PUBLICATIONS DU DÉPARTEMENT DE MATHÉMATIQUES DE LYON

DOMINIQUE PERRIN
JACQUES SAKAROVITCH

Automates finis

Publications du Département de Mathématiques de Lyon, 1982, fascicule 1B
« Quelques thèmes de la théorie des algorithmes », , p. 27-36

<http://www.numdam.org/item?id=PDML_1982___1B_27_0>

AUTOMATES FINIS

Dominique PERRIN et Jacques SAKAROVITCH
LITP

La théorie des automates finis est une partie de l'Informatique dite théorique ou fondamentale. Elle a pour objet de décrire des algorithmes qui sont en un sens les plus élementaires possibles. Il s'agit des algorithmes qui ne requièrent pour s'exécuter qu'une mémoire de taille fixe indépendante de la donnée.

Cette restriction peut paraître bien mince au regard du fait que tout système physique, et en particulier tout ordinateur, ne possède qu'une mémoire de taille finie. Cependant un algorithme est un procédé qui s'applique en général à des objets dont la taille peut être arbitrairement grande. L'exécution d'un tel "algorithme nécessite une mémoire dont la taille est "potentiellement infinie". Par exemple, supposons donnés deux nombres écrits l'un au-dessous de l'autre en base 10. Un algorithme pour effectuer l'addition peut consister à lire les nombres de droite à gauche et conserver dans la mémoire la retenue en sortant au fur et à mesure les chiffres du résultat. En revanche tout algorithme effectuant la multiplication de ces deux nombres nécessite une mémoire dont la taille est proportionnelle au plus court des deux nombres.

Le schéma habituel d'un automate fini est un graphe fini dont les sommets sont nommés <u>états</u> et sont reliés par des flèches étiquetées par des lettres

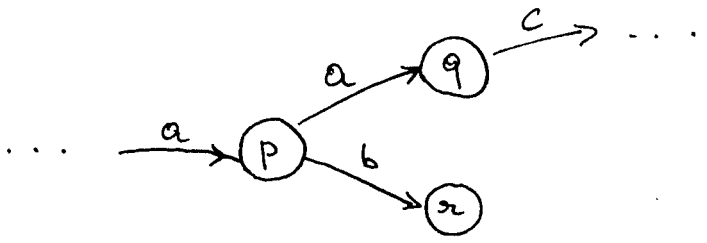

Figure 0

On distingue un sommet initial nommé <u>état initial</u>. Partant de ce sommet, on lit un mot donné w en suivant un chemin dans le graphe dont la suite des étiquettes correspond au mot w. Le résultat du calcul est l'état dans lequel se trouve l'automate en fin de lecture.

On peut ainsi aisément construire un automate fini vérifiant qu'un nombre

écrit en base 2 est multiple d'un nombre p.

Voici par exemple un automate fini correspondant au cas p = 3

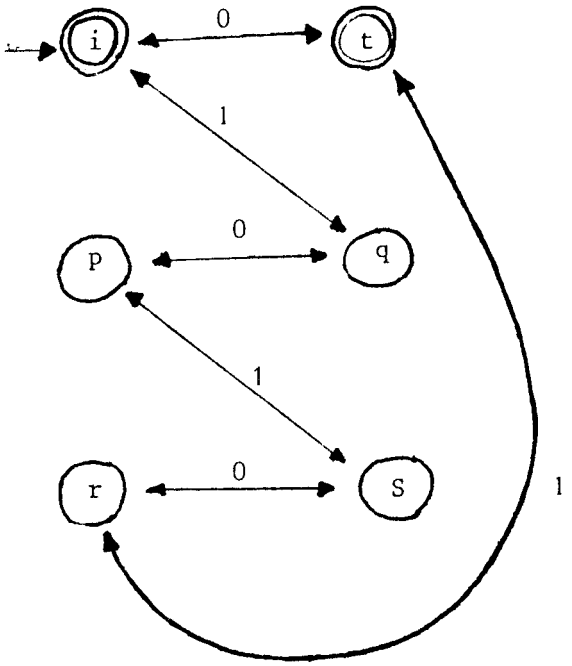

Figure 0bis

L'état initial est i. Chacune des flèches est double ($\overset{1}{\longleftrightarrow}$ est une abréviation pour $\underset{1}{\overset{1}{\rightleftarrows}}$). Si à la fin de la lecture de droite à gauche d'un nombre x l'automate se trouve dans les états i ou t il est multiple de 3. S'il se trouve en p ou q on a $x \equiv 1 \mod 3$ et s'il se trouve en r ou s on a $x \equiv -1 \mod 3$.

On ne pourrait par contre pas vérifier avec un automate fini qu'un tel nombre est une puissance exacte de 3 (Ceci est une conséquence d'un théorème dû à Cobham).

La théorie des automates finis a réellement commencé il y a une trentaine d'année avec le travail du logicien S. Kleene. Elle s'est développée depuis dans de nombreuses directions avec des motivations diverses dont certaines purement mathématiques. Le livre de S. Eilenberg (Automata, languages and Machines, Académic Press, 1974) en donne une présentation entièrement algébrique.

Dans les lignes qui suivent, nous avons cherché à donner des exemples aussi concrets que possible d'automates finis. Ils ont été choisis de façon à se trouver dans des domaines variés de l'informatique. La lecture de ce texte ne suppose aucune connaissance préalable de cette théorie. Si par contre elle donnait au lecteur le désir d'en connaître plus, les auteurs auraient atteint leur but.

1. SCHEMAS DE PROGRAMME

Considérons le petit programme suivant qui calcule la factorielle du nombre entier $n \geq 1$.

```
X   programme  FACT 1(n)
    début
a   x ← n  ;
b   y ← 1  ;
    tant que x ≠ 1 faire
        début
c       y ← y * x ;
d       x ← x - 1 ;
        fin
e   f ← y  ;
    fin
```

Figure 1.

En fin de calcul la variable f a pour valeur n!. La suite des instructions effectuées par ce programme peut être représentée par un mot sur l'alphabet $\{a,b,c,d,e\}$ en donnant aux instructions les noms indiqués sur la Figure 1.

L'ensemble X de toutes les suites possibles (pour les différentes valeurs de l'entier n) est

$$X = ab(cd)^{*}e$$

où le symbole $*$ désigne une itération de l'expression sur lequel il porte un nombre quelconque de fois. L'ensemble X peut être reconnu par un automate fini représenté par le graphe de la Figure 2.

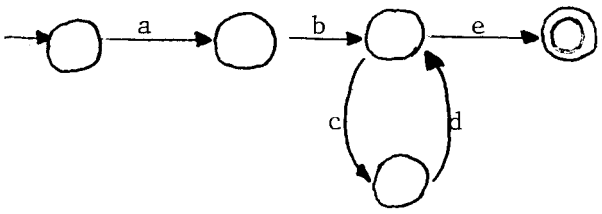

Figure 2.

On voit que ce graphe n'est rien d'autre que la représentation du programme de la Figure 1 sous la forme habituelle d'une charte (ou "organigramme").

Considérons maintenant un autre programme de calcul de n! où, comme dans le programme précédent le résultat se trouve dans la variable f.

```
Z     programme   FACT 2(n)
      début
      si n = 1 alors
e                  f ← 1
                  sinon
                     début
a                    n ← n-1 ;
Z                    FACT 2 (n) ;
b                    f ← (n+1) * f ;
                     fin
      fin
```

Figure 3

Cette fois l'ensemble des suites possibles de calculs est l'ensemble

$$Z = \{a^i \ e \ b^i \mid i \geqslant 0 \}$$

On ne peut reconnaître un tel ensemble avec un automate fini (cf. l'exposé de S. Grigorieff dans le même volume). Cela est lié au fait que, dans le programme FACT 2(n) il faut avant chaque appel récursif de FACT2(n) conserver la valeur de n à ce moment pour l'utiliser de nouveau dans l'instruction b.
Il est intéressant de remarquer que ce n'est pas le fait qu'il y ait un appel récursif dans FACT2 qui, à lui seul, suffit à produire ce phénomène. Considérons en effet le programme FACT3 ci-dessous :

```
X     programme  FACT3(n)
      début
a     x ← n ;
b     y ← 1 ;
Y     fact (x,y) ;
      fin
```

Figure 4.

Ce programme appelle le programme fact(x,y) ci-dessous

```
Y  programme fact(x,y)
   début
   si x = 1 alors
e          f← y
           sinon
              début
c             y← y * x  ;
d             x← x - 1  ;
Y             fact(x,y) ;
              fin

    fin
```

Figure 5

L'ensemble des suites d'instructions effectuées par fact(x,y) est

$$Y = (cd)^* e$$

et pour FACT3

$$X = abY$$

d'où

$$X = ab(cd)^* e$$

qui est identique à l'expression obtenue pour FACT1. Le fait que, bien que fact(x,y) s'appelle lui-même, il conduise à une expression reconnaissable par un automate fini est le phénomène de la récursion terminale : l'appel récursif de fact (x,y) est la dernière instruction du programme.

De façon générale, un programme de la forme

$$f(x) = \underline{si}\ p(x)\ \underline{alors}\ h(x)\ \underline{sinon}\ f(g(x)) \qquad (1)$$

peut transformer directement en

```
tant que ¬p(x) faire  x← g(x) ;          (2)
f← h(x) ;
```

Le programme FACT1 est de la forme (2) et le programme FACT3 de la forme (1)

En effet, FACT3 est de la forme :

$$f(x,y) = \underline{si}\ x = 1\ \underline{alors}\ y\ \underline{sinon}\ f(x-1,\ y * x)$$

Par contre le programme FACT2 s'écrit dans ces notations

$$f(x) = \underline{si}\ x = 1\ \text{alors}\ 1\ \underline{sinon}\ x * f(x-1)$$

où l'on voit bien apparaître le fait qu'il n'y a pas récursion terminale. Le passage d'un programme sous la forme (1) à un programme sous la forme (2) peut se faire de façon automatique et cette possibilité est utilisée dans certains systèmes existants.

2.SYNCHRONISATION DE PROCESSUS

L'exemple suivant est emprunté à M. Nivat (in Formal language Theory, R.Book ed., Academic Press, 1980). On considère un processus p qui peut alternativement lire ou écrire dans une mémoire, la première action étant de lire. Il peut aussi attendre sans effectuer aucune action

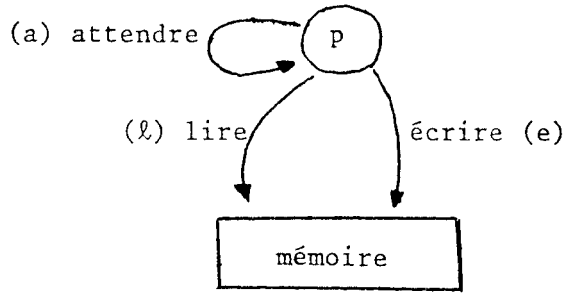

Figure 6

La suite des actions possibles du processus est donnée par l'expression

$$X = (a^* \ell\, a^*\, e\,)^* (a^* \ell a^* + a^*)$$

qui est asssociée à l'automate fini

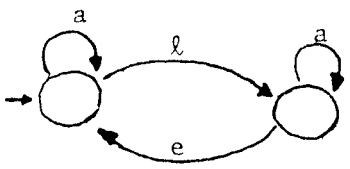

Figure 7.

de telle sorte que les comportements possibles sont les étiquettes des chemins partant de  .

On considère maintenant deux processus p_1 et p_2 ayant le même comportement que p. On impose de plus une condition de synchronisation : ils ne peuvent écrire tous deux en même temps et l'un d'eux ne peut lire pendant que l'autre écrit. La suite des actions possibles du couple (p_1,p_2) est alors décrite par l'automate fini ci-dessous

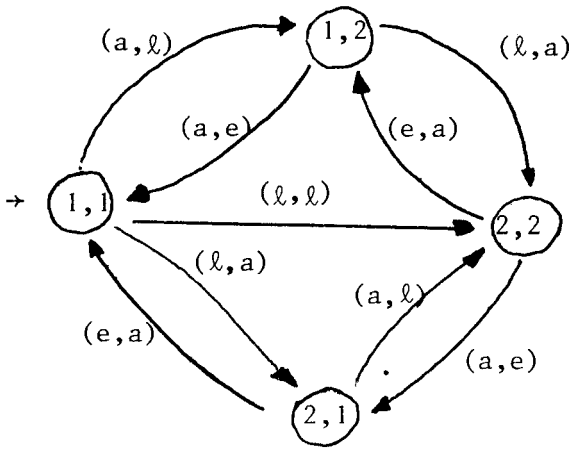

Figure 8.

Les problèmes typiques dont il est question dans le domaine de la synchronisation des processus sont : existe-t-il une possibilité de blocage du système ? Le système est-il équitable en ce sens qu'aucun des processus ne peut indéfiniment rester en attente ? De tels problèmes sont aisément décidables quand le processus est donné par un automate fini comme dans l'exemple ci-dessus.

3. AUTOMATES ET CIRCUITS

L'une des origines de la notion d'automate fini provient de la description des circuits logiques. De tels circuits constitués de lignes reliées par des portes "et", "ou" constituent des automates finis. Récemment ce point de vue a été repris pour la description des circuits dits VLSI ("very large scale integrated circuits"). On trouve en particulier dans un article récent (R. Floyd et D.Ullmann, JACM, 29 (1982) (603-622)), un bel exemple d'automate fini que voici : on considère une unité de contrôle recevant une suite de bits groupés par paire et interprétés de la façon suivante :

00	additionner
01	soustraire
10	charger
11	charger le complément.

La sortie est constituée de trois ordres possibles A,C et L, qui ont la signification suivante :

(A) additionner au contenu du registre celui du tampon mémoire.
(C) changer le contenu du tampon mémoire en son complément.
(L) charger le contenu du tampon mémoire dans le registre.

Quand l'ordre C a été envoyé, le contrôleur attend de recevoir un signal (X) indiquant que l'opération de complémentation est terminée avant d'envoyer un signal A ou L selon qu'il s'agissait d'une soustraction (01) ou de charger le complément (11). L'alphabet d'entrée du contrôleur sera donc constitué des quatre symboles 0,1,X et d'un quatrième symbole N indiquant l'absence d'un ordre à l'entrée. On peut décrire le fonctionnement du contrôleur par un automate fini comme ci-dessous :

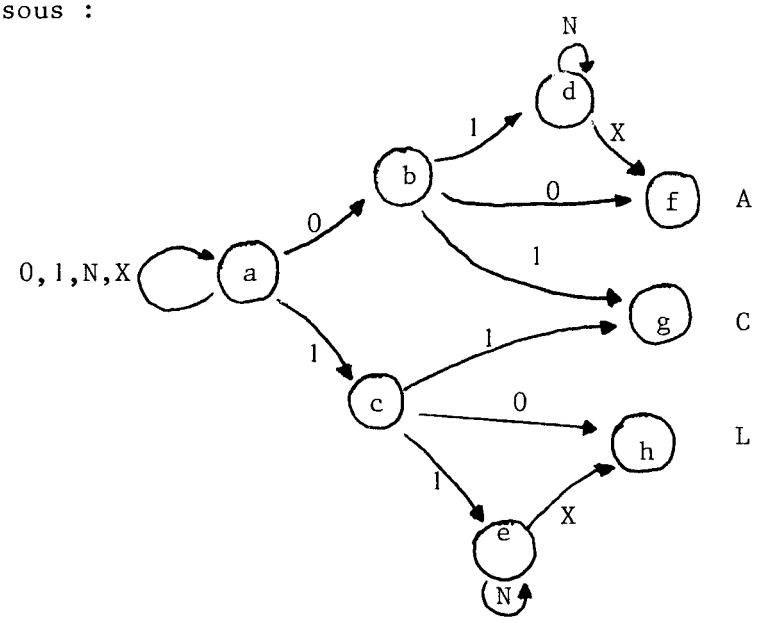

Figure 9.

On a indiqué à côté des états f,g,h les sorties A,C,L correspondantes. L'ensemble des suites de symboles d'entrée se traduisant à la fin par un signal A (additionner) est donné par l'expression.

$$.^{*}\, 0(0 + 1\, N^{*} X) \qquad (3)$$

où le symbole . désigne n'importe quel symbole d'entrée.

L'automate décrivant le fonctionnement du contrôleur peut, comme tout automate fini, être représenté par un tableau dit Tableau Logique Programmable (ce sont les PLA décrits dans le livre de Conway et Mead (VLSI, Addison Wesley, 1980)) qui est indiqué sur la Figure 10.

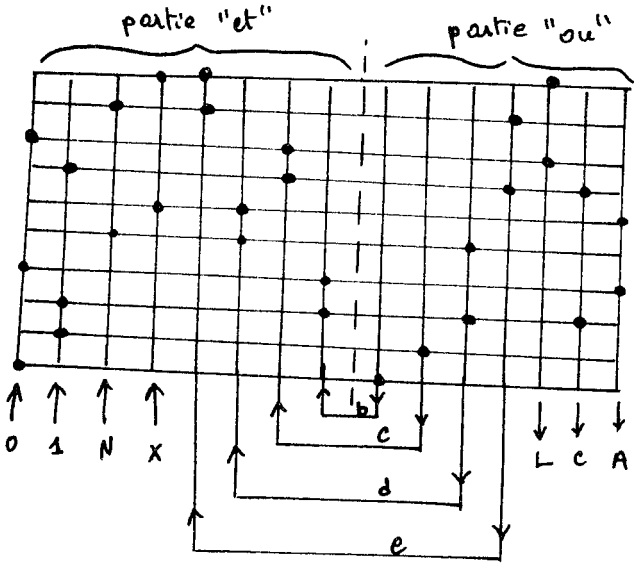

Figure 10. Un PLA

Le fonctionnement d'un tel tableau est le suivant : le plan quadrillé est constitué de lignes qui s'intersectent aux points indiqués par des • . Le plan est partagé entre une partie "et" et une partie "ou". Certains signaux (b,c,d,e) rentrent de la partie "ou" dans la partie "et". Toutes les lignes horizontales sont alimentées en courant à partir de la gauche. Pour pouvoir traverser la partie "et" il faut que toutes lignes verticales rencontrées en un • aientla valeur 1. Il traverse alors la partie "ou" et donne la valeur 1 à toutes les lignes verticales qu'il rencontre suivant un • . Les signaux rentrant de la partie"ou" dans la partie "et" le font avec un délai d'une unité de temps. Si par exemple la quatrième et la cinquième ligne verticale ont la valeur 1 (entrée X et état e) le signal traversera la ligne horizontale du haut et donnera la valeur 1 à la colonne correspondant à la sortie L.

La construction du PLA de la Figure 10 à partir de l'automate de la Figure 9 est très simple : on crée pour chaque flèche $p \xrightarrow{a} q$ de l'automate une ligne horizontale avec les connexions

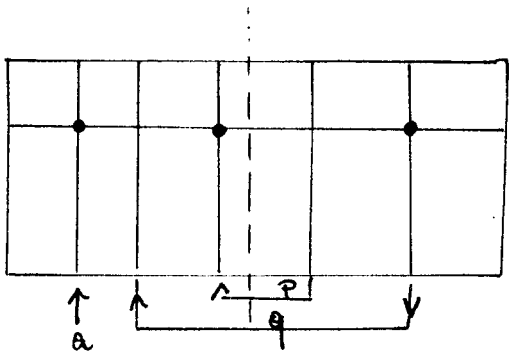

Figure 11

On peut de plus simplifier un peu le résultat en supprimant les colonnes correspondant à l'état a qui a toujours la valeur 1 et en confondant les lignes horizontales identiques. L'intérêt de la représentation d'un automate par un PLA est que le PLA est le dessin de l'implémentation physique d'un circuit.

4. RECHERCHE D'UN MOT.

La recherche dans un texte d'un mot donné est un problème qui doit être résolu à chaque fois qu'on veut disposer de ce qu'on appelle un éditeur de texte. C'est un programme que l'on peut utiliser pour modifier un texte déjà mémorisé dans une machine. On peut par exemple lui demander de substituer le mot "rigueur" au mot "austérité" à chaque fois qu'il se trouve dans le texte.

Supposons qu'on recherche dans un texte le mot

w = a b a a b

Le fait que ce mot n'appartienne probablement à aucune langue naturelle n'importe pas pour notre problème. L'algorithme de recherche le plus simple consiste à essayer de placer la première lettre de w en première position dans le texte et de cher cher si les caractères suivants coïncident. Sinon on essaie de placer la deuxième lettre en deuxième position, etc... Cela revient en un sens à utiliser l'automate suivant

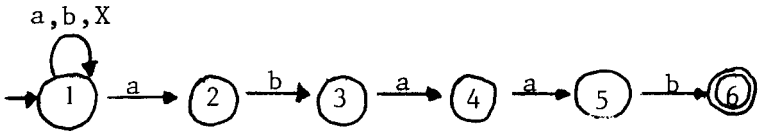

Figure 12.

où X figure un caractère distinct de a et b. On peut en fait aller plus vite en utilisant l'automate de la Figure 13.

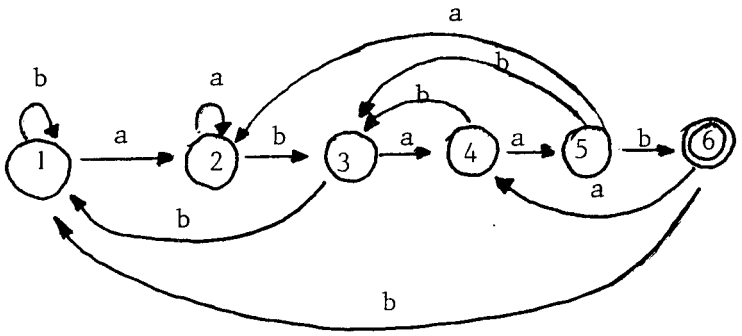

Figure 13.

Cet automate est déterministe : pour chaque état et chaque lettre il existe un unique état suivant. Cela permet de rechercher le mot w dans le texte sans revenir en arrière dans la lecture du texte comme on le faisait avec l'automate de la Figure 12. Ce dernier n'est pas déterministe et il faut donc essayer tous les chemins possibles.

La construction de l'automate de la Figure 13 à partir de celui de la Figure 12 est le résultat d'un procédé général appliquable à tout automate, connu sous le nom de déterminisation. En général cette construction mène à un automate dont le nombre d'états est beaucoup plus grand que celui de l'automate non déterministe de départ. Dans le cas d'un automate du type de celui de la Figure 12 la déterminisation ne change pas le nombre d'états. Pour calculer l'automate de la Figure 13, on peut se servir de l'algorithme suivant, connu sous le nom d'algorithme de Morris et Pratt (cf. M. Lothaire, Combinatorics on words, Addison Wesley, 1983).

```
1     φ(1) ← 0
2     pour i ← 2 jusqu'à n faire
3           début
4           j ← φ(i-1) ;
5           tant que (j > 0 et a_i ≠ a_{j+1} ou j = m faire j ← φ(j) ;
6           si j = 0 et a_i ≠ a_{j+1} alors φ(i) ← 0
7                                     sinon φ(i) ← j+1 ;
8           fin
```

Figure 14.

Si t est le texte dans lequel on doit chercher w on pose $wt = a_1a_2\ldots a_n$. La fonction $\varphi(i)$ calculée par l'algorithme de la Figure 14 a pour valeur j SSi $a_1\ldots a_j$ est un début de $w = a_1a_2\ldots a_m$ et une fin de $a_1\ldots a_i$ avec $j < i$ et la convention que $j = 0$ si un tel mot $a_1\ldots a_j$ n'existe pas.

L'état de l'automate de la Figure 13 après la lecture de $a_1a_2\ldots a_i$ pour $i \geq m+1$ est précisément $\varphi(i) + 1$.