Inf 431 – Cours 13

Machines finies et infinies

jeanjacqueslevy.net

secrétariat de l'enseignement:
Catherine Bensoussan
cb@lix.polytechnique.fr
Aile 00, LIX,
01 69 33 34 67

www.enseignement.polytechnique.fr/informatique/IF

Plan

1. Automates finis
2. Les 3 théorèmes
3. Machines de Turing
4. Diagonalisation
5. Langages non récursivement énumérables

M. L. Minsky, *Computation : Finite and Infinite Machines*. Prentice Hall, 1967.

D. Kozen, *Automata and Computability*. Springer, 1997.

J. Hopcroft, R. Motwani, J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2000.

Exemples d'automates finis

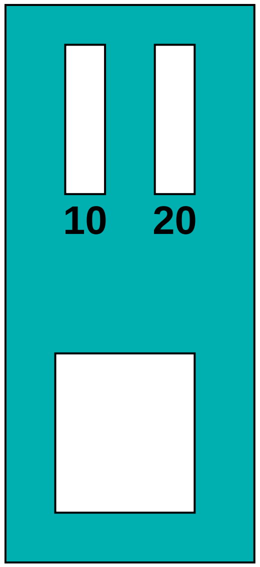
- Mécanismes à mémoire finie
- Protocoles dans les réseaux
- *Model checking* dans la concurrence
- Automates cellulaires
- Systèmes réactifs
- Controleurs dans les circuits
- Analyseurs lexicaux

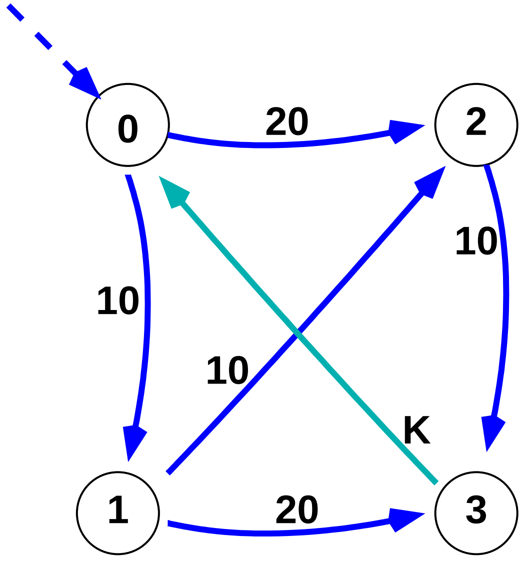
Mécanisme fini (1/2)

Distributeur de boissons :

Deux fentes pour pièces de 0,10€, 0,20€.

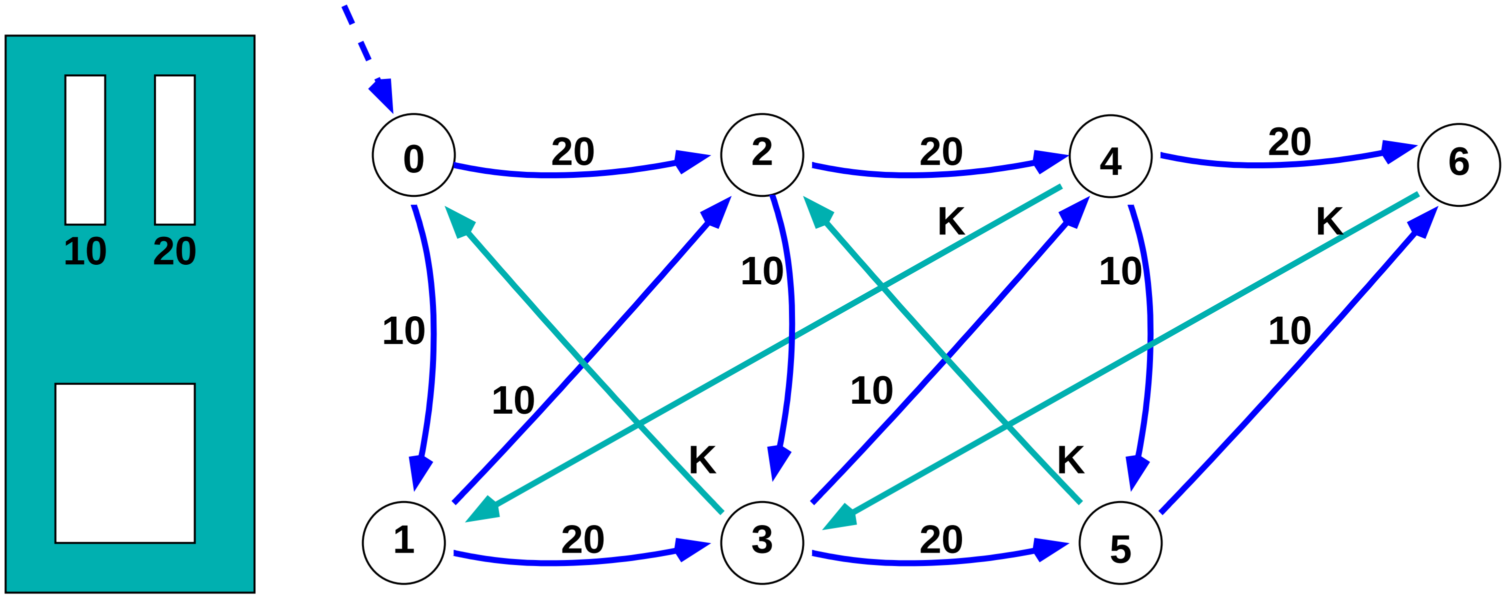
Café si on met 0,30€.





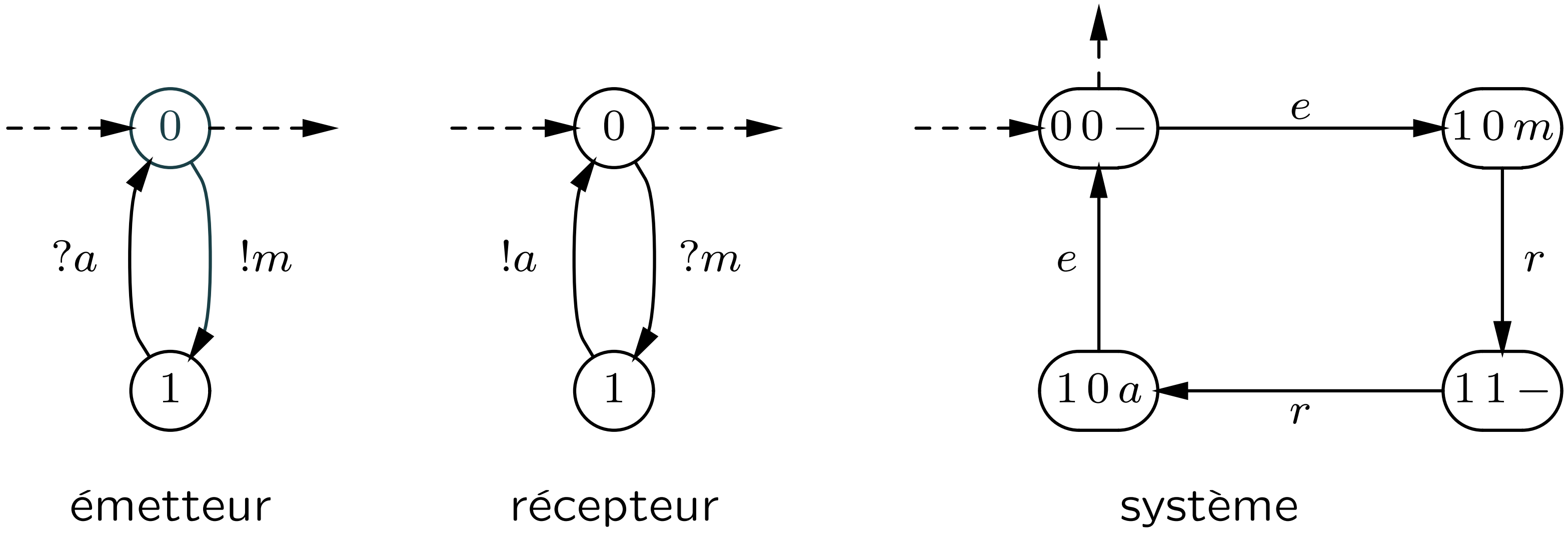
Mécanisme fini (2/2)

Machine avec une mémoire tampon (*buffer*) de taille 2,
2 cafés avec 0,60€.



Contrôle de flux

Un récepteur ne doit pas être submergé par les messages d'un émetteur.



$!m$ envoi du message m

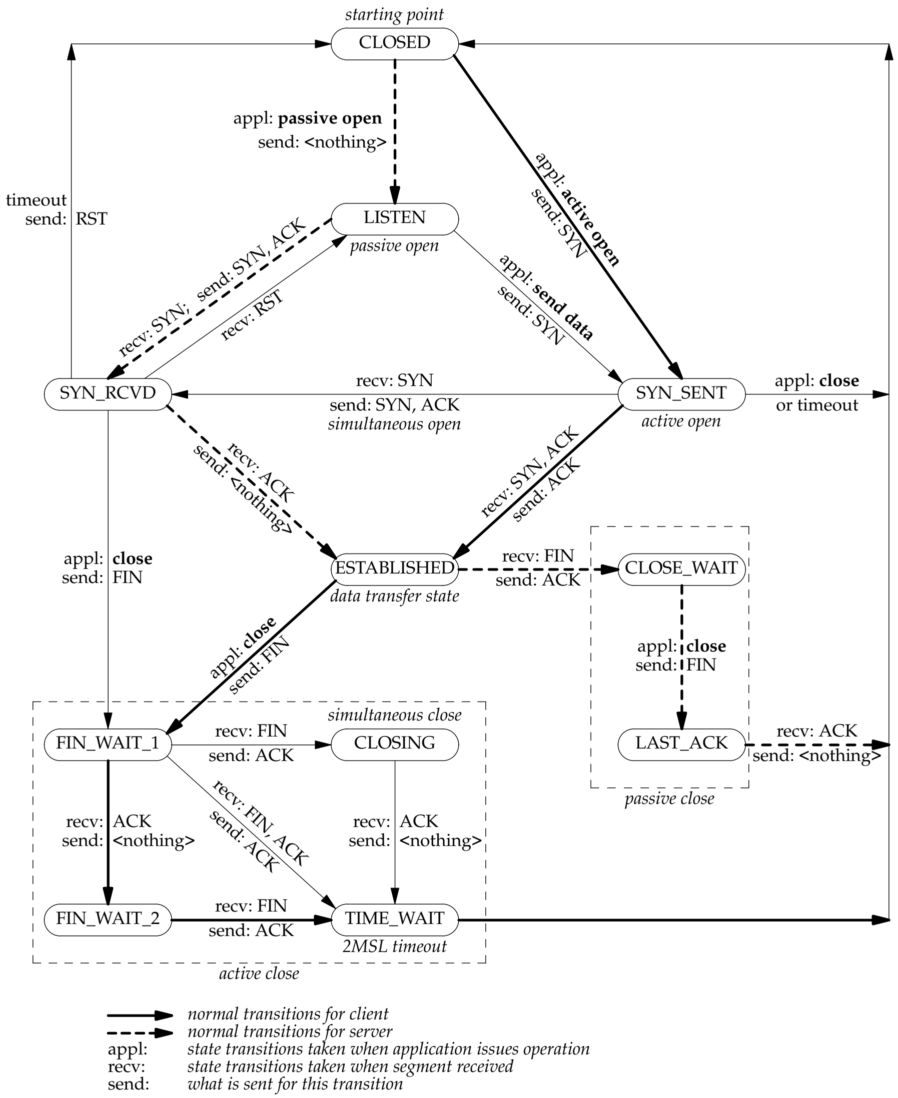
$?a$ lecture accusé de réception

$?m$ lecture du message m

$!a$ envoi de l'accusé de réception

Le problème se complique en cas de pertes de messages.
$\Rightarrow$ protocole du bit alterné.

Protocoles réseau

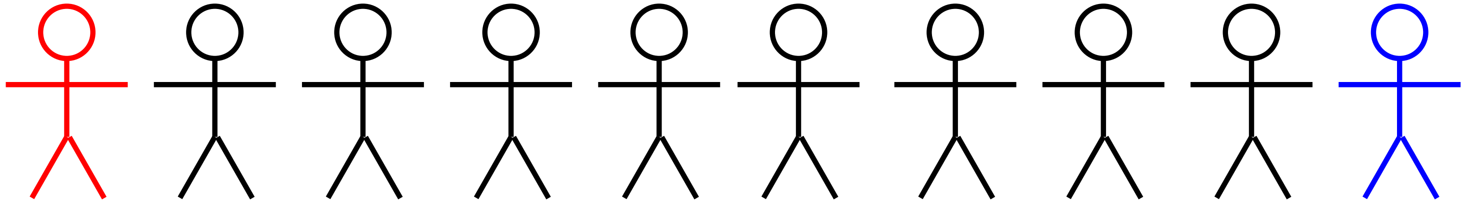


Les protocoles TCP/IP (*Transmission Control Protocol/Internet Protocol*) sont les protocoles de base pour la transmission en série de paquets sur l'internet.

Certains protocoles peuvent générer beaucoup d'états [1, 2, 3].

Automate cellulaire

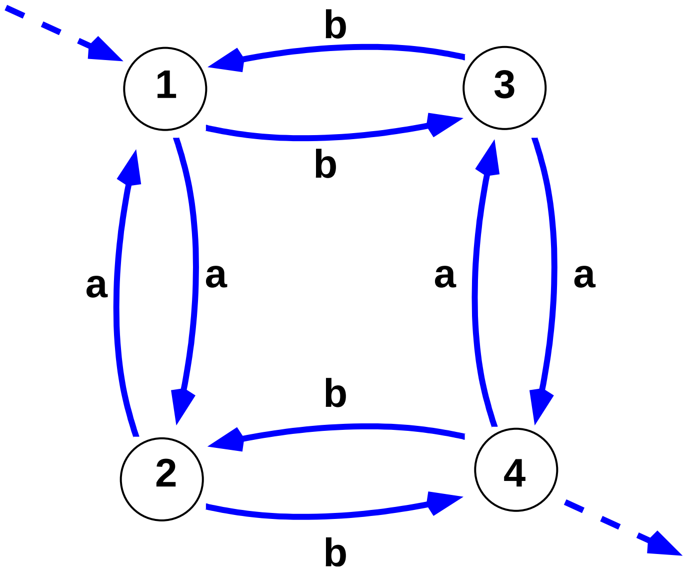
- Tempête sur le plateau : on ne voit rien, on n'entend rien.
- Le général convoque $n-1$ soldats, les aligne sur son coté, et veut les faire tirer tous en même temps.
- Le général et les soldats sont des machines synchrones réagissant tous en phase en ne tenant compte que des états de leurs voisins de gauche ou de droite.
- 3 types de machines : le général, les soldats, le soldat en bout de chaîne.



Exercice 1 Trouver une solution, avec un nombre fini d'états (indépendant de n) pour chaque machine.

Langage reconnu par un automate fini

Mots contenant un nombre impair de a et un nombre impair de b.



Automate fini (1/3)

Un automate fini A est un quintuplet $A = (Q, \Sigma, \delta, q_0, F)$ où :

- Σ est un alphabet donné,
- Q est un ensemble fini d'états,
- $q_0 \in Q$ est l'état initial,
- $F \subset Q$ est l'ensemble des états finaux,
- $\delta : Q \times \Sigma \to Q$ est la fonction de transition

On peut étendre δ sur $Q \times \Sigma^* \to Q$ comme suit :

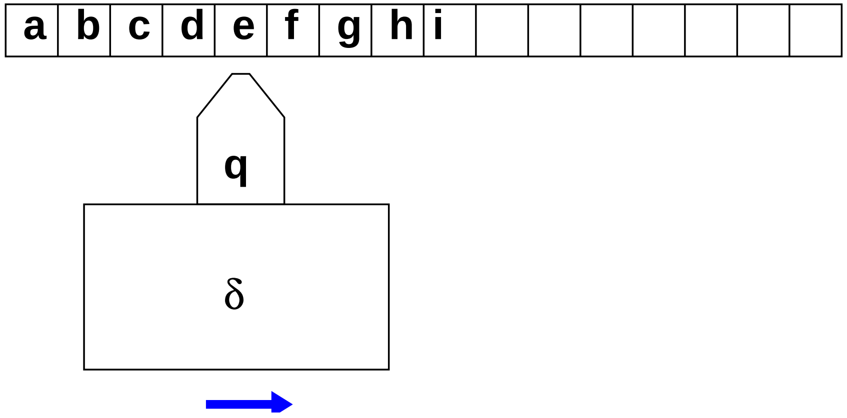
$$\delta(q, \epsilon) = q$$
$$\delta(q, aw) = \delta(\delta(q, a), w)$$

Le langage reconnu par l'automate A est :

$$T(A) = \{w \mid \delta(q_0, w) \in F\}$$

Automate fini (2/3)

Graphiquement :



- La bande de lecture n'est pas modifiable (*read-only*)
- La tête de lecture avance d'une case vers la droite à chaque transition.
- L'état q évolue vers l'état $q' = \delta(q, e)$ si le caractère e est sous la tête de lecture.

Automate fini (3/3)

Plusieurs représentations possibles :

- En dur, dans le contrôle fini du programme

  ```
  int c = in.read(); int q = q0;
  switch (q) {
    case 0: if (c == 'a') ... else if (c == 'b') ...; break;
    case 1: if (c == 'b') ... else if (c == 'c') ... ; break;
    ...
    case 10: if ... break;
  }
  ```

- matrice $|Q| \times |\Sigma|$ pour représenter la fonction de transition
- vecteur de $|Q|$ listes d'association (c', q')
  (graphe dont les sommets sont les états et les arcs sont les transitions étiquetées par les lettres)

Automate fini non déterministe (1/2)

- A chaque état, l'automate choisit entre plusieurs transitions. Formellement, $\delta : Q \times \Sigma \rightarrow 2^Q$

- On étend δ comme suit

$$\delta(q, \epsilon) = \{q\} \qquad \delta(q, aw) = \bigcup_{q' \in \delta(q,a)} \delta(q', w)$$

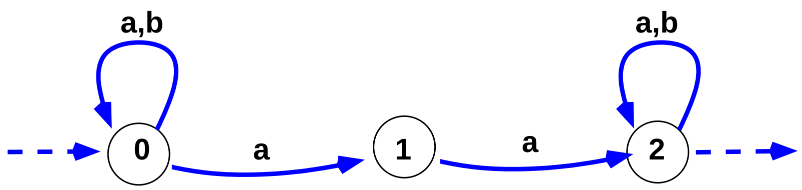
- Le langage reconnu par l'automate A est :

$$T(A) = \{w \mid \delta(q_0, w) \cap F \neq \emptyset\}$$

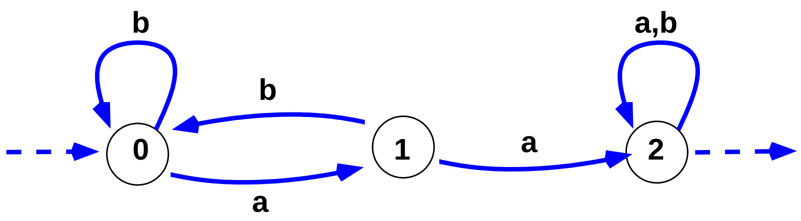
- Un mot est accepté si une série de choix amène à un état de F. (Les mauvais choix ne comptent pas).

Automate fini non déterministe (2/2)

$\Sigma = \{a, b\}$ et $L_1 = \{xaay \mid x, y \in \Sigma^*\}$.



(non déterministe)



(déterministe)

Exercice 2 Trouver des automates finis pour reconnaitre les langages suivants :

- $L_2 = \{xaay \mid x, y \in \Sigma^*\} \cup \{xbby \mid x, y \in \Sigma^*\}$
- $L_3 = \{xwy \mid x, y \in \Sigma^*\}$ où w est un mot donné de Σ^*
- $L_4 \subset \Sigma^*$ tel que a est la 6ème lettre à partir de la fin dans tout $w \in_4 L$.

Les 3 théorèmes fondamentaux

Theorem 1 [Rabin-Scott] Tout langage reconnu par un automate fini non déterministe est reconnu par un automate fini déterministe.

Démonstration : on considère l'automate déterministe défini sur 2^Q, en prenant $\{q_0\}$ comme état initial et F comme ensemble de fin.

Remarque : l'automate déterminisé peut avoir 2^n états, si n est le nombre d'états de l'automate non-déterministe.

Theorem 2 [Myhill - Nerode] Tout langage reconnu par un automate fini est reconnu par un automate déterministe minimal unique à un isomorphisme près sur le nom des états.

Theorem 3 [Kleene] Les langages reconnus par un automate fini sont exactement ceux décrits par les expressions régulières.

Langages réguliers (ou rationnels en France).

Autres définitions d'automate fini

- $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow Q$
  déterministe avec transitions vides
- $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$
  non déterministe avec transitions vides
- $\delta : Q \times \Sigma^* \rightarrow 2^Q$ défini sur un sous-ensemble fini de Σ^*
  non déterministe avec transitions composées
- $\delta : Q \times \Sigma \rightarrow 2^{Q \times \{gauche, droite\}}$
  non déterministe, déplacement dans les 2 sens

Exercice 3 Montrer que toutes ces définitions sont équivalentes à la définition initiale d'automate fini (dernier cas plus dur).

Exercice 4 Montrer que si dans la définition 3 on enlève la restriction *finie*, alors on sort de la définition des automates finis.

Machine de Turing (1/6)

Reconnaître $\{a^n b^n \mid n > 0\}$

$\boxed{a}aabbb$
$X\boxed{a}abbb$
$Xa\boxed{a}bbb$
$Xaa\boxed{b}bb$
$Xa\boxed{a}Ybb$
$X\boxed{a}aYbb$
$\boxed{X}aaYbb$

$X\boxed{a}aYbb$
$XX\boxed{a}Ybb$
$XXa\boxed{Y}bb$
$XXaY\boxed{b}b$
$XXa\boxed{Y}Yb$
$XX\boxed{a}YYb$
$X\boxed{X}aYYb$

$XX\boxed{a}YYb$
$XXX\boxed{Y}Yb$
$XXXY\boxed{Y}b$
$XXXYY\boxed{b}$
$XXXY\boxed{Y}Y$
$XXX\boxed{Y}YY$
$XX\boxed{X}YYY$

$XXX\boxed{Y}YY$
$XXXY\boxed{Y}Y$
$XXXYY\boxed{Y}$
$XXXYYY\boxed{B}$
$XXXYYYB\boxed{B}$

Table de transitions (q_1 état initial, $F = \{q_5\}$)

	a	b	X	Y	B
q_1	(q_2, X, d)			(q_4, Y, d)	
q_2	(q_2, a, d)	(q_3, Y, g)		(q_2, Y, d)	
q_3	(q_3, a, g)		(q_1, X, d)	(q_3, Y, g)	
q_4				(q_4, Y, d)	(q_5, B, d)
q_5					

Exécution

Machine de Turing (2/6)

Une machine de Turing M est un 7-uplet $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ où :

Σ est un alphabet d'entrée,

Γ est un alphabet des symboles de la bande ($\Sigma \subset \Gamma$),

$B \in \Gamma - \Sigma$ est le symbole blanc,

Q est un ensemble fini d'états,

$q_0 \in Q$ est l'état initial,

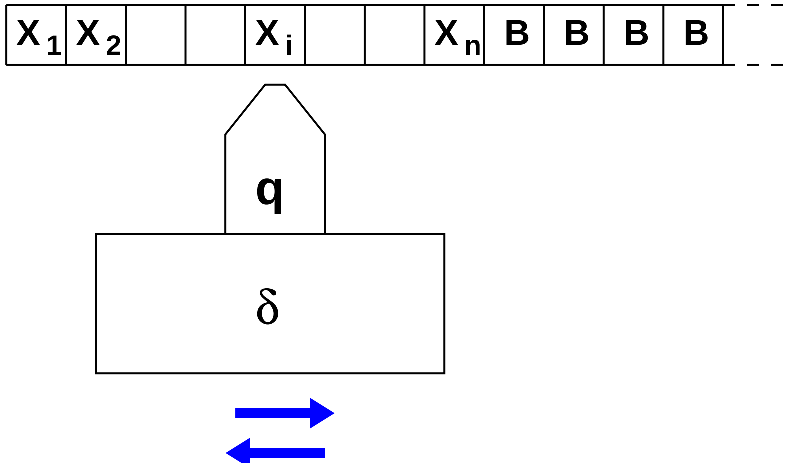
$F \subset Q$ est l'ensemble des états finaux,

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{gauche, droite\}$ est la fonction de transition

(u, q, v) est une configuration où

- $q \in Q$ est l'état courant,
- $u = X_1 X_2 \ldots X_{i-1} \in \Gamma^*$ est le mot à gauche de la tête de lecture,
- $v = X_i X_{i+1} \ldots X_n \in \Gamma^*$ est le mot à droite de la tête de lecture jusqu'à un caractère X_n suivi uniquement de blancs.

Machine de Turing (3/6)



Les transitions sont définies par :

$$
\begin{array}{rcl}
\delta(q,X) = (q',Y,droite) & \Rightarrow & (u,q,Xv) \longrightarrow (uY,q',v) \\
\delta(q,B) = (q',Y,droite) & \Rightarrow & (u,q,\epsilon) \longrightarrow (uY,q',\epsilon) \\
\delta(q,X) = (q',Y,gauche) & \Rightarrow & (uZ,q,Xv) \longrightarrow (u,q',ZYv) \\
\delta(q,B) = (q',Y,gauche) & \Rightarrow & (uZ,q,\epsilon) \longrightarrow (u,q',ZY)
\end{array}
$$

Le langage (récursivement énumérable) reconnu par M est :

$$T(M) = \{w \mid w \in \Sigma^*,\ (\epsilon, q_0, w) \xrightarrow{*} (u,q,v), q \in F\}$$

(La bande modifiable constitue une réserve infinie de mémoire)

Machine de Turing (4/6)

Exercice 5 Donner une machine de Turing qui reconnaisse les palindromes w formés de a et de b tels que $w = w^R$ où w^R est l'image mirroir de w.

Exercice 6 Trouver une machine de Turing qui reconnaisse les mots bien parenthésés formés de a et de b.

Exercice 7 Donner une machine de Turing qui calcule le successeur $x + 1$ de tout nombre binaire x.

Exercice 8 Donner une machine de Turing qui calcule la somme $x + y$ de tous nombres binaires x et y.

Exercice 9 Soit $\Sigma(n)$ le nombre maximal d'étapes que peut mettre une machine de Turing à n états avant de s'arréter en partant d'une bande complètement blanche. Calculer $\Sigma(n)$ pour $n \leq 4$ (le castor affairé, `Rado`).

Machine de Turing (5/6)

Autres définitions possibles de Machine de Turing :

- la bande est infinie à gauche et à droite
- avec plusieurs bandes et têtes de lecture
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{gauche, immobile, droite\}$
- $\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{gauche, droite\}}$
  non déterministe
- $\delta : Q \times \Gamma^* \rightarrow 2^{Q \times \Gamma \times \{gauche, droite\}}$ défini sur un sous-ensemble fini de Γ^* non déterministe avec transitions composées

Exercice 10 Montrer que toutes ces définitions sont équivalentes à la définition initiale.

Exercice 11 Montrer que si dans la définition 4 on enlève la restriction *finie*, alors on sort de la définition des machines de Turing.

Machine de Turing (6/6)

```
class Turing {
  int q0; Liste fin; Action[ ][ ] delta;

  static boolean accepter (String w, Turing m) {
    StringBuffer bande = new StringBuffer (w);
    int tete = 0; int q = m.q0;
    while ( !Liste.estDans(m.fin, q) ) {
      char c = tete < bande.length() ? bande.charAt(tete) : BLANC;
      Action a = m.delta [q][c];
      if (a == null) return false;
      q = a.q;
      if (a.dept == GAUCHE && tete > 0)
        bande.setCharAt(tete--, a.c);
      else if (a.dept == DROITE && tete < bande.length())
        bande.setCharAt(tete++, a.c);
      else if (a.dept == DROITE && tete == bande.length()) {
        bande.append(a.c); // new déguisé (allocation dans le tas)
        ++tete;
      } else return false;
    }
    return true;
  }

class Action { int q; char c; int dep;}
```

Turing [1936], Proc. London Math. Soc.

"On computable numbers, with an application to the Entscheidungsproblem"

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, i.e., on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrary small extent. The effect of this restriction of the number of symbols is not very serious. It is always possible to use sequences of symbols in the place of single symbols. Thus an Arabic numeral such as 17 or 9999999999999 is normally treated as a single symbol. Similarly in any European language words are treated as single symbols (Chinese, however, attempts to have an enumerable infinity of symbols). The differences from our point of view between the single and compound symbols is that the compound symbols, if they are not lengthy, cannot be observed at one glance. This in accordance with experience. We cannot tell at a glance whether 999999999999999999 and 999999999999999999 are the same.

The behaviour of the computer at any moment is determined by the symbols which he is observing, and his 'state of mind' at that moment. We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite. The reasons for this are of the same character as those which restrict the number of symbols. If we admitted an infinity of states of mind, some of them will be 'arbitrarily close' and will be confused. Again, the restriction is not one which seriously affects computation, since the use of more complicated states of mind can be avoided by writing more symbols on the tape.

Let us imagine the operations performed by the computer to be split up to 'simple operations' which are so elementary that it is not easy to imagine them further divided. Every such operation consists of some change of the physical system consisting of the computer and his tape. We know the state of the system if we know the sequence of symbols on the tape, which of these are observed by the computer (possibly with a special order), and 'state of mind' of the computer. We may suppose that in a simple operation not more than one symbol is altered. Any other changes can be split up into simple changes

.

of this kind. The situation in regard to the squares whose symbols may be altered in this way is the same as in regard to observed squares. We may, therefore, without loss of generality, assume that the squares whose symbols are changed are always 'observed' squares.

Diagonalisation

- Un langage récursivement énumérable peut être infini.
- Sa représentation par une machine de Turing est finie (nombre d'états fini, fonction de transition finie).
  $\Rightarrow$ On peut énumérer les machines de Turing, en prenant un codage quelconque de leur description. Soit M_i la i-ème machine de Turing.
- Egalement, on peut énumérer les mots sur Σ^* (en ordre par longueur, puis par ordre alphabétique par exemple). Soit w_i le i-ème mot.
- Considérons le langage $L = \{w_i \mid w_i \in \Sigma^*,\ w_i \notin T(M_i)\}$.
  S'il existe une machine M_d telle que $L = T(M_d)$, alors

  $$w_d \in T(M_d) \quad \text{ssi} \quad w_d \notin T(M_d)$$

  Contradiction !
- $\Rightarrow$ Il existe des langages non récursivement énumérables.

Fonctions calculables

- Si les mots figurant au début et à la fin sur la bande sont des nombres, une machine de Turing (déterministe) calcule des fonctions de $\mathbb{N} \rightarrow \mathbb{N}$.

  Une fonction calculable est une fonction calculée par une telle machine de Turing.

- Si, avec j au début sur la bande, M_i termine sans produire un entier sur la bande, on donnera 0 comme résultat de la fonction calculable correspondante.

- Comme les machines de Turing sont dénombrables, les fonctions calculables sont en bijection avec $\mathbb{N}$.

- $card(\mathbb{N} \rightarrow \mathbb{N}) > card(\mathbb{N})$ [Cantor]

- $\Rightarrow$ Il existe des fonctions non calculables.

Fonction non calculable

- Soit H une machine testant l'arrêt de toute machine M_i de Turing.

  $$H(i) = \begin{array}{l} 0 \;\; si \;\; M_i(i) \;\; diverge \\ 1 \;\; si \;\; M_i(i) \;\; converge \end{array}$$

- On peut alors construire une machine D telle que

  $$D(i) = \begin{array}{l} converge \;\; si \;\; M_i(i) \;\; diverge \\ diverge \;\; si \;\; M_i(i) \;\; converge \end{array}$$

  Alors si $D = M_d$, contradiction sur $D(d)$.

- $\Rightarrow$ Il n'existe pas de machine de Turing testant l'arrêt de toute machine de Turing.

Le problème de l'arrêt en Java

```
class Turing {
  static boolean termine (Object o) {
    // La valeur retournée vaut true si o.f()
    // termine. Sinon la valeur retournée vaut false.
    // (le code est breveté par le vendeur)
} }

class Facile { void f () { } }
class Boucle { void f () { while (true) do ; } }
class Absurde { void f () { while (Turing.termine(this)) ; } }

class Test {
  public static void main (String[ ] args) {
    Facile o1 = new Facile();
    System.out.println (Turing.termine(o1));
    Boucle o2 = new Boucle();
    System.out.println (Turing.termine(o2));
    Absurde o3 = new Absurde();
    System.out.println (Turing.termine(o3));
} }
```

`o3.f()` termine ssi `o3.f()` ne termine pas ! !

⇒ Contradiction (vive le logiciel libre !).

Expressivité – calculabilité

- Programmer des machines de Turing est fastidieux. Il manque un langage de programmation de haut niveau.
- Si on rajoute des instructions de haut niveau, peut-on reconnaître plus de langages sur Σ^* ?
- Peut-on calculer plus de fonctions avec des instructions de plus haut niveau ?

> **Thèse** [Church, 35] Tous les modèles de la calculabilité sont équivalents.

- Les machines de Turing, calculent toutes les fonctions calculables. Cette hypothèse n'a jamais été invalidée.
- Autres modèles tous équivalents : λ-calcul [Church], systèmes de Post [Post], fonctions récursives [Kleene], dominos de Wang, Java, ML, Ocaml, Ada, C, C++, PC Compaq, PC hp, Mac, etc.

Finitude de la calculabilité

Ingrédients nécessaires pour un modèle général de la calculabilité :

- description finie du fonctionnement
- réserve infinie d'espace mémoire

Exercice 12 Montrer que les machines de Turing *read-only* (ne pouvant donc pas écrire sur la bande) sont identiques aux automates finis.

Exercice 13 Montrer que les machines de Turing (ne pouvant écrire qu'une partie bornée de la bande) sont identiques aux automates finis.

Exercice 14 Pourquoi un ordinateur n'est pas un automate fini ?

Exercice 15 Refaire les raisonnements de diagonalisation avec des programmes Java au lieu de machines de Turing.