow\}$ [gauche,immobile,droite]
— $\forall q \in Q$, $\delta(q,\$)$ de la forme $(q', \$, \rightarrow)$
— $\forall q \in Q$, $\forall a \in \Sigma - \{\$\}$, $\delta(q,a)$ de la forme (q', b, d) avec $b \neq \$$

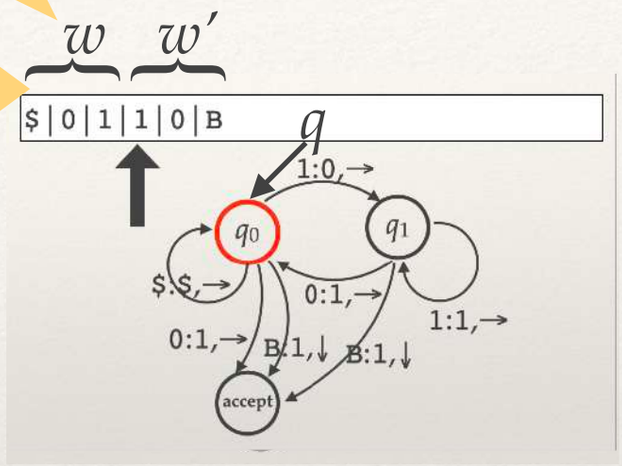
Dans quel but, d'après vous?

Machines de Turi figu

le mot à **gauche** de la tête

le mot à **droite** de la tête

ceci code donc aussi la **position** de la tête



- Une **configuration** de M est un triplet (w,q,w') où:
  — $ww' \in \$(\Sigma - \{\$\})^*$
  — $q \in Q^+$
- Sur l'**entrée** $x \in (\Sigma - \{\mathtt{B}, \$\})^*$, on démarre M dans la **configuration initiale** $(\varepsilon, q_0, \$x)$
- Configurations **finales**: (w,q,w') où $q \in \{\textbf{accept}, \textbf{reject}\}$

Définition. Une *machine de Turing* est un sextuplet
$M \stackrel{\text{def}}{=} (Q, q_0, \Sigma, \delta, \mathtt{B}, \$)$ où:

Q est un ensemble fini dit d'**états** [de contrôle]

$q_0 \in Q$ [l'état **initial**]

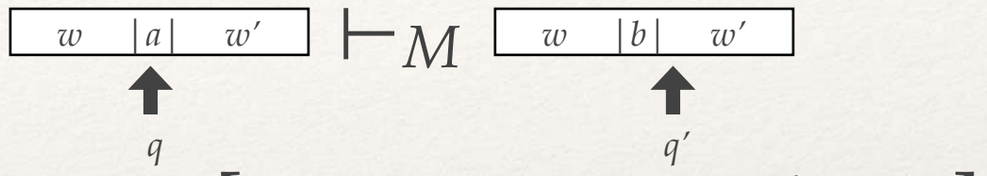
Σ = **alphabet** fini (card ≥ 3)

B, $ ∈ Σ lettres distinctes
— B = **blanc**
— $ = marqueur de **début**

Fonction **de transition** $\delta : Q \times \Sigma \to Q^+ \times \Sigma \times$ Di
— $Q^+ \stackrel{\text{def}}{=} Q \uplus$ {**accept**, **reject**}, **accept**≠**reject**
— Dir $\stackrel{\text{def}}{=}$ {←,↓,→} [gauche,immobile,droite]
— $\forall q \in Q$, $\delta(q,\$)$ de la forme $(q',\$,\to)$ ou $(q',\$,\downarrow)$
— $\forall q \in Q$, $\forall a \in \Sigma - \{\$\}$, $\delta(q,a)$ de la forme (q',b,d) avec $b \neq \$$

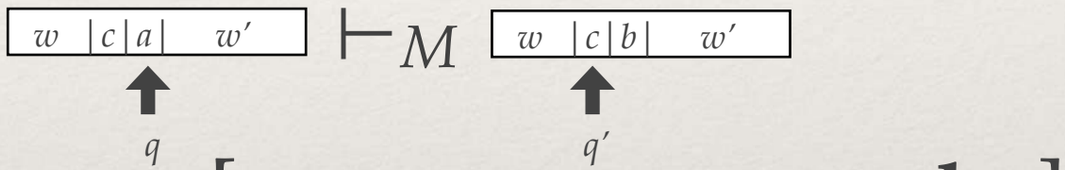
Machines de Turing: exécution (1/2)

- $(w,q,aw') \vdash_M (wb,q',w')$
  si $\delta(q,a) = (q',b,\rightarrow)$



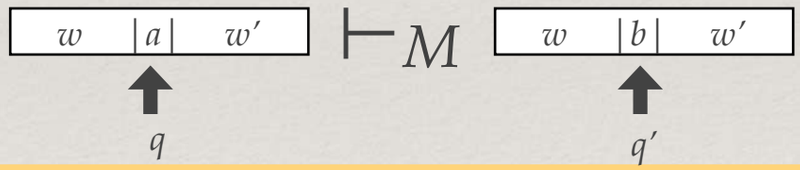
[mouvement **droit**]

- $(wc,q,aw') \vdash_M (w,q',cbw')$
  si $\delta(q,a) = (q',b,\leftarrow)$



[mouvement **gauche**]

- $(w,q,aw') \vdash_M (w,q',bw')$
  si $\delta(q,a) = (q',b,\downarrow)$

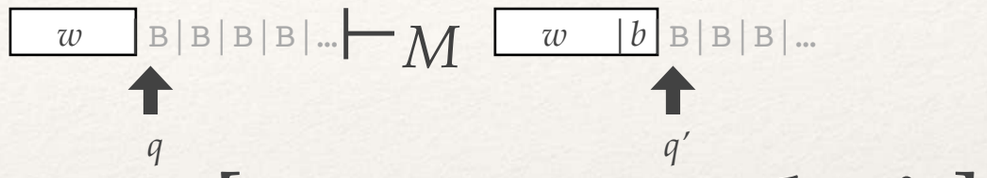


- … mais il en manque
  (aussi dans le poly):
  on peut « déborder du mot
  par la droite »

On rappelle:
— $\forall q \in Q, \exists q' \in Q, \delta(q,\$) = (q',\$,\rightarrow)$
— $wcaw' \in \$(\Sigma - \{\$\})^*$
Donc on ne peut jamais
« déborder du mot par la gauche »

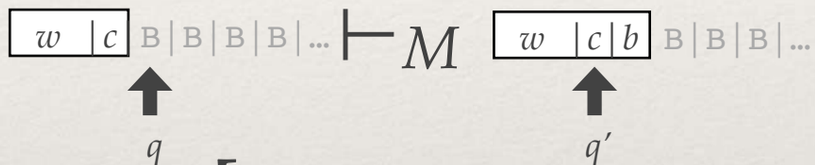
Machines de Turing: exécution (2/2)

- $(w,q,\varepsilon) \vdash_M (wb,q',\varepsilon)$
  si $\delta(q,\mathtt{B}) = (q',b,\rightarrow)$



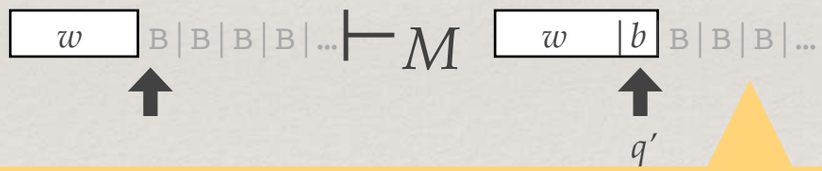
[mouvement **droit**]

- $(wc,q,\varepsilon) \vdash_M (w,q',cb)$
  si $\delta(q,\mathtt{B}) = (q',b,\leftarrow)$



[mouvement **gauche**]

- $(w,q,\varepsilon) \vdash_M (w,q',b)$
  si $\delta(q,\mathtt{B}) = (q',b,\downarrow)$



Le **truc**: on fait comme si
la bande s'étendait à droite
par une infinité de B
(ce truc n'est **pas** une définition formelle!)

Calculs

- Un **calcul** de M est une suite (finie ou infinie)

  $\gamma_0 \vdash_M \gamma_1 \vdash_M \ldots \vdash_M \gamma_n \vdash_M \ldots$ où:

  — chaque γ_n est une configuration

  — γ_0 est une configuration **initiale** $(\varepsilon, q_0, \$x)$

- Un calcul de longueur maximale est:

  — soit infini: M ne **s'arrête pas** sur l'entrée x

  — soit fini: $\gamma_0 \vdash_M \gamma_1 \vdash_M \ldots \vdash_M \gamma_n$

  où γ_n est une **configuration finale**

  (**acceptante** si d'état de contrôle **accept**, **rejetante** sinon)

Remarques

- Etant donnée x, il existe un unique calcul maximal

  $$\gamma_0 \vdash_M \gamma_1 \vdash_M \ldots \vdash_M \gamma_n \vdash_M \ldots \text{ avec } \gamma_0 = (\varepsilon, q_0, \$x)$$

  On dit que M est **déterministe**

  (Il existe des machines non déterministes: plus tard)

- On s'autorisera à définir des δ **partielles**
  si $\delta(q,a)$ pas défini, on conviendra que $\delta(q,a) = (\textbf{reject}, a, \downarrow)$

- Il existe des machines à **plusieurs bandes**: plus tard

- On distingue parfois alphabet des entrées x et alphabet de travail (utilisé sur la bande); ici l'**alphabet des entrées** est $\Gamma \stackrel{\text{def}}{=} \Sigma - \{\mathtt{B}, \mathtt{\$}\}$

Fonctions calculables, langages décidables

Que calcule une machine de Turing?

- Pour une machine de Turing M, pour tout $x \in \Gamma^*$, $[\Gamma \stackrel{\text{def}}{=} \Sigma - \{\mathtt{B}, \mathtt{\$}\}]$

  — si M **accepte** l'entrée x, i.e.,

  si $\gamma_0 = (\varepsilon, q_0, \mathtt{\$}x) \vdash_M \gamma_1 \vdash_M \ldots \vdash_M \gamma_n = (w, \mathbf{accept}, w')$,

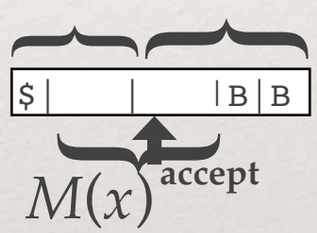
  on a ww' de la forme $\mathtt{\$}w''$, $w'' \in (\Sigma - \{\mathtt{\$}\})^*$

  et l'on pose:

  $M(x) \stackrel{\text{def}}{=}$ le plus grand préfixe de w'' sans $\mathtt{B}$

  — sinon $M(x)$ est indéfini

  (que le calcul ne termine pas ou rejette)



- Ceci définit une fonction (partielle) de Γ^* vers Γ^*

Fonctions calculables

- Une fonction (partielle) est **semi-calculable** ssi elle est de la forme $x \mapsto M(x)$ pour (au moins) une machine de Turing M
- Elle est **calculable** ssi de plus, on peut demander que M s'arrête sur toute entrée x
  (son domaine définition est l'ensemble des x acceptées par M; sur les autres, M **rejette**)
- Toute fonction calculable est semi-calculable
  (on verra que la réciproque est fausse)
- **Fait.** Une application (=fonction **totale**) est calculable ssi semi-calculable

définition légèrement différente, mais équivalente, dans le poly

Fonctions calculables

- On utilise de conventions de codage pour définir les fonctions calculables sur d'autres domaines:
- Les entiers naturels codés en binaire sur {0,1}*

  [Le poly dit « 0+1(0+1)* », mais pourquoi coder 0 par le mot 0, alors qu'on dispose de ε?]

  — lsb first: 182 = 01101101 [$0.2^0+1.2^1+1.2^2+\ldots$]

  — msb first: 182 = 10110110 [$1.2^7+0.2^6+1.2^5+\ldots$]

  — ou en unaire [rare]

- Les couples (x,y) par $x\#y$,

  où # lettre hors de l'alphabet de x, y

  [ou par '(' x ',' y ')']

Addition: sketch (1/2)

- Entrée: $x\#y$, x et y en binaire, lsb first (182 = 01101101 $[0.2^0+1.2^1+1.2^2+\ldots]$)
- On souhaite émettre en sortie: z, où $z \stackrel{\text{def}}{=} x+y$
- L'algorithme, abstraitement:

$\texttt{add\_no\_carry}(0x,0y) \stackrel{\text{def}}{=} 0\ \texttt{add\_no\_carry}(x,y)$
$\texttt{add\_no\_carry}(0x,1y) \stackrel{\text{def}}{=} 1\ \texttt{add\_no\_carry}(x,y)$
$\texttt{add\_no\_carry}(1x,0y) \stackrel{\text{def}}{=} 1\ \texttt{add\_no\_carry}(x,y)$
$\texttt{add\_no\_carry}(1x,1y) \stackrel{\text{def}}{=} 0\ \texttt{add\_carry}(x,y)$

$\texttt{add\_no\_carry}(\varepsilon,y) \stackrel{\text{def}}{=} y$
$\texttt{add\_no\_carry}(x,\varepsilon) \stackrel{\text{def}}{=} x$

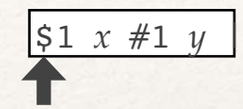
$\texttt{add\_carry}(0x,0y) \stackrel{\text{def}}{=} 1\ \texttt{add\_no\_carry}(x,y)$
$\texttt{add\_carry}(0x,1y) \stackrel{\text{def}}{=} 0\ \texttt{add\_carry}(x,y)$
$\texttt{add\_carry}(1x,0y) \stackrel{\text{def}}{=} 0\ \texttt{add\_carry}(x,y)$
$\texttt{add\_carry}(1x,1y) \stackrel{\text{def}}{=} 1\ \texttt{add\_carry}(x,y)$

$\texttt{add\_carry}(\varepsilon,\varepsilon) \stackrel{\text{def}}{=} 1$
$\texttt{add\_carry}(\varepsilon,0y) \stackrel{\text{def}}{=} 1y$
$\texttt{add\_carry}(\varepsilon,1y) \stackrel{\text{def}}{=} 0\ \texttt{add\_carry}(\varepsilon,y)$
$\texttt{add\_carry}(0x,\varepsilon) \stackrel{\text{def}}{=} 1x$
$\texttt{add\_carry}(1x,\varepsilon) \stackrel{\text{def}}{=} 0\ \texttt{add\_carry}(x,\varepsilon)$

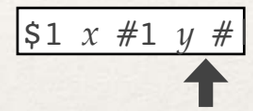
On va garder dans l'état l'information `carry`/`no_carry`… mais on va devoir garder bien davantage!

Addition: sketch (2/2)

Pour simplifier, je vais supposer que x et y ont la même longueur

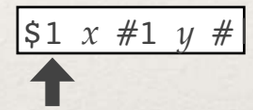


1. On va d'abord aller tout à droite, écrire un #



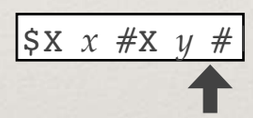
$\delta(q_1,a) \stackrel{\text{def}}{=} (q_1,a,\rightarrow)$ si $a \neq$ B
$\delta(q_1,\texttt{B}) \stackrel{\text{def}}{=} (q_2,\#,\leftarrow)$

2. et revenir tout à gauche, après le $ [en fait, le préfixe $X*]
[si on y trouve un #, on a fini: aller en 5]



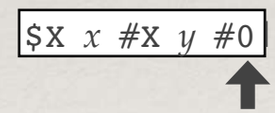
$\delta(q_2,a) \stackrel{\text{def}}{=} (q_2,a,\leftarrow)$ si $a \neq$ $
$\delta(q_2,\$) \stackrel{\text{def}}{=} (q_3,\$,\rightarrow)$
$\delta(q_3,\texttt{X}) \stackrel{\text{def}}{=} (q_3,\texttt{X},\rightarrow)$

3. On reparcourt de gauche à droite, collectant les lsb des arguments (1 et 1, ici), et les remplaçant par des X;

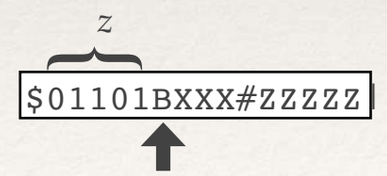


$\delta(q_3,a) \stackrel{\text{def}}{=} (q_{4a},\texttt{X},\rightarrow)$ si $a \neq$ X,#
$\delta(q_{4a},b) \stackrel{\text{def}}{=} (q_{4a},b,\rightarrow)$ si $b \neq$ X,#
$\delta(q_{4a},\#) \stackrel{\text{def}}{=} (q_{5a},\#,\rightarrow)$
$\delta(q_{5a},\texttt{X}) \stackrel{\text{def}}{=} (q_{5a},\texttt{X},\rightarrow)$
$\delta(q_{5a},b) \stackrel{\text{def}}{=} (q_{6ab},\texttt{X},\rightarrow)$ si $a \neq$ X,#
…

4. On écrit la somme des lsbs (0, ici), et on se souvient de la retenue; revenir en 2



5. Nettoyage: par des allers-retours, recopier z bit à bit au début de la bande, et écrire un B juste à sa droite



On s'en souvient dans l'**état**
Q = {*phases d'exploration*} ×
{0,1} (lsb de x) × {0,1} (lsb de y) ×
{carry, no_carry} (retenue)

Autres opérations

- Les **opérations arithmétiques** usuelles (+, –, ×, /, mod) sont calculables; aussi ^ (qui est partielle)
- La **composition** g o f de deux fonctions calculables
  est calculable
  [calculer f sur l'entrée x, puis calculer g sur le résultat…
  mais faites attention au traitement des `B`, qui engendrent une infinité de difficultés
  attention: la preuve de la Proposition 6.3.2 du poly est erronée
  nous admettrons ce résultat pour l'instant, et le prouverons comme conséquence d'autres résultats plus tard]
- La **composition** g o f de deux fonctions semi-calculables
  est semi-calculable
  [pareil: g o f définie en x si f l'est, et si g est définie en $f(x)$]
- **Conditionnelles**, **boucles** while, etc.
  […plus tard]

Langages récursivement énumérables

- Un langage $L \subseteq \Gamma^*$ est **récursivement énumérable** (=**r.e.**, = **semi-décidable**) ssi il existe une machine de Turing M telle que pour toute entrée x,

  $x \in L$ ssi M **accepte** l'entrée x

- Si $x \notin L$, M peut rejeter, ou ne pas terminer

- Remarque: on ignore le contenu de la bande en sortie

- Autrement dit, L est r.e.:
  — ssi il existe une fonction partielle **semi-calculable** de domaine L
  — ssi la fonction partielle de domaine L **constante** de valeur ε est **semi-calculable** [pourquoi?]

On notera $L(M)$ le langage des mots **acceptés** par M.
Les langages de la forme $L(M)$ sont donc exactement les langages r.e.

Langages récursifs (décidables)

- Un langage $L \subseteq \Gamma^*$ est **récursif** (=**décidable**) ssi il existe une machine de Turing M telle que pour toute entrée x,

  $x \in L$ ssi M **accepte** l'entrée x

  et $x \notin L$ ssi M **rejette** l'entrée x

- Autrement dit, M **termine** sur toute entrée

- L est décidable:

  — ssi il existe une fonction **calculable** de domaine L

  — ssi la fonction partielle de domaine L **constante** de valeur ε est **calculable**

Récursif vs. r.e.

- Tout langage récursif est r.e.
- On verra qu'il existe des langages r.e. **non récursifs**
  [le problème de l'arrêt, A. Turing 1936]
- L'**Entscheidungsproblem**,
  ENTREE: une formule *F* de la logique du premier ordre
  QUESTION: *F* a-t-elle une démonstration?
  est r.e.:
  — énumérer tous les mots;
  — pour chacun, tester si c'est une preuve de *F*;
  — si oui, **accept**er, sinon continuer
- mais **pas récursif**! [admis, au moins pour l'instant]

Tous les langages ne sont pas r.e.

- Raison 1: il n'y a qu'un nombre **dénombrable** de machines de Turing (sur un alphabet donné),
  donc un nombre dénombrable de langages r.e.

- Raison 2 (constructive). Enumérons les machines de Turing: $M_0, M_1, \ldots$ et les mots: $w_0, w_1, \ldots$ (sans doublon)

  - **Fait:** $L \stackrel{\text{def}}{=} \{w_i \mid w_i \notin L(M_i)\}$ n'est pas r.e.

  - Sinon, $L = L(M_i)$ pour un certain i.

  - Alors $w_i \in L$ ssi $w_i \notin L(M_i)$ [déf. de L]
    ssi $w_i \notin L$ [car $L = L(M_i)$]

Argument **diagonal**

La classe des langages r.e.

- … est close par **image inverse** semi-calculable:
  si L r.e. et f semi-calculable, alors
  $$f^{-1}(L) \stackrel{\text{def}}{=} \{w \mid f(w) \text{ défini et dans } L\}$$
  est r.e.
  [car L est le domaine d'une fonction semi-calculable g, et $f^{-1}(L)$ est celui de $g \circ f$]

- est close par **intersections finies, unions finies**
  [intersections: on peut écrire des conditionnelles… pas encore vu;
  unions: plus difficile (dovetailing)]

- mais **pas** par **négation** [on verra plus tard pourquoi]

- contient la classe des langages récursifs

La classe des langages récursifs

- … est close par **image inverse calculable**:
  si L décidable et f **calculable**, alors
  $$f^{-1}(L) \stackrel{\text{def}}{=} \{w \mid f(w) \text{ défini et dans } L\}$$
  est décidable.
  [car L est le domaine d'une fonction calculable g, et $f^{-1}(L)$ est celui de $g \circ f$]

- est close par **intersections finies, unions finies**
  [car on peut écrire des conditionnelles… pas encore vu]

- et aussi par **négation** […on peut écrire des conditionnelles (!)]

- contient les langages rationnels, et même les algébriques

Variantes

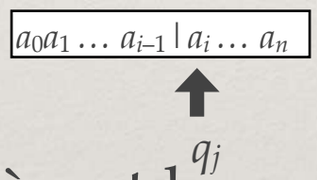
Variantes

- On peut se demander:
  - de combien d'états de contrôle on a besoin
  - d'alphabets de quelle taille
  - etc.
- Quelques exemples dans la suite, mais…
- la notion fondamentale est celle de machine de Turing **à plusieurs bandes**

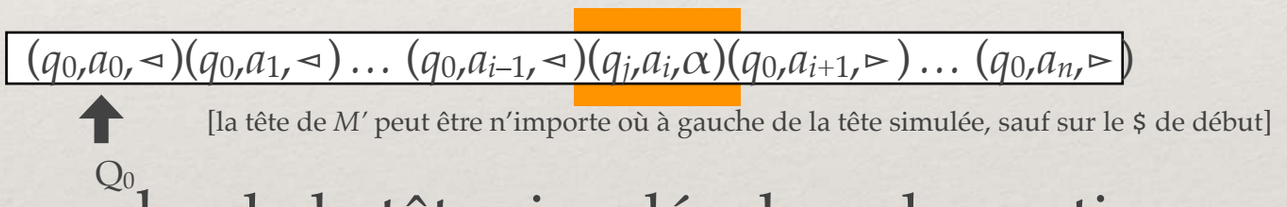
Réduire le nombre d'états

- **Théorème.** [6.4.1 dans le poly] Pour toute mT M,
  il existe une mT M' **à 2 états** telle que $L(M')=L(M)$
  [et qui rejette les mêmes entrées que M aussi]

- Démonstration (idée). On étend l'alphabet.
  On numérote les états $q_1, \ldots, q_m$ + un pseudo-état q_0.
  On recode toute configuration



  en:



  [la tête de M' peut être n'importe où à gauche de la tête simulée, sauf sur le $ de début]

- Les $\triangleleft$ repèrent la portion de bande à gauche de la tête simulée, les $\triangleright$ la portion à droite; α est un marqueur, initialement $\triangleright$ (il y en aura d'autres)

- Deux états dans la nouvelle machine M': Q_0, Q_1.

2 états: simulation d'une transition

- Exemple: simulation de $\delta(q_j,a_i) = (q_k,b,\rightarrow)$
- Aller à droite jusqu'à rencontrer ▻
  [en Q_0, on sera toujours à gauche de la tête simulée]
- Passer en mode « simulation de changement d'état » (Q_1)
- on décrémente à partir de k–1
  et on incrémente son voisin de droite
  partant de 0, en alternance (le **truc**!)
- on nettoie, en repassant en mode
  « recherche de tête simulée » (Q_0),
  la tête ne bouge pas

2 états: initialisation

- Au démarrage, on doit convertir la bande

  $\$a_1 \ldots a_n \,\mathtt{B}$

  en: $\$(q_1,\underline{\$},\triangleright)(q_0,a_1,\triangleright)(q_0,a_2,\triangleright)\ldots(q_0,a_n,\triangleright)$

  où $\underline{\$}$ est une nouvelle lettre se comportant comme $\$$

  ($\delta(q,\underline{\$}) \stackrel{\text{def}}{=} (q',\underline{\$},\rightarrow)$ pour tous états q, q' tels que $\delta(q,\underline{\$}) \stackrel{\text{def}}{=} (q',\underline{\$},\rightarrow)$)

- Ce qui est facile, c'est de convertir la bande initiale en:

  $\$(\trianglerighteq,a_1,-)(\trianglerighteq,a_2,-)\ldots(\trianglerighteq,a_n,-)\,\mathtt{B}$ (–, $\trianglelefteq$, $\trianglerighteq$ nouveaux marqueurs)

- puis $\$(\trianglerighteq,a_1,\mathtt{0})(\trianglerighteq,a_2,\mathtt{0})\ldots(\trianglerighteq,a_n,\mathtt{0})(\trianglerighteq,\mathtt{B},\mathtt{0})$ ($\mathtt{0}=\underline{\$}$; voir plus tard)

  par un aller-retour sur la bande, en restant dans l'état Q_1;

  (on passera dans l'état Q_0 pour la suite)

- La difficulté est d'**insérer** $(q_1,\underline{\$},\triangleright)$ (et changer les $\mathtt{0}$ en $\triangleright$, etc.)

  On utilise une astuce similaire, en numérotant les **lettres** $\mathtt{0}$, $\mathtt{1}$,…, k

Insertion à l'initialisation (1/2)

On suppose ici $\underline{\$}=0$, $a_1=3$, $a_2=1$, $a_3=6$)

$\$(\trianglerighteq,3,0)(\trianglerighteq,1,0)(\trianglerighteq,6,0)\ldots$ (Q_0)

$\$(\trianglerighteq,3,0)(\trianglerighteq,1,0)(\trianglerighteq,6,0)\ldots$ (Q_0)

[écriture de $\underline{\$}=0$ +]échange

$\$(\trianglelefteq,0,3)(\trianglerighteq,1,0)(\trianglerighteq,6,0)\ldots$ (Q_1)

$\$(\trianglelefteq,0,2)(\trianglerighteq,1,0)(\trianglerighteq,6,0)\ldots$ (Q_1)

Transfert de $a_1=3$, comme l'astuce précédente

$\$(\trianglelefteq,0,2)(\trianglerighteq,1,1)(\trianglerighteq,6,0)\ldots$ (Q_1)

$\$(\trianglelefteq,0,1)(\trianglerighteq,1,1)(\trianglerighteq,6,0)\ldots$ (Q_1)

$\$(\trianglelefteq,0,1)(\trianglerighteq,1,2)(\trianglerighteq,6,0)\ldots$ (Q_1)

$\$(\trianglelefteq,0,0)(\trianglerighteq,1,2)(\trianglerighteq,6,0)\ldots$ (Q_1)

$\$(\trianglelefteq,0,0)(\trianglerighteq,1,3)(\trianglerighteq,6,0)\ldots$ (Q_1)

$\$(q_1,0,\diamondsuit)(\trianglerighteq,1,3)(\trianglerighteq,6,0)\ldots$ (Q_0)

$\$(q_1,0,\diamondsuit)(\trianglelefteq,3,1)(\trianglerighteq,6,0)\ldots$ (Q_1)

$\$(q_1,0,\diamondsuit)(\trianglelefteq,3,0)(\trianglerighteq,6,0)\ldots$ (Q_1)

$\$(q_1,0,\diamondsuit)(\trianglelefteq,3,0)(\trianglerighteq,6,1)\ldots$ (Q_1)

$\$(q_1,0,\diamondsuit)(q_0,3,\diamondsuit)(\trianglerighteq,6,1)\ldots$ (Q_0)

…

$\$(q_1,0,\diamondsuit)(q_0,3,\diamondsuit)(q_0,1,\diamondsuit)(\trianglerighteq,a_4,6)\ldots$ (Q_0)

…

$\$(q_1,0,\diamondsuit)(q_0,3,\diamondsuit)(q_0,1,\diamondsuit)\ \ldots\ (q_0,a_n,\diamondsuit)$ B (Q_0)

$\delta(Q_0,\$) = (Q_0,\$,\rightarrow)$

$\delta(Q_0,(\trianglerighteq,m,n)) = (Q_1,(\trianglelefteq,n,m),\downarrow)$

$\delta(Q_1,(\trianglelefteq,m,n)) = (Q_1,(\trianglelefteq,m,n-1),\rightarrow)$ si $n\neq 0$

$\delta(Q_1,(\trianglerighteq,m,n)) = (Q_1,(\trianglerighteq,m,n+1),\leftarrow)$

$\delta(Q_1,(\trianglelefteq,\underline{\$},0)) = (Q_0,(q_1,\underline{\$},\diamondsuit),\rightarrow)$

$\delta(Q_1,(\trianglelefteq,m,0)) = (Q_0,(q_0,m,\diamondsuit),\rightarrow)$ si $m\neq\underline{\$}$ (i.e., $m\neq 0$)

Insertion à l'initialisation (2/2)

$(⊵,3,0)(⊵,1,0)(⊵,6,0)…

Q_0 …

$\$(q_1,0,\diamondsuit)(q_0,3,\diamond)(q_0,1,\diamond)\ \ldots\ (q_0,a_n,\diamond)$ B

Q_0

$\$(q_1,0,\diamondsuit)(q_0,3,\diamond)(q_0,1,\diamond)\ \ldots\ (q_0,a_n,\diamond)(q_0,\underline{B},\triangleright)$

Q_0

…

$\$(q_1,0,\triangleright)(q_0,3,\triangleright)(q_0,1,\triangleright)\ \ldots\ (q_0,a_n,\triangleright)(q_0,\underline{B},\triangleright)$

Q_0

(résumé)

On suppose ici $\underline{\$}=0$, $a_1=3$, $a_2=1$, $a_3=6$)

$\delta(Q_0,\$) = (Q_0,\$,\rightarrow)$

$\delta(Q_0,(⊵,m,n)) = (Q_1,(⊴,n,m),\downarrow)$

$\delta(Q_1,(⊴,m,n)) = (Q_1,(⊴,m,n–1),\rightarrow)$ si $n\neq 0$

$\delta(Q_1,(⊵,m,n)) = (Q_1,(⊵,m,n+1),\leftarrow)$

$\delta(Q_1,(⊴,\underline{\$},0)) = (Q_0,(q_1,\underline{\$},\diamondsuit),\rightarrow)$

$\delta(Q_1,(⊴,m,0)) = (Q_0,(q_0,m,\diamondsuit),\rightarrow)$ si $m\neq\underline{\$}$ (i.e., $m\neq 0$)

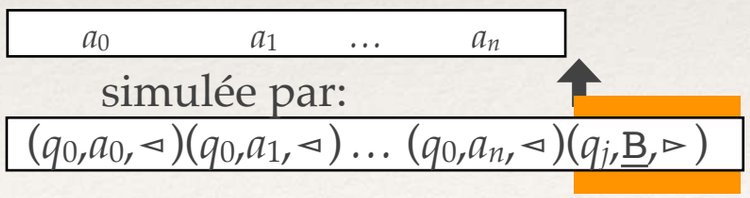
$\delta(Q_0,B)) = (Q_0,(q_0,\underline{B},\triangleright),\leftarrow)$

$\delta(Q_0,(q_0,a,\diamond))) = (Q_0,(q_0,a,\triangleright),\leftarrow)$

- Et on a fini l'initialisation. (Finalisation: pas la peine, on s'intéresse aux langages, bande de sortie ignorée.)

- **Note:** $\underline{B}$ est un marqueur de fin de bande (simulée)
  on en a besoin pour stocker
  l'état q_j si en fin de bande

  Exercice: simuler les transitions à droite sur $(q_j,\underline{B},\triangleright)$



Avec un seul état?

- Non, il existe des langages r.e. qui ne sont les langages d'aucune machine de Turing à **un état**, car:
- il existe des langages r.e. non récursifs [voir plus tard]
- tous les langages de mT à un état sont récursifs:

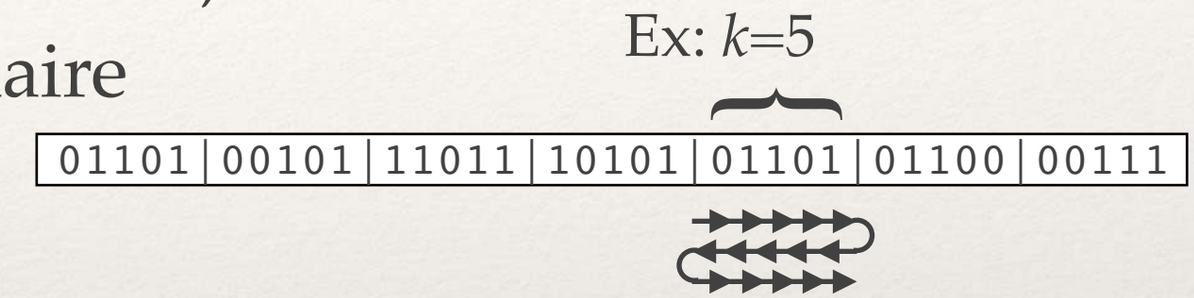
(1) Gabor T. Herman. *The halting problem of one state Turing machines with n-dimensional tape.* Mathematical Logic Quarterly, 14(7–12):185–191, 1968. https://onlinelibrary.wiley.com/doi/10.1002/malq.19680140706

(2) Yannick Saouter. *Halting problem for one-state Turing machines.* Rapport de recherche INRIA RR-2577, 1995. https://hal.inria.fr/inria-00074105/document

Réduire le nombre de lettres

- On peut descendre à **deux lettres** dans Γ,
  en écrivant toutes les lettres en binaire
  sur $k \stackrel{\text{def}}{=} \Theta(\log |\Sigma|)$ bits

Ex: k=5



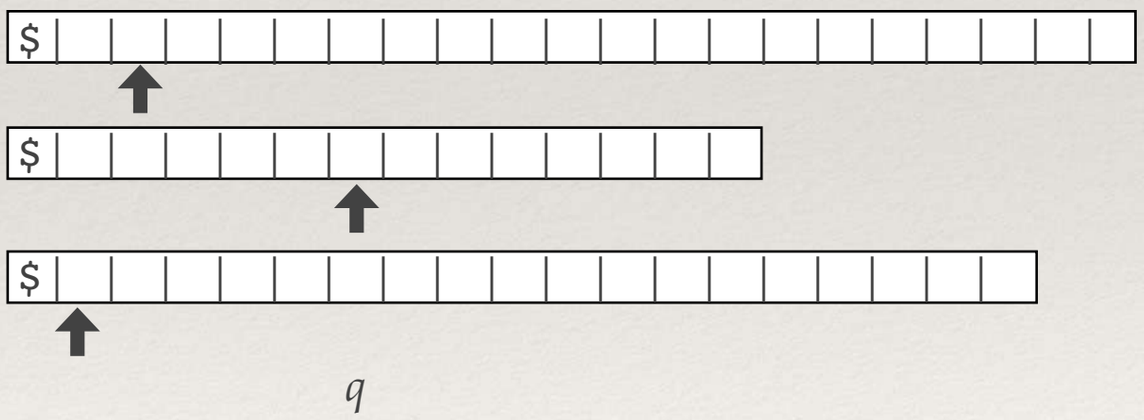
- Détails omis: pour simuler un mouvement à droite,
  on doit maintenant faire ***k* mouvements** à droite,
  en mémorisant les bits lus dans le contrôle

- … mais aussi k mouvements à gauche pour écrire la nouvelle lettre
  (sur k bits) et de nouveau k mouvements à droite

- Finalement, le langage reconnu n'est plus L,
  mais $\{\text{bin}(w) \mid w \in L\}$ [léger bug du poly]

Machines à plusieurs bandes

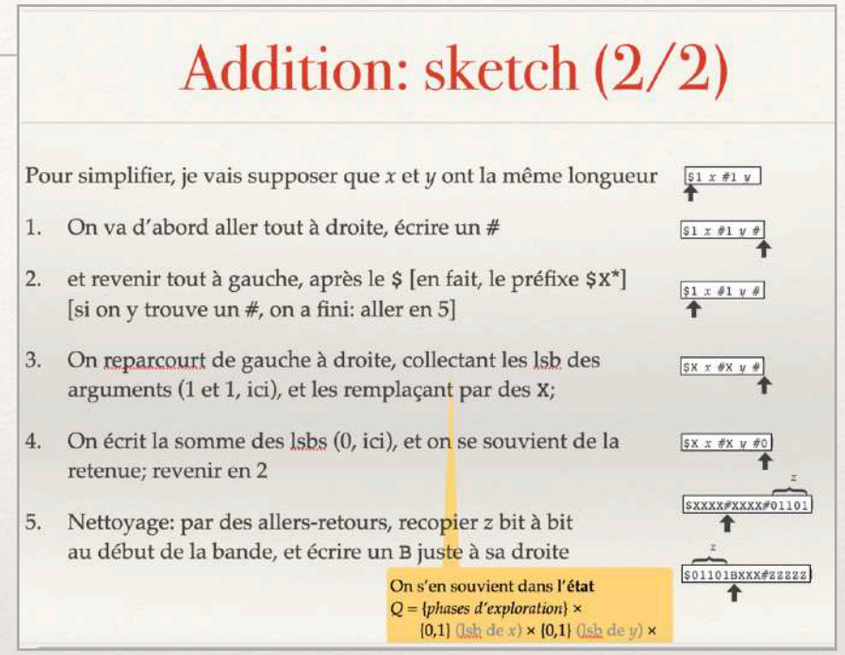
- La notion de machine de Turing à plusieurs bandes est bien plus importante:
  — plus facile d'écrire des mT à plusieurs bandes
  — **exactement** le même pouvoir expressif
- Définition formelle: voir définition 6.4.1 du poly
- Informellement: $k \geq 1$ bandes,
  k têtes indépendantes,
  mais toujours un seul état
- Initialement, toutes les bandes sauf la 1 (l'entrée) sont vides



- $\delta : Q \times \sum^k \rightarrow Q^+ \times \sum^k \times \mathrm{Dir}^k$ [contraintes: écrit $ uniquement en début de bandes]

Exemple: addition

- Reprenons l'exemple de l'addition en binaire
- On peut décrire **toute** une machine à deux bandes faisant la même chose:



	$\$,\$$	$0,\$$	$1,\$$	$\#,\$$	$0,B$	$1,B$	B,B	$0,0$	$0,1$	$1,0$	$1,1$	$B,0$	$B,1$	$B,\$$
q_0	q_0 $\$,\rightarrow$ $\$,\downarrow$	q_0 $0,\rightarrow$ $\$,\downarrow$	q_0 $1,\rightarrow$ $\$,\downarrow$	q_0 $B,\rightarrow$ $\$,\rightarrow$	q_0 $B,\rightarrow$ $0,\rightarrow$	q_0 $B,\rightarrow$ $1,\rightarrow$	q_1 $B,\leftarrow$ $B,\leftarrow$							
q_1	q_2 $\$,\rightarrow$ $\$,\rightarrow$	q_1 $0,\leftarrow$ $\$,\downarrow$	q_1 $1,\leftarrow$ $\$,\downarrow$					q_1 $0,\leftarrow$ $0,\leftarrow$	q_1 $0,\leftarrow$ $1,\leftarrow$	q_1 $1,\leftarrow$ $0,\leftarrow$	q_1 $1,\leftarrow$ $1,\leftarrow$	q_1 $B,\leftarrow$ $0,\leftarrow$	q_1 $B,\leftarrow$ $1,\leftarrow$	q_1 $B,\leftarrow$ $\$,\downarrow$
q_2					q_2 $0,\rightarrow$ $B,\downarrow$	q_2 $1,\rightarrow$ $B,\downarrow$		q_2 $0,\rightarrow$ $0,\rightarrow$	q_2 $0,\rightarrow$ $1,\rightarrow$	q_2 $1,\rightarrow$ $0,\rightarrow$	q_3 $0,\rightarrow$ $0,\rightarrow$	q_2 $0,\rightarrow$ $B,\rightarrow$	q_2 $1,\rightarrow$ $B,\rightarrow$	
q_3					q_2 $1,\rightarrow$ $B,\downarrow$	q_3 $0,\rightarrow$ $B,\downarrow$		q_2 $1,\rightarrow$ $0,\rightarrow$	q_3 $0,\rightarrow$ $1,\rightarrow$	q_3 $0,\rightarrow$ $0,\rightarrow$	q_3 $1,\rightarrow$ $0,\rightarrow$	q_2 $1,\rightarrow$ $B,\rightarrow$	q_3 $0,\rightarrow$ $B,\rightarrow$	

[Exemple 6.4.1 du poly]

La prochaine fois

La prochaine fois

- Machines I/O à k bandes de travail
- Equivalence avec les machines de Turing à une bande
- Complexité en temps, en espace