automates finis

par Audrey Bouchon (1°S), Catherine Enjalbert (1°S), Caroline Hostalery (2nde) et Lucile Toussaert (1°S) du **lycée Fustel de Coulanges de Massy (91)**

enseignants :
MM. Michel Enjalbert et Hervé Hamon

chercheur :
M. Jean-Pierre Ressayre

compte-rendu de parrainage :

Des élèves du lycée Fustel de Coulanges à Massy (91), une en seconde et trois en première, ont travaillé sur les automates finis. Ce sont des machines (systèmes) qui permettent de lire (déchiffrer) des combinaisons de symboles : lettres et/ou chiffres.

L'exposé était très clair, compréhensible, accessible à tous. Les trois filles ont très bien organisé leur exposé et leurs interventions.

CIN — Mots et automates finis. 16

Un automate lit et écrit des *lettres* (des symboles) suivant des règles invariables, fixées à l'avance. Quand un automate est mis en présence d'une lettre (il la "lit"), il effectue, en fonction de cette lettre et de l'état dans lequel il est, une opération élémentaire, conformément aux règles fixées : il "écrit" éventuellement une lettre, se déplace d'un cran à droite ou à gauche, puis adopte un nouvel état ; la lettre suivante est prête pour la lecture.

Une telle machine simule parfaitement le fonctionnement de tout ordinateur. Le problème, fondamental en informatique, est de trouver des règles (si elles existent !) qui vont permettre à un automate de reconnaître ou de fabriquer infailliblement une certaine famille de mots : il s'agit par exemple de reconnaître les mots, formés avec les lettres *a* ou *b*, qui contiennent autant de *a* que de *b*, ou encore d'écrire correctement la somme de deux nombres lus par l'automate, etc.

Sujet.

1.— Apprendre le fonctionnement des automates finis, lisant des suites finies et acceptant ou rejetant ces suites.

2.— Construire un automate acceptant un ensemble de suites fixé. (Ou montrer qu'un tel automate n'existe pas, dans certains cas.) Problème inverse : un automate est donné, déterminer l'ensemble de toutes les suites qu'il accepte.

3.— Proposer une application concrète utilisant un automate fini ; réaliser le schéma de l'automate. (Réponse apportée à 3. : un digicode commandant l'ouverture d'une porte.)

Contenu de l'article.

Introduction : qu'est ce qu'un automate ?

Trouver les mots acceptés par un automate.

Théorème.

Construire le graphe d'un automate acceptant exactement des mots donnés.

Problème technique : le digicode.

Autres questions.

Introduction :
Qu'est-ce qu'un automate ?

Un automate est une machine qui lit des suites de caractères, appelés mots.

Quand il lit le premier caractère du mot, l'automate est dans un certain état : l'état initial. Selon le caractère qu'il lit et son « programme », il change (ou non) d'état. Dans ce « nouvel » état, il lit le second caractère. Selon ce caractère et son « programme », il change (ou non) d'état. Il procède ainsi pour les autres caractères jusque la fin du mot.

Il existe deux types d'états différents : acceptant et non-acceptant.

• Si quand il a fini de lire le mot, l'automate se trouve dans un état acceptant, le mot est dit accepté par l'automate.

• Sinon, il est dit non-accepté.

Ces mots peuvent être composés de beaucoup de caractères différents, mais nous n'avons étudié que des mots composés uniquement de « 0 » et de « 1 ».

Il existe une version des automates qui peut lire des mots de longueur infinie. Nous n'avons étudié que les automates capables de lire des mots de longueur finie. Pour faciliter l'étude, on utilise les graphes des automates.

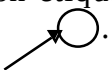
Pour matérialiser les états, on utilise des ronds :

• chaque état acceptant est un rond avec un + à l'intérieur : ⊕ ;

• chaque état non-acceptant est un rond vide : ◯.

Pour indiquer dans quel état l'automate doit passer après la lecture d'un caractère, on utilise une flèche étiquetée par le caractère en question ($\xrightarrow{1}$ ou $\xrightarrow{0}$) qui se dirige vers l'état dans lequel l'automate doit passer après lecture du caractère.

Par conséquent, il part de chaque état deux flèches (l'une étiquetée « 1 », l'autre « 0 ») ou une flèche étiquetée « 0, 1 » (ou « 1, 0 » ce qui revient au même : $\xrightarrow{0,\,1}$ signifie que quel que soit le caractère lu, l'automate passe dans l'état désigné par la flèche).

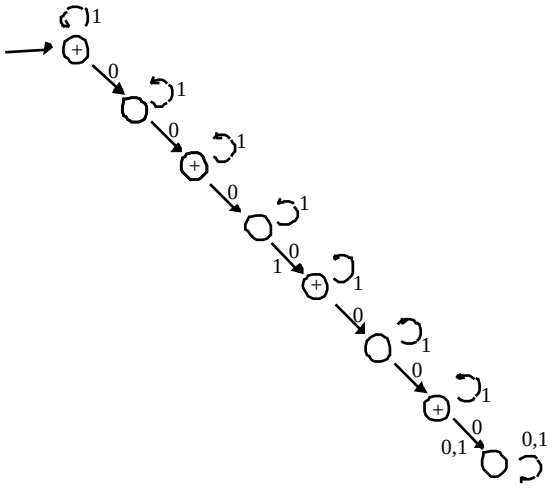
Pour indiquer quel est l'état initial, on utilise une flèche non étiquetée, qui ne provient d'aucun état :  .

Les premiers problèmes résolus donnent au lecteur des exemples pour comprendre la représentation d'un automate par un graphe.

Trouver les mots acceptés par un automate.

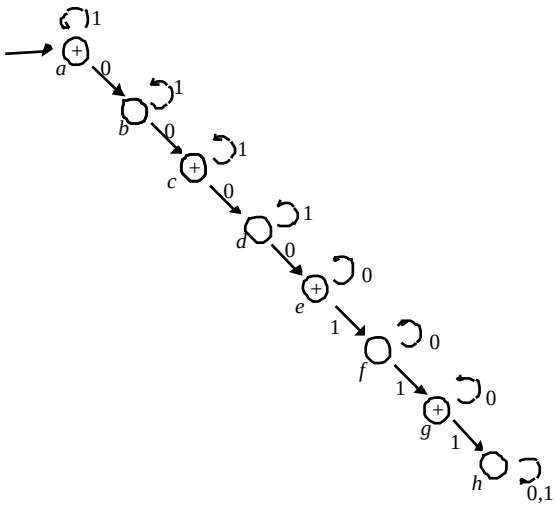
Problèmes simples :

Problème 1a
Quel est l'ensemble des mots acceptés par l'automate dont le graphe est donné ci-après.



Solution. Quand on fait un « 1 » on reste sur le même état donc seul le nombre de « 0 » importe. De plus, il y a un état acceptant tous les deux états et le mot vide est accepté. Il faut donc faire un nombre pair de « 0 ». Enfin si on fait plus de six « 0 » on arrive sur un état non-acceptant où, quoiqu'on fasse après, le mot ne sera pas accepté. Cet automate accepte donc les mots contenant aucun, deux, quatre, ou six « 0 ».

Problème 1b
Quel est l'ensemble des mots acceptés par l'automate dont le graphe est :



Solution. Les états acceptants sont placés tous les deux états et le mot vide est accepté.

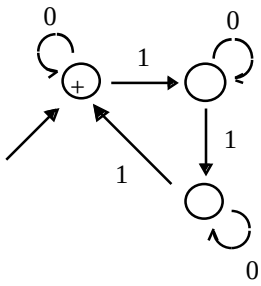
De plus, jusqu'à l'état *e* ce sont les « 0 » qui font changer d'état donc qui importent ; puis de l'état acceptant *e* à l'état *h* non-acceptant, c'est le nombre de « 1 ».

L'ensemble des mots acceptés est la réunion de :

• l'ensemble des mots contenant aucun, deux ou quatre « 0 ».

• l'ensemble des mots contenant plus de quatre « 0 » et aucun ou deux « 1 » après le quatrième « 0 ».

Problème 1c
Quel est l'ensemble des mots acceptés par l'automate dont le graphe est :



Solution. Ici seul le nombre de « 1 » importe car les « 0 » nous laissent sur place.

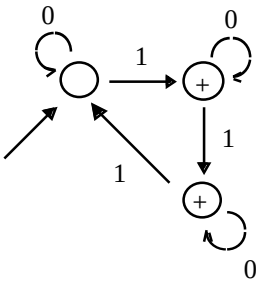
Cet automate comporte un seul état acceptant : le premier, le mot vide est donc accepté.

De plus, on remarque que l'automate forme une boucle et que l'on revient sur l'état acceptant de départ tout les trois « 1 ».

On en déduit donc que les mots acceptés par cet automate sont les mots comprenant un nombre de « 1 » divisible par trois.

Nous avons appelé cet automate un automate cyclique périodique de période trois « 1 ».

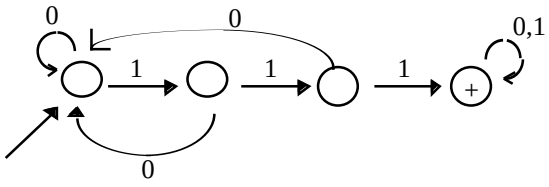
Problème 1d
Quel est l'ensemble des mots acceptés par l'automate dont le graphe est :



Solution. On remarque que le graphe de cet automate est identique à celui de l'automate précédent (**1c**) excepté les états : les états acceptants de cet automate sont des états non-acceptants dans l'automate **1c** et vice versa. Les mots acceptés par cet automate sont donc ceux qui ne sont pas acceptés par l'automate **1c**.

Les mots acceptés par cet automate sont donc les mots contenant un nombre de « 1 » non divisible par 3.

Problème 1e
Quel est l'ensemble des mots acceptés par l'automate dont le graphe est :



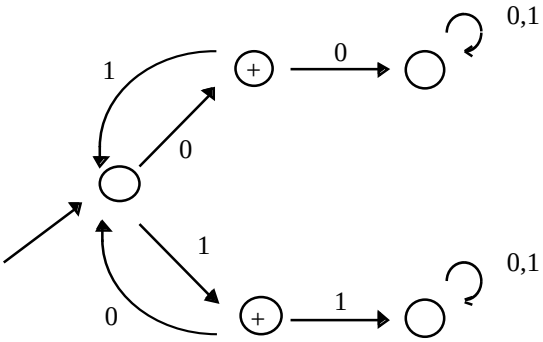
Solution. Le mot vide n'est pas accepté. On remarque qu'il existe un état terminal (duquel on ne peut pas sortir) acceptant. Tous les mots qui y font arriver sont donc forcément acceptés.

Pour y arriver, il faut faire trois « 1 » de suite car quel que soit l'emplacement d'un « 0 » dans le mot, quand l'automate le lit, il revient à l'état initial non acceptant.

L'ensemble des mots acceptés est donc l'ensemble des mots comportant au moins trois « 1 » de suite.

Problèmes plus compliqués

Problème 2a
Quel est l'ensemble des mots acceptés par l'automate dont le graphe est :

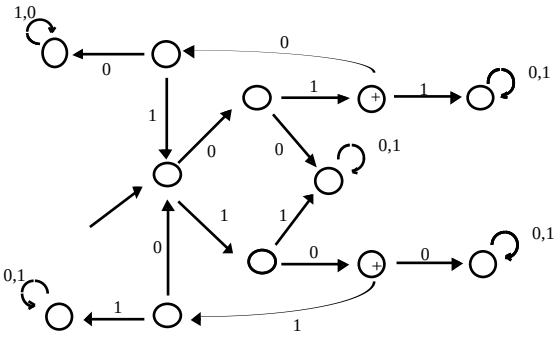


Solution. Ici le mot vide n'est pas accepté mais tous les mots de une lettre le sont. De plus, une fois arrivé sur un état acceptant, il ne faut pas refaire la même lettre que précédemment sinon on va sur un état où, quoi qu'on fasse, le mot n'est pas accepté. Et le nombre de termes doit être impair car l'état de départ n'est pas acceptant.

Les mots acceptés sont ceux comprenant un nombre impair de termes et dont chaque terme de rang pair au sens défini ci-dessous est différent du précédent.

définition : on appelle le premier terme d'un mot le terme de rang un, le deuxième terme celui de rang deux, etc.

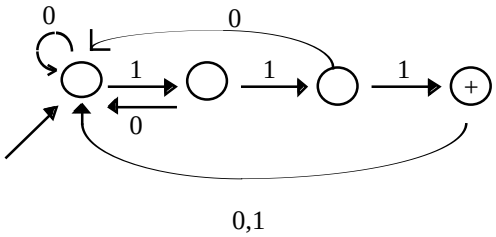
Problème 2b
Quel est l'ensemble des mots acceptés par l'automate dont le graphe est :



Solution. Cet automate comporte deux états acceptants qu'on atteint uniquement de l'état de départ en faisant « 01 » ou « 10 ». Mais l'état de départ n'est pas acceptant. Donc de l'état de départ il faut faire deux termes différents pour arriver sur un état acceptant puis de nouveau deux termes différents (et dont le premier est différent du précédent) pour revenir à l'état de départ. Sinon on arrive sur des états terminaux non-acceptants. Puis on a de nouveau le choix entre « 0 » et « 1 » et ainsi de suite.

Les mots acceptés sont ceux comprenant un nombre de termes divisible par deux mais pas par quatre (sinon on arrive sur l'état de départ non-acceptant) ; et dont chaque terme est différent du précédent sauf ceux de rang divisible par quatre qui peuvent être suivis de n'importe quel terme (car on est sur l'état de départ).

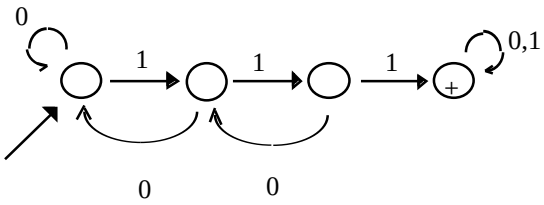
Problème 2c
Quel est l'ensemble des mots acceptés par l'automate dont le graphe est :



Solution. Ici le mot vide n'est pas accepté et il n'existe qu'un seul état acceptant ; de plus, c'est le nombre de « 1 » qui est important. Les flèches portant les « 0 » ramènent toutes à l'état de départ et la flèche portant le « 1 » et partant de l'état acceptant ramène aussi à l'état de départ. Après le dernier « 0 », on peut avoir trois « 1 », on est alors sur l'état acceptant. S'il y a plus de trois « 1 », pour revenir à l'état acceptant, il faut ajouter un nombre de « 1 » divisible par quatre (le premier « 1 » pour revenir à l'état initial).

Donc les mots acceptés par cet automate sont les mots comprenant $4n$+3 « 1 » ($n \in$ N) après le dernier « 0 » (si le mot en contient).

Problème 2d
Quel est l'ensemble des mots acceptés par l'automate dont le graphe est :



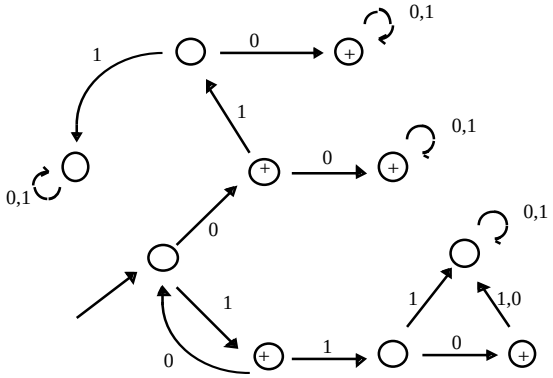
Solution. Le mot vide n'est pas accepté. Il existe un seul état acceptant et il est terminal. Pour y arriver, il faut faire trois « 1 » de suite, ou « 1 » puis « 10 » un nombre de fois au moins égal à un, puis « 11 ». On note ceci ainsi : 1(10)+ 11.

L'ensemble des mots acceptés est donc l'ensemble des mots comprenant trois « 1 » de suite ou 1(10)+11. C'est-à-dire l'ensemble 1(10)®11, la notation (10)® indiquant qu'on peut faire « 10 » un nombre de fois quelconque, y compris zéro fois.

Théorème

Pour tout automate *A*, il existe un automate *B* acceptant les mêmes mots et ayant au plus deux états terminaux.

Preuve :

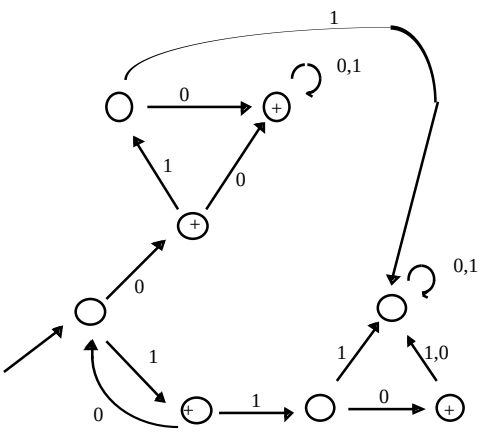


Un état de *A* est terminal si on n'en sort plus : les flèches qui en partent y reviennent. Il existe deux sortes différentes d'états terminaux : ceux acceptants, et les autres.

On peut toujours choisir un des états terminaux acceptants (*a*) et y envoyer les flèches conduisant aux autres états terminaux acceptants.

De même, pour les états terminaux non acceptants, en choisissant l'un d'eux (*b*).

On a alors construit un automate B acceptant les mêmes mots que *A* et qui a au plus deux états terminaux.



Construire le schéma d'un automate acceptant exactement des mots donnés.

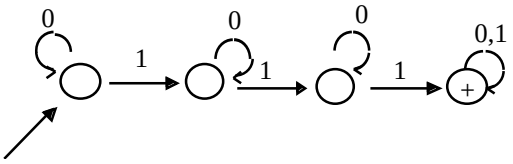
Problèmes simples :

Problème 3a
Construire un automate acceptant précisément les mots contenant au moins trois « 1 ».

Solution. [Remarque : Ici les « 1 » ne sont pas forcément consécutifs, contrairement au problème **1e**.] Les mots acceptés ne tiennent pas compte des « 0 », donc le fait de faire un « 0 » ne change rien, et donc chaque flèche avec « 0 » revient à l'état duquel elle est partie.

Les mots acceptés sont ceux avec au moins trois « 1 », donc il faut qu'à partir du troisième « 1 », tous les mots soient acceptés, alors qu'avant ils doivent être tous refusés.

On fait donc, à partir de l'état de départ non acceptant, trois flèches avec des « 1 », qui partent toutes d'un état non acceptant et dont la troisième arrive à un état acceptant terminal.



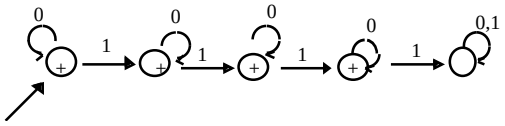
Problème 3b
Construire un automate acceptant précisément les mots contenant au plus trois 1.

Solution. Les mots acceptés ne tiennent pas compte des « 0 », donc le fait de faire un « 0 » ne change rien, et donc chaque flèche avec « 0 » revient à l'état duquel elle est partie.

Les mots acceptés sont ceux avec au plus trois « 1 », donc il faut qu'à partir du quatrième « 1 », tous les mots soient refusés, alors qu'avant ils doivent être tous acceptés.

On fait donc, à partir de l'état de départ acceptant, quatre flèches avec des « 1 », qui

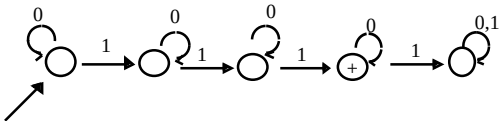
partent toutes d'un état acceptant et dont la quatrième arrive à un état non acceptant terminal.



Problème 3c
Construire un automate acceptant précisément les mots contenant exactement trois « 1 ».

Solution. Les mots acceptés ne tiennent pas compte des « 0 », donc le fait de faire un « 0 » ne change rien, et donc chaque flèche avec « 0 » revient à l'état duquel elle est partie.

Les mots acceptés sont ceux avec exactement trois « 1 », donc il faut qu'au troisième « 1 », les mots soient acceptés, alors qu'avant ce troisième « 1 » et à partir du quatrième « 1 », ils doivent être tous refusés.

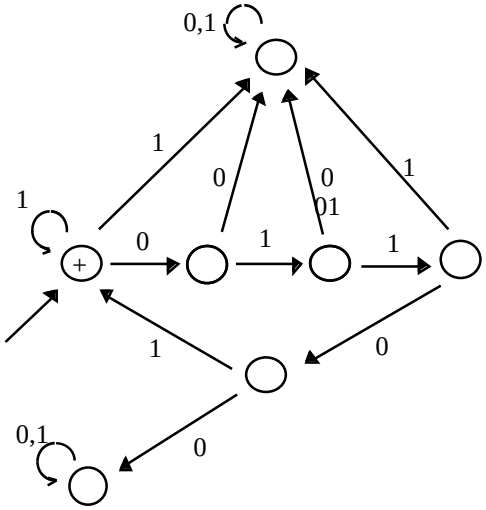


On fait donc, à partir de l'état de départ non acceptant, quatre flèches avec des « 1 », dont la troisième mène à un état acceptant, alors que les deux premières mènent à des états non acceptants, et que la quatrième mène à un état non acceptant terminal.

Problèmes plus compliqués :

Problème 4a
Construire un automate acceptant précisément tous les mots de la forme $(01101)^n$, $n \in \mathrm{N}$.

Solution.

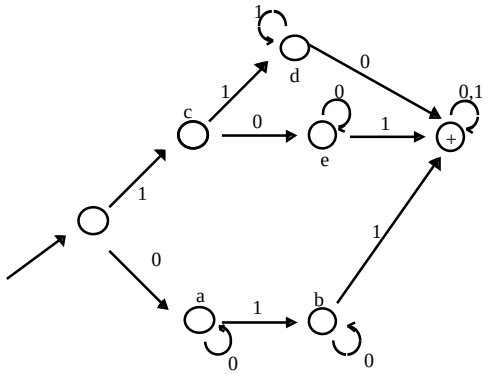


D'après l'énoncé, les mots acceptés sont constitués de (01101) répété n fois, $n \in \mathrm{N}$, donc le mot vide est accepté. On peut dire aussi que dès qu'on fait un chiffre ne correspondant pas au cycle (01101), la condition n'est définitivement pas respectée et donc on arrive à un état non acceptant terminal. Comme on vient de le dire, on peut assimiler le « n fois (01101) » à un cycle de (01101), qui, à chaque fois qu'il se finit mène à un état acceptant, alors qu'avant, par exemple quand on a fait n–1 fois (01101) puis (011), on n'arrive qu'à des états non acceptants.

Donc, sur le graphe, on fait (01101) en flèches dont la dernière mène à l'état de départ acceptant, alors que les autres mènent à des états non acceptants. Pour chacun des états déjà mis, on fait une flèche qui en part et qui mène à un état non acceptant terminal. L'étiquette de chacune de ces flèches est complémentaire de celle de la flèche qui part du même état, c'est-à-dire que, si une flèche déjà inclue au graphe est étiquetée « 1 », l'autre flèche qui part du même état doit être étiquetée « 0 », et vice-versa.

Problème 4 b
Construire un automate acceptant précisément les mots contenant au moins un « 0 » et deux « 1 ».

Solution.



Tout d'abord, l'état de départ n'est pas acceptant, car la condition n'est pas encore respectée. La flèche avec un « 0 » mène à un état (*a*) non acceptant d'où part et revient une flèche avec un « 0 », car quand on a fait un « 0 », la partie de la condition concernant les « 0 » est respectée et on peut faire autant de « 0 » que l'on veut, cela ne changera rien.

La flèche avec le « 1 » qui part de cet état (*a*) mène à un état (*b*) non acceptant d'où part et revient une flèche avec un « 0 », toujours pour la même raison. Par contre, la flèche avec le « 1 » partant de l'état (*b*) mène à un état acceptant terminal, car alors la condition est définitivement respectée.

A partir de l'état de départ, la flèche avec le « 1 » mène à un état (*c*) non acceptant d'où part une flèche avec un « 1 » et une flèche avec un « 0 », menant à deux états non acceptants différents, nommés respectivement (*c*) et (*d*). L'état (*d*) où mène la flèche avec le « 1 » est le point de départ d'une flèche avec un « 1 » qui revient de là où elle est partie (car on a déjà fait deux « 1 » et cette partie de la condition est définitivement respectée quel que soit le nombre de « 1 » qu'on fera après) et d'une flèche avec un « 0 » qui mène à un état acceptant terminal (car la condition est définitivement respectée).

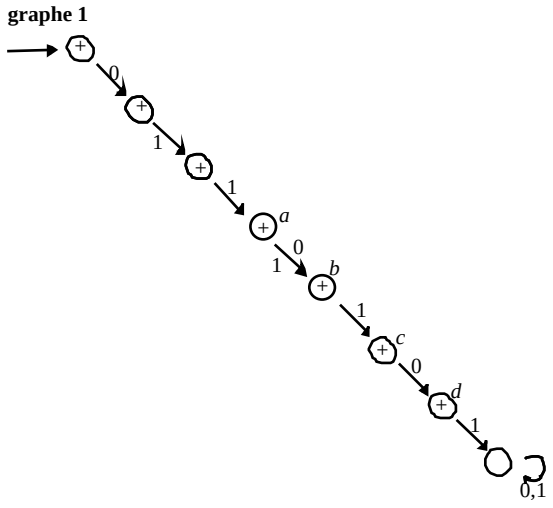
L'état « e » est le point de départ d'une flèche avec un « 0 » qui revient elle est partie (car en ayant fait un « 0 », cette partie de la condition est définitivement respectée quel que soit le nombre de « 0 » qu'on fera après) et d'une flèche avec un « 1 » qui mène à un état acceptant terminal (car la condition est définitivement respectée).

Problème 4c
Construire un automate acceptant précisément les mots de l'ensemble L tel que si L contient un « 0110 », alors L ne contient pas « 0101 ».

Solution. D'après l'énoncé, l'automate est tout le temps acceptant sauf dès qu'on a « 0110 » et « 0101 » dans le même mot. A ce moment, l'état non acceptant est final, car la condition n'est définitivement pas respectée. Donc on va étudier tous les cas possibles où « 0110 » et « 0101 » sont présents dans le même mot.

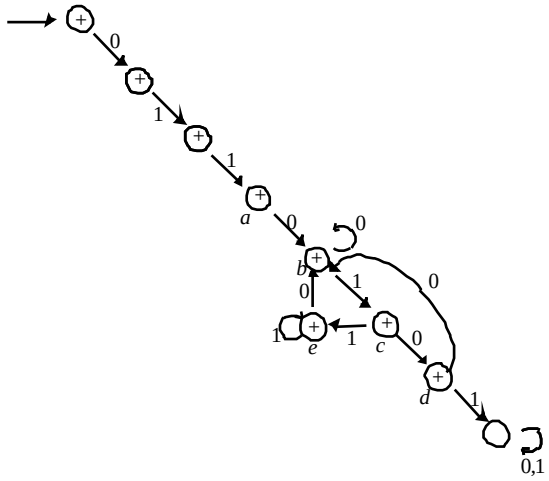
Tout d'abord, le cas où c'est « 0110 » qui se présente en premier.

On construit le parcours « 0110101 » en flèches qui partent toutes d'états acceptants et dont la dernière arrive à un état non acceptant terminal :



Nous complétons ce graphe de manière que si on a fait « 0110 », alors « 0101 » soit la seule suite conduisant à l'état final non acceptant.

graphe 2



Voici les remarques qui conduisent du **graphe 1** au **graphe 2** et montrent que celui-ci convient. Nous partons de l'état (*b*) car on vient de faire « 0110 ».

A partir de l'état (*c*), il y a une possibilité qu'on refasse « 0110 », car on vient de faire « 01 ». Si on fait « 1 » après l'état (*c*), on aura fait « 011 », ce qui n'est pas le début de « 0101 », c'est pourquoi la flèche avec le « 1 » qui part de l'état (*c*) mène à un état acceptant, appelé (*e*).

A partir de cet état, on peut faire autant de « 1 » qu'on veut car la fin du mot déjà fait, « 111 », n'est pas le début de « 0101 », donc la flèche avec le « 1 » qui part de l'état (*e*) y revient.

Si on fait « 0 » à partir de l'état (*e*), alors on peut avoir le début de « 0101 », comme quand on arrive à l'état (*b*), donc la flèche avec le « 0 » qui part de (*e*) mène à (*b*). Quand on est à l'état (*b*), on peut faire autant de « 0 » qu'on veut, ça sera toujours le début de « 0101 », donc la flèche avec le « 0 » qui part de l'état (*b*) y revient.

Quand on fait un « 0 » à partir de l'état (*d*), il n'y a que ce « 0 » qui soit le début de « 0101 », ce qui revient au même que quand on se trouve à l'état (*b*), donc la flèche avec le « 0 » qui part de l'état (*d*) mène à l'état (*b*).

D'où le **graphe 2** !

Voici des remarques qui conduisent alors au **graphe 3** plus bas.

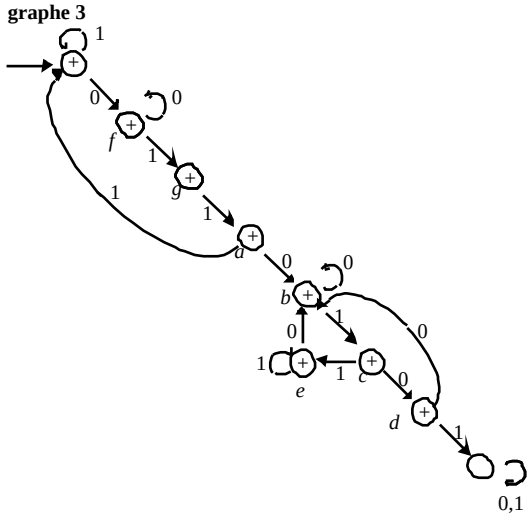
Quand on fait « 1 » à partir de l'état (*a*), la fin de notre mot est (0111) ce qui n'est le début ni de (0110) ni de (0101), comme quand on se trouve à l'état de départ, donc la flèche avec le « 1 » qui part de l'état (*a*) mène à l'état de départ.

Quand on se trouve à l'état de départ, on peut faire autant de « 1 » qu'on veut, car ce n'est le début ni de (0110), ni de (0101), donc la flèche avec le « 1 » qui part de l'état de départ y revient.

De même à partir de l'état (*f*), on peut faire autant de « 0 » que l'on veut, car ce sera toujours le premier « 0 » de (0110) et de (0101) donc la flèche avec le « 0 » qui part de (*f*) y revient.

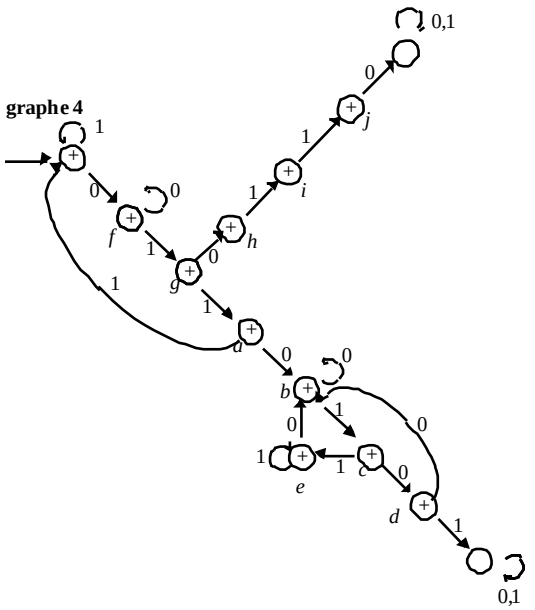
A ce moment l'automate déjà constitué ressemble à ceci :

graphe 3



Le graphe rejette tous les mots qui comportent *d'abord* « 0110 », *puis* « 0101 »; mais il nous faut le compléter pour qu'il rejette aussi ceux qui ont « 0101 » en premier et plus tard « 0110 ».

On construit donc (010110) en flèches, les deux premières étant celles allant aux états (*f*) et (*g*), qui toutes partent d'états acceptants, et dont la dernière mène à un état non acceptant terminal. On a alors …



Quand on fait un « 0 » à partir de l'état (*h*), le mot se finit par deux « 0 », on a donc alors le premier « 0 » de (0101) ou de (0110), ce qui nous ramène au même point que quand on est dans l'état (*f*). La flèche avec le « 0 » qui part de l'état (*h*) mène donc à l'état (*f*).

Quand on arrive à l'état (*i*), on a déjà fait (0101). On ne doit donc pas faire (0110) sous peine d'arriver à un état non acceptant final.

La flèche avec le « 0 » qui part de (*i*) mène à un état acceptant (*k*), car la condition est pour l'instant respectée.

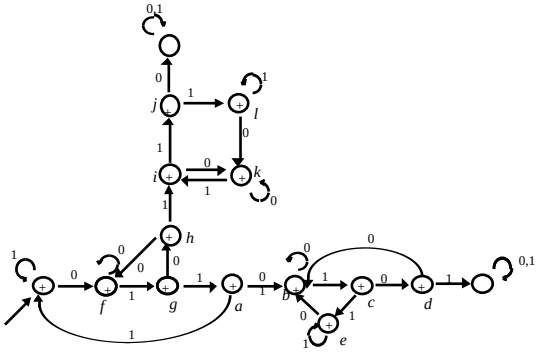
Quand on fait un « 1 » à partir de (*k*), la fin du mot est (01), ce qui est le début de (0110) et ce qui nous mène au même point que lorsqu'on se trouve sur l'état (*i*), donc la flèche avec le « 1 » qui part de (*k*) mène à (*i*).

Quand on se trouve sur (*k*), on peut faire autant de « 0 » qu'on veut, on aura toujours le « 0 » de début de (0110), donc la flèche avec le « 0 » qui part de (*k*) y revient.

La flèche avec le « 1 » qui part de (*j*) mène à un état acceptant (*l*), car la condition est pour l'instant respectée.

Quand on arrive sur (*l*), on vient de faire (0111), ce qui n'est pas le début de (0110), donc, à ce moment là, on peut faire autant de « 1 » qu'on veut, on n'aura toujours pas le début de (0110), donc la flèche avec le « 1 » qui part de (*l*) y revient.

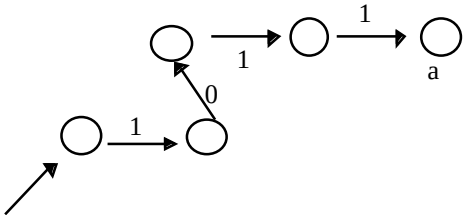
Si on fait « 0 » à partir de (*l*), on a juste le premier « 0 » de (0110), ce qui nous mène au même point que lorsqu'on est sur l'état (*k*), donc la flèche avec le « 0 » qui part de (*l*) arrive à (*k*). L'automate est alors fini : OUF!



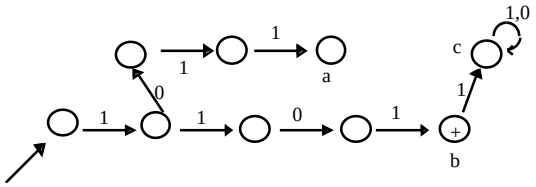
Problème 4d
Construire un automate acceptant précisément les mots contenant (1101) et pas (1011).

Solution. Cet exercice est difficile et il faut le faire avec méthode.

Tout d'abord, dès qu'on a fait (1011), on arrive à un état non acceptant terminal, alors que quand on a fait (1101), on arrive à un état acceptant mais pas terminal.



Sur notre graphe, à partir de l'état de départ non acceptant (car la condition n'est pas respectée), on écrit (1011) en flèches qui mènent toutes à des états non acceptants, la dernière menant à l'état (*a*) non acceptant terminal (la condition est définitivement non respectée).



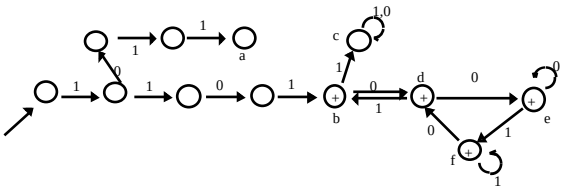
Puis, à partir de l'état de départ, on écrit (1101) en flèches (la première de celles-ci étant la même que celle du (1011)), qui

mènent à des états non acceptants sauf la dernière qui mène à l'état (*b*). La condition (1101) est alors respectée. A partir de là on cherche donc à ne pas avoir (1011).

On s'aperçoit alors qu'en rajoutant un « 1 » après (1101), on a (11011), qui contient donc (1011), ce qui fait que la condition n'est définitivement pas respectée. Donc, la flèche avec un « 1 » partant de (*b*) mène à l'état (*c*) non acceptant terminal.

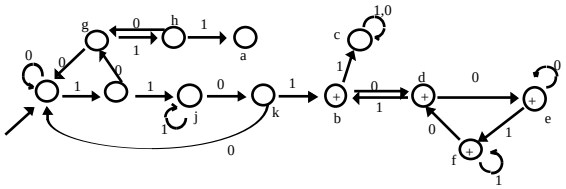
Si on fait (01) après l'état (*b*), on se trouve dans la même situation qu'en (*b*) car les trois derniers caractères lus sont (101), donc on fait une flèche avec un 0 menant à un état acceptant (*d*), d'où part une flèche avec un « 1 » menant à l'état (*b*).

Par contre, si on fait (00) après l'état (*b*), on arrive à l'état (*e*). On n'est pas au début de (1011), et on peut faire alors autant de « 0 » que l'on veut, donc l'état (*e*) est acceptant et la flèche avec un « 0 » y revient. A partir de (*e*), en rajoutant un ou plusieurs « 1 », on n'est pas à la fin d'un (1011), donc on fait une flèche avec un « 1 » menant à un état acceptant (*f*) d'où part et revient une flèche avec un « 1 ». Puis, à partir de (*f*), si on fait un « 0 », on vient de faire (10). On est donc dans la même situation qu'en arrivant sur l'état (*d*) donc la flèche avec le « 0 » partant de (*f*) revient à l'état (*d*). On a ainsi fini une première étape.



On passe alors à la deuxième étape : pour chaque état, on cherche où mènent les flèches qu'on n'a pas encore mises. On commence par l'état de départ : la flèche avec le zéro revient d'où elle part, car « 0 » n'est le commencement ni de (1011), ni de (1101). Puis, on passe à l'état (*g*) situé après (10) en flèches : la flèche avec le « 0 » revient à l'état de départ, car (100) n'est le début ni de (1011), ni de (1101). La flèche avec le « 1 » mène à l'état (*h*).

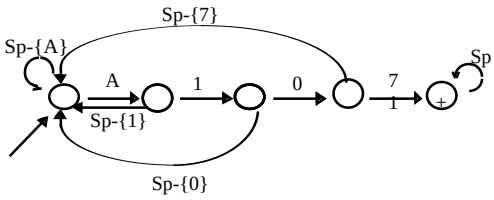
En faisant « 0 » après (101), on a (10), le début de (1011), donc la flèche avec le « 0 » partant de (*h*) revient à l'état (*g*).



A partir de l'état de départ, après avoir fait (11), on arrive à l'état (*j*). Si on refait encore un ou plusieurs « 1 », on a toujours le début de (1101), donc la flèche avec le « 1 » revient à l'état (*j*) duquel elle est partie.

Pour l'état (*k*) après (110), la flèche avec le « 0 » revient à l'état de départ, car (100) n'est le début ni de (1101), ni de (1011). On a alors fini le graphe.

Problème technique : le digicode.



Nous avons imaginé une application technique des automates : le digicode commandant l'ouverture d'une porte (d'un hall d'immeuble par exemple). Supposons que le code soit : A107.

On appelle *P* l'automate commandant l'ouverture. *P* n'ouvre la porte que si le mot tapé contient A107. L'ensemble des caractères de *P* est :

$Sp = \{A , B , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 \}\}$

(nous ne prenons pas en compte que la porte se referme au bout d'un certain temps, ce n'est plus de notre ressort !).

Autres questions.

Nous avons maintenant pensé à de nouvelles pistes de recherche :

• construire un automate programmable c'est-à-dire un automate qui se modifie quand il lit certains mots ;

• trouver parmi les automates à nombre fini d'états, celui (ou ceux) qui accepte(nt) les mots les plus longs et trouver ces mots.

Point de vue du chercheur.

Du bon travail 1) de recherche
2) pour rédiger celle-ci.

1) La solution des premiers problèmes donne l'impression qu'ils sont faciles. Mais pas au départ, quand on connaît *seulement* la définition des automates et leur graphe …

Quant aux problèmes compliqués, ils le sont un peu plus que ceux que se posent les étudiants en première année dans une U.V. « architecture des ordinateurs » par exemple.

2) La rédaction des résultats peut viser soit (*i*) soit (*ii*) :

i) vérifier avec *certitude* le résultat. Pour des problèmes compliqués (tels que 4.c, 4.d) ce n'est guère possible en temps raisonnable sinon à l'aide d'un *ordinateur*. C'est un objectif qui n'a pas été visé, mais qui pourrait l'être par la suite …

ii) permettre à un lecteur humain de vérifier en temps raisonnable et de manière qu'en lisant il apprendra lui-même à résoudre et vérifier de tels problèmes (étant entendu qu'on ne vise pas la rigueur totale et tolère un certain risque d'erreur).

Les jeunes ont très bien réalisé (*ii*) ci-dessous : à leur place, j'aurais fait plutôt moins bien — plus lourd et moins suggestif du cheminement vers la solution.

Point de vue des enseignants.

Beaucoup d'élèves (une trentaine) se sont manifestés en début d'année, mais seulement quatre ont persévéré. Il est difficile de faire comprendre aux élèves qu'une activité volontaire nécessite une présence régulière.

A défaut de quantité on a eu la qualité et la régularité dans le travail. Nos participantes ont fait preuve d'autonomie dans la recherche (nous sommes très peu intervenu) et de persévérance dans le travail quelque peu fastidieux de rédaction.