Nous présentons ici un type de machines simples, appelés **automates fini déterministes**, qui disposent d'une **mémoire finie** (en abrégé *AFD*).

Un **automate fini déterministe** $\mathcal{A}$ est un quintuplet $< Q, X, \delta, q_0, Q' >$ dans lequel :

- Q est l'**ensemble fini des états** (la mémoire de l'automate) ;
- X est l'alphabet fini des lettres sur lequel l'automate travaillera : l'automate recevra comme donnée un mot de X^* sur lequel il réalisera un certain type de calcul ;
- δ (qu'on notera éventuellement $\delta_{\mathcal{A}}$ si on manipule plusieurs automates) est la **fonction de transitions** de l'automate $\mathcal{A}$:

  $$\delta : Q \times X \to Q \quad \text{(éventuellement non définie pour certains couples)}$$

  Cette fonction définit le mode de fonctionnement de l'automate (c'est l'expression d'un certain algorithme). Elle définit dans quel état l'automate va passer quand il se trouve dans un état q et lit une lettre x (ou si il va éventuellement se bloquer). Le qualificatif déterministe correspond au fait que pour un couple (q, x) donné (constitué d'un état q et d'une lettre x) :
  - soit la fonction est définie et l'automate passe dans l'état unique q' spécifié par la fonction de transitions $(q' = \delta(q, x))$;
  - soit elle n'est pas définie et l'automate se bloque ;
- $q_0 \in Q$ est **l'état initial** de l'automate $\mathcal{A}$. C'est l'état dans lequel l'automate se trouve lorsqu'il commence à travailler ;
- $Q' \subset Q$ est l'ensemble des **états terminaux** (ou **états finals**) de l'automate $\mathcal{A}$: il contient un certain nombre d'états distingués qui permettront d'identifier des calculs particuliers.

Avant de définir de manière formelle le fonctionnement d'un tel automate, disons que le calcul de l'automate $\mathcal{A}$ pour un mot w de X^* de longueur n ($\mid w \mid = n$ et $w = x_1 \dots x_n$), consistera en la suite de m états ($m \leq n$), q_0 (l'état initial de l'automate), q_1, ... , q_m où pour chaque i ($1 \leq i < m$), $q_i = \delta(q_{i-1}, x_i)$.

Dans le cas où l'automate se bloque (du fait de la non-définition de la fonction δ en un point du calcul), on a $m < n$.

Le mot w est reconnu si et seulement si $m = n$ et $q_m \in Q'$ (c'est-à-dire si après avoir lu les $\mid w \mid$ lettres de w de gauche à droite, l'automate atteint un état terminal).

L'ensemble de tous les mots qui sont reconnus par $\mathcal{A}$ est appelé **langage reconnu par l'automate** $\mathcal{A}$ et est noté $\mathrm{L}(\mathcal{A})$.

Pour formaliser le calcul d'un automate fini déterministe $\mathcal{A}$, on définit l'extension δ^* de la fonction δ aux mots de X^* de la manière suivante :

- $\forall\ q \in Q,\ \delta^*(q, \epsilon) = q$
- $\forall\ q \in Q, \forall\ x \in X, \forall\ u \in X^*,\ \delta^*(q, ux) = \delta(\delta^*(q, u), x)$.

Un mot w de X^* est reconnu par l'automate fini déterministe $\mathcal{A}$ si $\delta^*(q_0, w) \in Q'$ (la fonction δ^* est donc définie en (q_0, w) et lui associe un élément de Q').

Le langage $\mathrm{L}(\mathcal{A})$ reconnu par l'automate est donc $\{u / u \in X^*$ tels que $\delta^*(q_0, u) \in Q'\}$.

se bloque, soit le calcul conduit à un état qui n'est pas terminal.

Premiers exemples et représentation graphique des automates

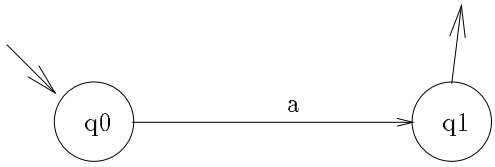
Les exemples suivants sont simples et visent de plus à introduire la représentation des automates sous forme de graphes.

Exemple 1.

Le langage L_1 `= {a}`, qui contient contient comme seul mot celui formé de la lettre `a` est reconnu par l'automate $\mathcal{A}_1$ défini comme le quintuplet :

$< Q_1 = \{q_0, q_1\}, X = \{a\}, \delta_1, q_0, \{q_1\} >$ avec $\delta_1(q_0, a)) = q_1$ (δ_1 non définie ailleurs).

Il est agréable de représenter un tel automate sous la forme d'un graphe :

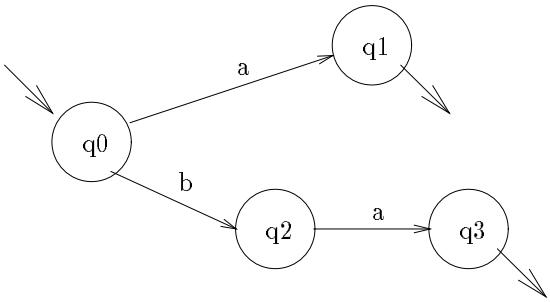


dans lequel l'état initial est marquée par une flèche entrante et un état terminal par une flèche sortante.

Le seul chemin dans le graphe conduisant de l'état initial q_0 à l'état terminal q_1 est de longueur 1 et ne contient que l'arc étiqueté a : le mot a est donc le seul mot reconnu par l'automate.

Exemple 2.

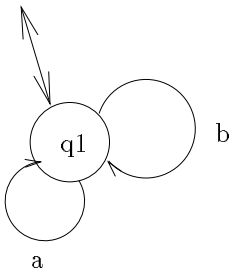
L'automate $\mathcal{A}_2$ représenté graphiquement ci-dessous reconnaît le langage fini L_2 `= {a,ba}` contenant les deux mots `a` et `ba` :



Il y a deux chemins menant de l'état initial à un état terminal :

- le chemin contenant l'arc de q_0 à q_1 étiqueté a et le mot a est donc reconnu ;
- le chemin de longueur 2 contenant l'arc de q_0 à q_2 étiqueté b et l'arc de q_2 à q_3 étiqueté a : le mot ba est donc reconnu.

L'automate ne reconnaît donc que les deux mots a et ba.



La flèche dans les deux sens utilisée dans la représentation graphique exprime le fait que l'état initial est également terminal.

De la construction d'un automate fini reconnaissant un langage donné

Les exemples précédents sont particulièrement simples.
Construire un automate fini (si tant est qu'il en existe un) pour un langage donné (c'est-à-dire pour le moment décrit sous forme littéraire ou sous forme énumérative) n'est cependant pas nécessairement aussi trivial (les langages n'étant pas eux-mêmes aussi simples que les précédents).

Lors de la construction d'un automate fini, il ne faut pas perdre de vue que l'ensemble des états en constitue la mémoire. Donc, lors de la construction d'un automate pour un langage donné, il est commode et recommandé d'attacher une sémantique à chaque état : un état est censé mémoriser une information sur ce qu'il a lu précédemment et il est donc recommandé de lui associer une propriété caractérisant les mots qui permettent de l'atteindre à partir de l'état initial. Cette approche est similaire à celle consistant à attacher à des points d'un programme des *invariants* (propriété que satisfont un certain nombre de variables du programme en ce point et qui est généralement exprimée sous forme d'une expression booléenne).

Les exemples suivants vont éclairer cette vision des choses.

Exemple 4.
Intéressons nous pour commencer au langage L_4 sur l'alphabet $\{a, b\}$ constitué exactement de tous les mots ayant un nombre impair d'occurrences de la lettre a.
Pour construire un automate pour ce langage, imaginons comment nous pourrions procéder à la main pour vérifier qu'un mot donné satisfait la propriété voulue :

- un être humain sachant compter pourrait, purement et simplement, compter le nombre des occurrences de la lettre a dans le mot à vérifier. Chaque nouvelle occurrence de la lettre a rencontrée lors de la lecture du mot provoquerait l'augmentation d'un compteur des occurrences de a, la lecture d'une occurrence de la lettre b étant sans effet sur la valeur du compteur. Une fois le mot lu entièrement, la réponse est fournie par la parité de ce nombre d'occurrences. En terme de programme, cela se traduirait par une fonction, écrite ici comme méthode statique en Java recevant une chaîne de caractères en paramètre et renvoyant un booléen vrai si la chaîne correspondante contient un nombre impair de a :

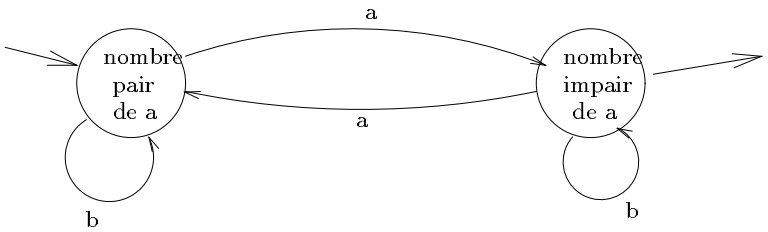
```
   // boucle de lecture des lettres successives du mot
   for (int i = 0; i < mot.length(); i++)
      // invariant : compteur est égal au nombre d'occurrences de a
      // rencontré dans le facteur gauche de longueur i
      if(mot.charAt(i) == 'a') compteur ++;
   return (compteur % 2 == 1);
}
```

Si on essaie d'en donner une formulation en termes d'automate, cela conduit à une machine qui doit mémoriser toutes les valeurs possibles du compteur, nombre entier a priori quelconque (et non borné). Cette solution ne permet pas de construire un automate fini (par contre on peut concevoir un automate à nombre infini d'états) ;

– pour arriver à une solution transposable en terme d'AFD, on peut procéder autrement et plutôt que de mémoriser le nombre d'occurrences de a déjà rencontrées, on peut utiliser une variable pouvant prendre deux valeurs (par exemple un entier pouvant prendre les valeurs 0 ou 1 ou mieux un booléen) mémorisant la parité du nombre d'occurrences de a déjà rencontrées. La variable est alors une simple bascule que la lecture de la lettre a fait changer de valeur :

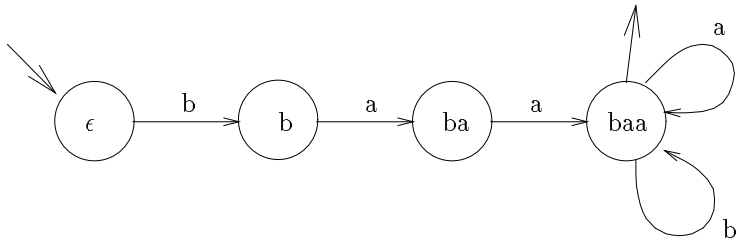
```
static boolean nbImpairDeA(String mot) {
   boolean bascule = false;  // nombre de a lus pair puisque nul
   // boucle de lecture des lettres successives du mot
   for (int i = 0; i < mot.length(); i++)
      // invariant : bascule est vrai ssi le nombre d'occurrences de a
      // rencontrées dans le facteur gauche de longueur i est impair
      if(mot.charAt(i) == 'a') bascule = !bascule;
   return bascule;
}
```

Transposé en terme d'automates, il suffit de deux états correspondant aux deux valeurs possibles (ou aux propriétés associées à chacune de ces valeurs), ce qui conduit à l'automate fini $\mathcal{A}_4$ suivant :



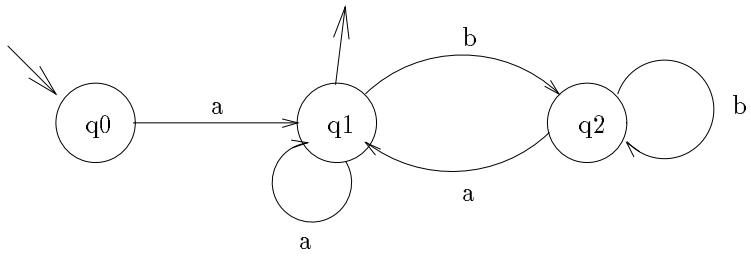
Exemple 5.

Dans l'exemple suivant, le langage L_5 est constitué des mots dont `baa` est facteur gauche (c'est-à-dire commençant par `baa`). On peut noter au passage que L_5 `= {baa}.{a,b}`*.
L'automate suivant a été construit en associant à chaque état le mot que l'automate a effectivement lu pour l'atteindre. Nous avons gardé comme nom de chaque état ce mot. Pour reconnaître un mot, l'automate partira d'un état où il n'a rien lu (ϵ), devra lire `b` et se retrouvera dans l'état nommé `b`, où il devra lire `a` ce qui le conduira en l'état nommé `ba`, et enfin devra lire `a` et se retrouvera en l'état `baa`. Dans cet état qui est terminal, la vérification que le mot donné commence par `baa` a été faite. Il pourra lire n'importe quelle lettre : le mot satisfera la condition désirée d'où la boucle sur cet état par lecture



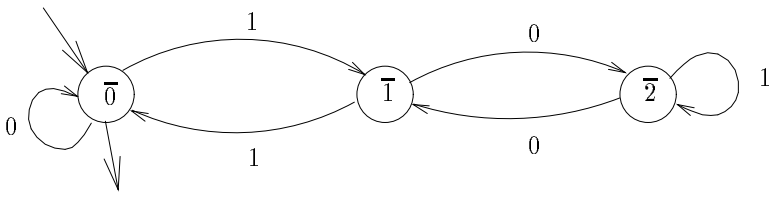
Exemple 6.

On s'intéresse ici au langage L_6 constitué de tous les mots qui commencent et finissent par la lettre `a`. Partant de son état initial q_0, il vérifie que la première lettre est `a` et si c'est le cas passe dans l'état q_1. Cet état symbolise le fait que l'automate a lu un mot commençant par `a` et finit par `a`. S'il lit la lettre `a`, le mot lu vérifie encore la propriété, mais s'il lit la lettre `b`, il ne vérifie plus la conditon sur sa dernière lettre (par contre la condition sur la première lettre reste vérifiée). Il passe donc dans l'état q_2 : pour repasser à l'état q_1, l'automate devra lire la lettre `a`. D'où l'automate $\mathcal{A}_6$ suivant reconnaissant le langage L_6 :



Exemple 7.

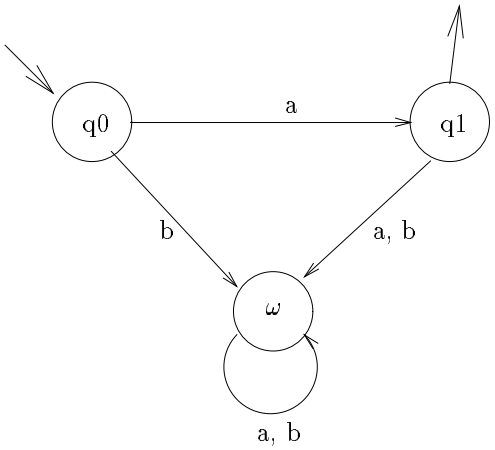
Ce dernier exemple vise à montrer que les automates finis, en dépit de leur simplicité, sont susceptibles de reconnaître des langages un peu moins simples que les précédents.
On souhaite construire un automate reconnaissant le langage L_7 sur l'alphabet $\{0,1\}$ contenant les mots vus comme des multiples de 3 écrits en base 2. De fait, cela est possible avec un automate à 3 états qui correspondent chacun à une des classes de congruence modulo 3 de l'ensemble $\mathbb{N}$ des entiers naturels. Son fonctionnement (et donc la définition de sa fonction de transitions) correspond à l'évaluation du polynôme associé au nombre binaire par la méthode dite de Horner (par lecture du polynôme à partir des coefficients de poids fort), le calcul se faisant dans $\mathbb{Z}/3\mathbb{Z}$ (c'est-à-dire modulo 3). Étant dans un état $\overline{a}$ (avec $a = 0$, 1 ou 2), à la lecture du caractère x (c'est-à-dire 0 ou 1), on passe à l'état $\overline{b}$ où $\overline{b}$ est la classe de congruence modulo 3 de l'entier $2 \times a + x$.
D'où l'automate $\mathcal{A}_7$ suivant reconnaissant le langage L_7 :



totale (i.e. partout définie).
Tout automate fini déterministe incomplet $\mathcal{A}$ peut être transformé en un automate fini déterministe complet $\mathcal{B}$ équivalent (c'est-à-dire reconnaissant le mme langage) en lui ajoutant un état ω évidemment non terminal (souvent appelé **puits** ou état poubelle) dont la fonction de transition $\delta_{\mathcal{B}}$ est définie par :

- $\forall\ (q,x) \in Q \times X$ tel que $\delta_{\mathcal{A}}(q,x)$ est défini, $\delta_{\mathcal{B}}(q,x) = \delta_{\mathcal{A}}(q,x)$
- $\forall\ (q,x) \in Q \times X$ tel que $\delta_{\mathcal{A}}(q,x)$ n'est pas défini, $\delta_{\mathcal{B}}(q,x) = \omega$
- $\forall\ x \in X$, $\delta_{\mathcal{B}}(\omega,x) = \omega$

Les automates $\mathcal{A}_3$, $\mathcal{A}_4$ et $\mathcal{A}_7$ des exemples précédents sont complets, les autres non.
L'automate suivant est un automate complet équivalent à l'automate $\mathcal{A}_1$:

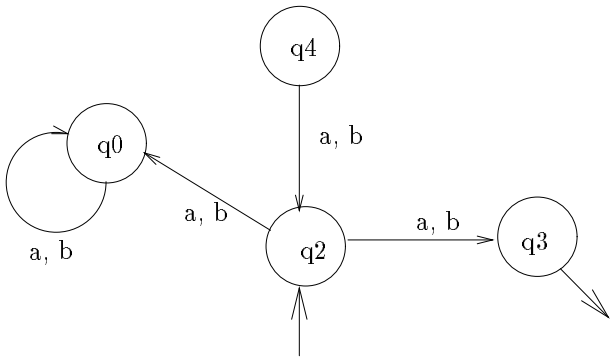


b) Automate fini déterministe émondé

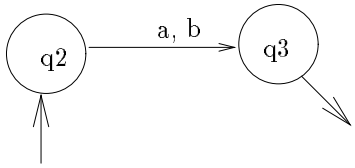
L'opération d'émondage vise à éliminer d'un automate les états qui ne servent à rien dans son calcul, c'est-à-dire de ne garder dans l'automate que

- les états qu'on peut atteindre depuis l'état initial (ces états sont dits **accessibles** ;
- les états à partir desquels on peut atteindre un état terminal (ces états sont dits **coaccessibles**.

Un automate émondé est un automate si tous ses états sont accessibles et coaccessibles.
Les sept automates $\mathcal{A}_1, \ldots, \mathcal{A}_7$ présentés sont émondés.
L'automate $\mathcal{A}_8$ suivant n'est pas émondé :



équivalent.
L'automate suivant est un automate émondé équivalent à $\mathcal{A}_7$:



La famille des langages reconnaissables

Un langage L est dit reconnaissable s'il existe un automate fini $\mathcal{A}$ tel que $L = L(\mathcal{A})$.
On note $\mathcal{R}\text{ec}(\text{X}^*)$ la famille des langages reconnaissables et la suite de ce cours est consacrée à l'étude des propriétés de cette famille de langages. Nous allons essayer en particulier de répondre aux deux questions suivantes :

- quelles propriétés particulières les langages reconnaissables possèdent-ils qui font que certains langages ne le sont pas. Les choses seraient plus agréables et plus simples si tous les langages l'étaient!
- existe-t-il d'autres méthodes de description des langages reconnaissables que les automates finis déterministes ?

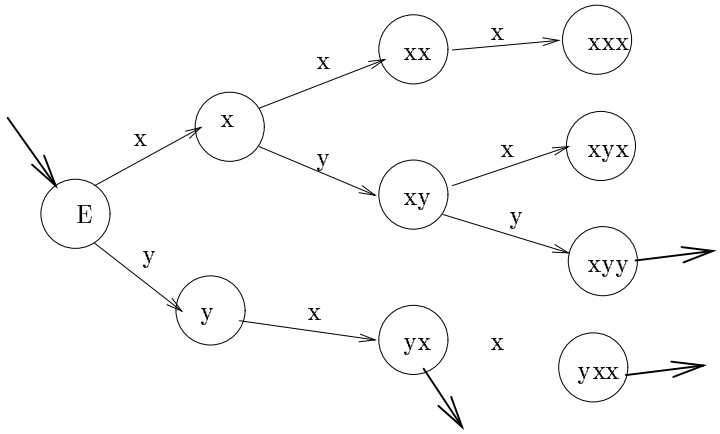
Première propriété : *tout langage fini est reconnaissable*
Si L est un langage fini, $Pref(L) = \{f \mid \exists w \in L$ et $\exists g \in X^*$ tels que $w = f.g\}$, c'est-à-dire le langage contenant exactement tous les facteurs gauches des mots de L, est lui-même fini.
Le langage L est alors reconnu par l'automate fini

$$< Pref(L) \cup \{\omega\}, X, \delta, \epsilon, L >$$

où la fonction de transitions δ est définie par :

- $\delta(u, x) = ux$ si $ux \in Pref(L)$
- $\delta(u, x) = \omega$ sinon

Exemple : soit L ={xx, xxx, xyx, xyy, yx,yxx}.
Le langage $Pref(L)$ de ses facteurs gauches est {ϵ,x,xx,xxx,xy,xyx,xyy,y,yx,yxx}.
Le langage L est reconnu par l'automate fini suivant :



Il faut évidemment les rechercher dans les langages infinis.
Un langage infini a la propriété qu'on ne peut pas borner la longueur des mots qu'il contient : cela signifie que quel que soit l'entier n, tout langage infini contient un mot de longueur supérieure ou égale à n.

Soit un langage L infini supposé reconnaissable.

Il existe donc un automate fini $\mathcal{A}$ le reconnaissant.
Soit m le nombre d'états de cet automate (q_0 état initial, q_1, ... , q_{m-1}).
Le langage L étant infini, il contient un mot w de longueur $n \geq m$: $w = x_1 \dots x_n$ (les x_i en sont les lettres successives).
Le calcul de l'automate $\mathcal{A}$ sur le mot w implique le passage successifs par les états q_0 (l'état initial), q_{i_1}, ... , q_{i_n} (q_{i_n} est terminal puisque le mot est reconnu par l'automate).
Si on se limite au travail sur le facteur gauche $u = x_1 \dots x_m$ de longueur m de w, l'automate passe successivement par les $m+1$ états : q_0, q_{i_1}, ... , q_{i_m}.
L'automate n'ayant que m états, parmi les $m+1$ états précédents, il en existe nécessairement deux qui sont identiques(principe des tiroirs) : soit $q_{i_j} = q_{i_k}$ ces deux états ($j < k$) :

- le passage de q_0 à q_{i_j} se fait par lecture du facteur gauche $f = x_1 \dots x_j$ de longueur j de w ;
- le passage de q_{i_j} à q_{i_k} (et donc à q_{i_j}) se fait par lecture du facteur $v = x_{j+1} \dots x_k$ de longueur ≥ 1 (puisque $j < k$) ;
- le reste du calcul de l'automate fait passer par les états q_{i_k}, ... , q_{i_m}, ... , q_{i_n} par lecture de gu'.

Ce qui est important, c'est que si, étant dans l'état q_{i_j}, on lit successivement plusieurs occurrences du mot v, l'automate revient en cet état.
Par suite tout mot de la forme fv^pgu' (où p est un entier quelconque) appartient au langage L.
On peut donc énoncer la **propriété** suivante (lemme d'itération simple) :

> Soit L un langage reconnaissable infini.
> Il existe un entier $n > 0$, tel que tout mot w de L tel que $\mid w \mid \geq n$ peut s'écrire (se factoriser) sous la forme $w = fvg$ avec $\mid fv \mid \leq n$ et $v \neq \epsilon$ et pour tout entier $k \geq 0$, $w = fv^kg$ appartient à L.

Il est important de noter que cette propriété constitue une **condition nécessaire** pour qu'un langage soit reconnaissable.
Elle n'est pas suffisante : il existe en effet des langages qui la satisfont bien que n'étant pas reconnaissables.
En conséquence ce lemme est utilisé pour montrer que certains langages ne sont pas reconnaissables comme dans ce qui suit.

naissable.

- tout d'abord ce langage est infini et s'il est reconnaissable, la propriété énoncée peut lui être appliquée. Soit l'entier $n > 0$ dont l'existence est affirmée par la propriété de reconnaissabilité ;
- considérons le mot $w = \mathtt{a}^n\mathtt{b}^n$. Il peut se factoriser en fvg, avec $\mid fv \mid \leq n$. Donc fv est un mot de la forme $\mathtt{a}^p$ et g est de la forme $\mathtt{a}^q\mathtt{b}^n$, avec $n = p + q$.
- on a : $f = \mathtt{a}^{p_1}$, $v = \mathtt{a}^{p_2}$, $p = p_1 + p_2$ et $p_2 \geq 1$, puisque $v \neq \epsilon$;
- le facteur v, c'est-à-dire $\mathtt{a}^{p_2}$ peut être itéré un nombre de fois quelconque sans qu'on sorte du langage, par exemple 2 fois.
  Le mot ainsi obtenu est $\mathtt{a}^{p_1}\mathtt{a}^{2\times p_2}\mathtt{a}^q\mathtt{b}^n$ et il doit appartenir à L ;
- cela signifie que $n = p_1 + 2 \times p_2 + q$, ce qui est impossible puisque $n = p_1 + p_2 + q$ et $p_2 \neq 0$;
- le langage L n'est donc pas reconnaissable (par un automate fini).

Remarque

Si on supprime l'hypothèse de finitude de l'ensemble des états, on peut construire un automate (à nombre infini d'états) reconnaissant le langage précédent.

Un automate infini reconnaissant le langage est défini par :

- l' ensemble d'états $\mathbb{N} \cup \mathbb{N}'$ où $\mathbb{N}'$ est une copie de $\mathbb{N}$ (ses éléments sont de la forme n' où n est un entier) ;
- *0* comme état initial ;
- *0* et *0'* comme états terminaux ;
- fonction de transitions δ définie par :
  - $\delta(n, a) = n + 1$ pour tout entier n ;
  - $\delta(n, b) = n' - 1$ pour tout entier $n > 0$;
  - $\delta(n', b) = n' - 1$ pour tout entier $n > 0$;
  - δ non définie ailleurs.