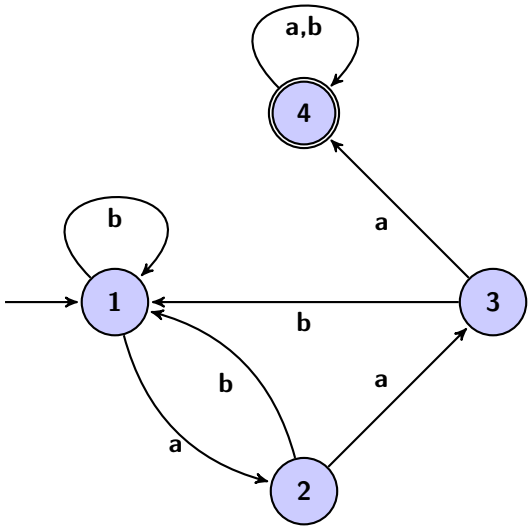
Automates finis

Il s'agit d'un modèle théorique utilisé un peu partout en informatique. Très grossièrement, il sert à représenter les divers états d'un système (mécanique, électronique ou abstrait) et les transitions entre ces états déclenchées par des actions extérieures. L'intérêt de ce modèle est qu'il est connu et étudié depuis les années 1950 et s'accompagne de nombreuses méthodes pour analyser le système modélisé.

1 Présentation

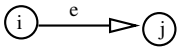
Le schéma ci-dessous représente un automate. On ne se préoccupe pas pour l'instant du rapport qu'il pourrait avoir avec un système réel, il va simplement servir à introduire les notions nécessaires par la suite.



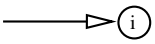
L'automate Le schéma ne représente pas un système en lui-même, mais les *évolutions* de ce système. En particulier, si on s'intéresse à un appareil (disons par exemple un distributeur de boissons), les cercles et les flèches n'ont pas une contrepartie matérielle dans cet appareil, comme des pièces mécaniques ou des composants électroniques.

- **État** 

Chaque état du système est représenté par un cercle (ou tout autre forme...). Les états doivent obligatoirement être en nombre fini. On attribue souvent un nom ou un numéro aux états pour pouvoir les repérer : dans cet exemple, les états sont numérotés de 1 à 4

- **Transition** 

Chaque transition possible entre deux états est représentée par une flèche étiquetée par le nom de l'action qui déclenche cette transition : quand le système est dans l'état i, l'action e le fait passer dans l'état j.

- **État initial** 

Un état particulier, en général repéré par une flèche sans origine, modélise l'état initial du système, avant qu'il ne subisse aucune action.

- **État(s) terminal(aux)** 

Un ou des état(s) particulier(s) représente(nt) la fin attendue du processus. On utilise un double cercle (ou toute autre convention) pour signaler un état terminal.

Les actions Pour décrire les actions extérieures et les successions d'actions subies par le système modélisé, on utilise une terminologie issue de l'algorithmique du texte (lettre, mot, etc.), car c'est historiquement un des domaines dont est issu le développement de la théorie des automates finis.

- **Lettres**
  - ⋆ Dans notre exemple, les lettres sont a et b. Autrement dit, le système n'est susceptible de subir que deux actions différentes. C'est évidemment une situation simplifiée à l'extrême par rapport à la plupart des systèmes réels.
  - ⋆ Dans les cas concrets, les « lettres » pourront être très variables selon le système considéré : détecter une pression sur la touche V, détecter l'introduction d'une pièce de 10cts, rencontrer un obstacle, lire la lettre m dans un fichier informatique, voir qu'un joueur avance un pion, etc.
  - ⋆ Le seul point important est que les lettres doivent être en nombre fini.
- **Mots**
  - ⋆ Dans cet exemple, un mot possible est *abaab* ou encore *bbababaaaaab*.
  - ⋆ Toute suite finie de lettres (i.e. d'actions) est un « mot ».
  - ⋆ Traditionnellement, les mots sont lus de gauche à droite.

Fonctionnement L'évolution du système modélisé sous l'effet d'une suite d'actions est visualisée par une série de déplacements d'état en état sur l'automate en suivant les transitions correspondantes.

- En particulier, à partir de l'état initial du sytème, toute suite finie d'actions (i.e. tout « mot ») amène à un certain état.
- Il y a deux sortes de suites d'actions : celles qui aboutissent dans un état terminal et les autres.
- Sur l'exemple, on vérifie que les mots qui conduisent dans l'état 4 (terminal) sont ceux qui contiennent (au moins) 3 lettres a consécutives et eux seuls. Par exemple *baabbaaaababab*, ou *aaaaaa*, ou *bbbbabbaaa* aboutissent dans l'état 4, mais pas *ababbaa*, ni *aaba*.
- Par conséquent, notre automate peut-être vu comme la description visuelle d'une méthode permettant de *reconnaître* à coup sûr, parmi tous les mots (constitués de a et de b) possibles, ceux qui contiennent *aaa*.
- C'est le concept théorique sous-jacent à la notion d'automate fini : il s'agit d'une façon de décrire un procédé permettant de séparer des autres un ensemble de mots définis par le fait qu'ils vérifient une propriété commune (par exemple contenir la suite de lettres *aaa* dans notre cas).

Description Une propriété importante des automates finis est que ce sont des objets qui possèdent une description finie (éventuellement très grande, s'il y a beaucoup d'états et de lettres). En effet, un automate *fini* possède un nombre fini d'états et un nombre fini de lettres : il peut donc être entièrement décrit à l'aide d'un tableau comme ci-dessous. En ce sens, les automates finis sont des objets simples, et en particulier des objets qui peuvent être analysés et traités par des logiciels. On note cependant que, malgré cette description finie, l'automate est susceptible de reconnaître des mots de longueur arbitrairement grande, par exemple $baab\underbrace{ab\ldots ab}_{\substack{\text{répété}\\ n \text{ fois}}}aaabab$ pour notre exemple.

	a	**b**
1	2	1
2	3	1
3	4	1
4	4	4

Lecture du tableau : Quand le système se trouve dans l'état 2, et qu'il subit l'action a (par exemple), alors il entre dans l'état 3 : c'est ce qu'indique le 3 qui se trouve à l'intersection de la troisième ligne (étiquetée ligne 2) et de la deuxième colonne (étiquetée colonne a).

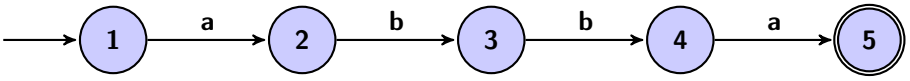
Réalisation concrète À partir du tableau ci-dessus, il n'est pas difficile de fabriquer un équivalent électronique de l'automate ou d'écrire un programme reconnaissant les mêmes mots. Il existe même des logiciels qui le font automatiquement.

2 Exemples

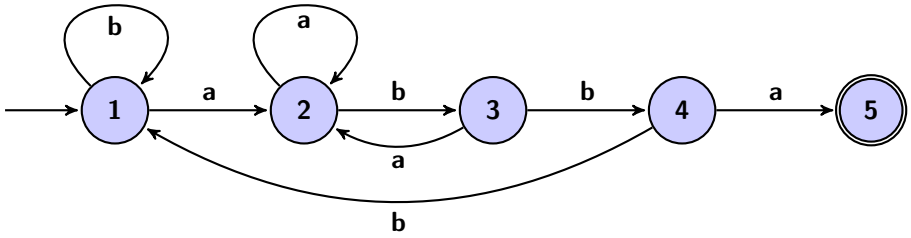
Un digicode On imagine pour simplifier un digicode à 2 touches (a et b), qui ouvre une porte lorsqu'on tape le code *abba* (par exemple). Toujours pour simplifier, on suppose que la porte s'ouvre sitôt qu'on tape une suite de lettres qui se termine par *abba*. On montrera ensuite une version plus réaliste. Pour l'instant, parmi toutes les suites de lettres a et b possibles, il s'agit donc de reconnaître celles qui se terminent par *abba* : on va voir qu'il est possible de modéliser le fonctionnement de cette machine à l'aide d'un automate fini.

Chaque état de l'automate va correspondre à la mémorisation de la partie du code *abba* qui a été frappée jusque là (ou de façon équivalente, de celle qui reste à frapper). L'état initial 1 correspond à la porte fermée, lorsqu'aucune touche n'a encore été frappée. Le système attend alors la totalité du code. Puis l'état 2 mémorise le fait que la lettre a vient d'être frappée, et donc *bba* reste attendu. L'état 3 est atteint lorsque la suite *ab* vient d'être tapée et qu'on n'attend plus que *ba*, puis l'état 4 lorsque ce sont respectivement les suites *abb* et a. Enfin,

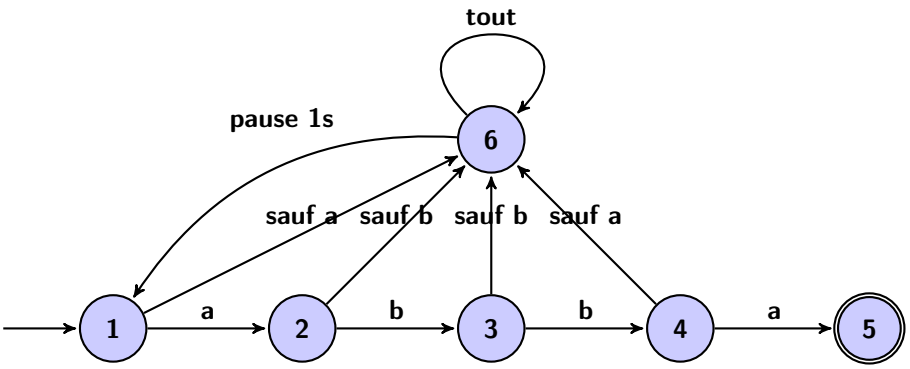
on accède à l'état 5 lorsque le code *abba* a été frappée en entier, on n'attend alors plus rien, et cet événement déclenche l'ouverture de la porte. L'état 5 sera donc l'unique état terminal de l'automate.
À ce stade, on obtient le schéma suivant :



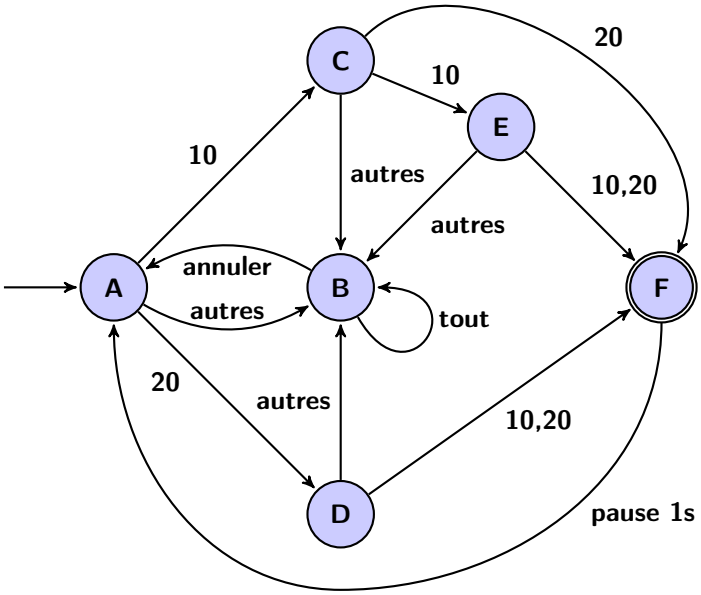
Il reste à compléter les transitions correspondant aux cas où les lettres frappées ne sont pas celles du code *abba*. Au départ, le système attend dans l'état 1 qu'on tape un a, qui est la première lettre du code. C'est pourquoi on met une transition étiquetée b qui boucle de l'état initial sur lui-même : tant que l'utilisateur ne tape pas un a, l'état du système ne change pas (il attend toujours un a). Ensuite, dans l'état 2, on sait que la lettre a a déjà été tapée, mais tant que l'utilisateur ne tape pas un b, l'écriture du code n'avance pas : c'est la raison de la boucle étiquetée a sur cet état. Dans l'état 3, c'est la touche b qui est attendue, mais si l'utilisateur tape un a, le système retourne à l'état 2 : c'est comme si on recommençait à écrire le code depuis le début. De même si on tape un b lorsque le système se trouve dans l'état 4 : il faut alors tout recommencer, y compris le a initial, donc on retourne à l'état 1.
Finalement, voici le schéma correspondant à ce digicode simplifié :



Cependant, les digicodes ont en général plus de 2 touches (donc il y a d'autres lettres que a et b à prendre en compte), d'une part, et souvent se bloquent quelques instants lorsqu'on se trompe en tapant le code, d'autre part. Pour prendre en compte ce dernier comportement, on introduit un nouvel état 6 qui mémorise le fait que l'utilisateur a fait une erreur en tapant le code, et dont on ne sort qu'après une pause d'une seconde (par exemple). Cette pause correspond à une nouvelle « lettre », qui n'est pas sur le clavier. Pour tenir compte des touches du clavier autres que a et b sans pour autant trop compliquer le schéma, on a simplement étiqueté les transitions avec `tout` pour indiquer que toutes les touches ont le même effet, et `sauf a` ou `sauf b`, pour indiquer que toutes les touches sauf a (ou b) ont le même effet.
Le schéma obtenu est le suivant :

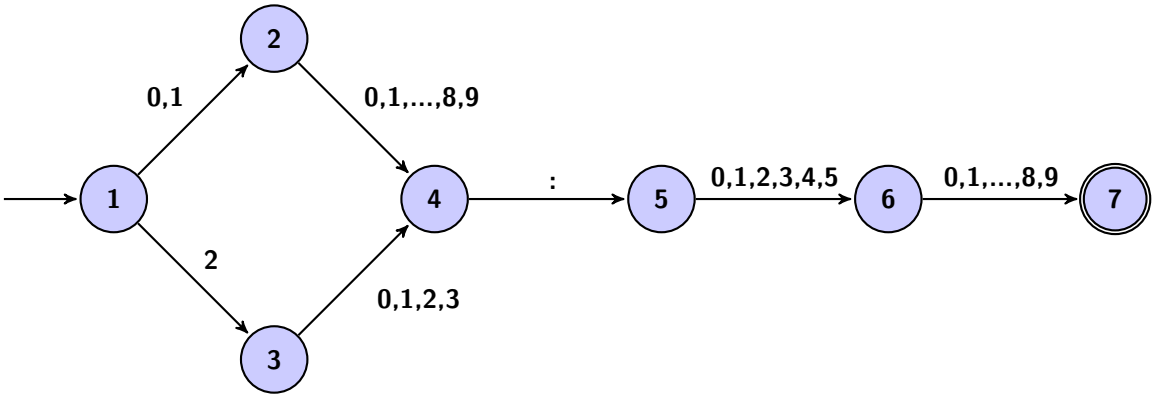


Une machine à café De nombreuses machines mécaniques ou électroniques peuvent être représentées par des automates finis en suivant un raisonnement du même type que celui qui précède. La plupart des automates obtenus dans la vie réelle sont évidemment beaucoup plus compliqués que ceux qu'il est possible de présenter ici. À titre d'exemple supplémentaire, on donne ci-dessous un automate correspondant à une machine à café simplifiée. Elle n'accepte que les pièces de 10 centimes et 20 centimes. Un café coûte 30 centimes. Quand le montant est suffisant (état F), elle délivre le café, rend éventuellement la monnaie et retourne dans l'état initial après une attente d'un seconde. Si on glisse une pièce d'un autre type que 10 ou 20 centimes (transitions *autres*), la machine passe dans l'état B, dont on ne peut sortir qu'en appuyant sur la touche *annuler*, qui déclenche le rendu des pièces et le retour à l'état initial.



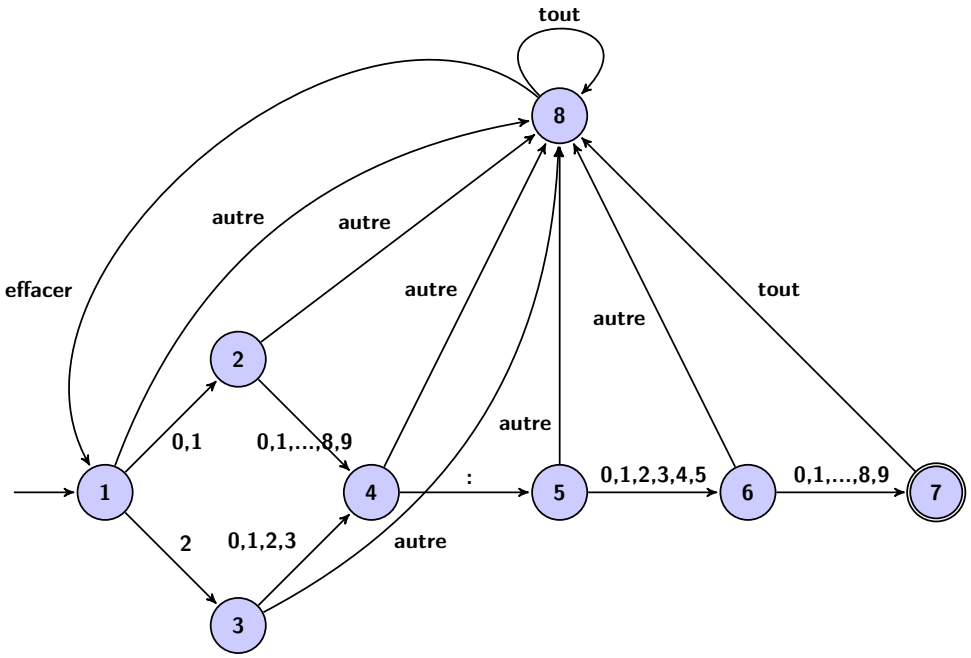
Vérificateur de format d'horaire Un autre domaine où la modélisation à l'aide d'automates finis est très utilisée est l'algorithmique du texte au sens large : recherche documentaire, traitement de texte, génomique, compilation, etc. On parle alors de recherche de motifs ou d'analyse syntaxique selon les domaines d'application. On détaille ci-dessous un exemple simple utilisé dans les activités proposées pour les élèves.

On s'intéresse à un formulaire sur un site internet qui demande d'entrer un horaire sous la forme `HH:MM` et vérifie si la valeur donnée est correcte avant de donner accès à la touche `Entrée`. Les horaires acceptables vont de 00 : 00 à 23 : 59, par conséquent des expressions comme 15 : 44 ou 08 : 00 sont correctes, mais pas 5 : 12, ni 42 : 05, ni 06 : 75, ni 04 : 255. Le fonctionnement du programme de vérification de la valeur donnée par l'internaute peut être modélisé par l'automate ci-dessous.

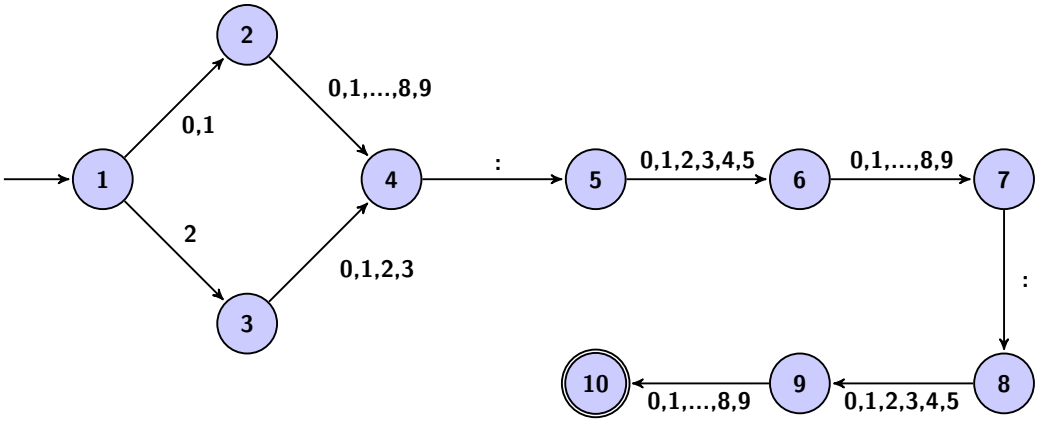


Détaillons le schéma : il faut impérativement deux chiffres, suivis du signe `:`, puis encore deux chiffres. Le premier chiffre (celui des dizaines des heures) ne peut prendre que les valeurs 0, 1 ou 2. Si c'est un 0 ou un 1, alors le chiffre des unités correspondant peut prendre n'importe quelle valeur entre 0 et 9, mais si c'est un 2, alors les seuls chiffres des unités possibles sont 0, 1, 2 et 3. Ensuite, le signe `:` est obligatoire. Pour les minutes, les seuls chiffres des dizaines possibles vont de 0 à 5, tandis que le chiffre des unités est quelconque.

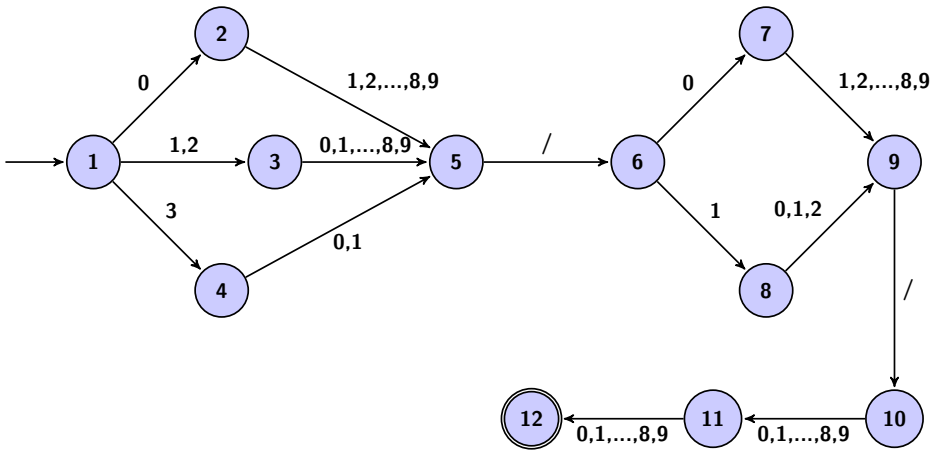
Pour aboutir à un modèle plus réaliste, il nous reste à tenir compte du fait que le clavier de l'ordinateur possède d'autres touches que les chiffres et le signe `:`, d'une part, et que le formulaire propose probablement une possibilité d'effacer en cas de fausse manœuvre, d'autre part. On introduit, comme dans le cas du digicode, un état supplémentaire mémorisant le fait qu'une touche non autorisée vient d'être frappée, et dont on ne peut sortir qu'en cliquant sur `effacer`, ce qui ramène à l'état initial du système (il faut donc recommencer à frapper son horaire). On aboutit alors à l'automate suivant :



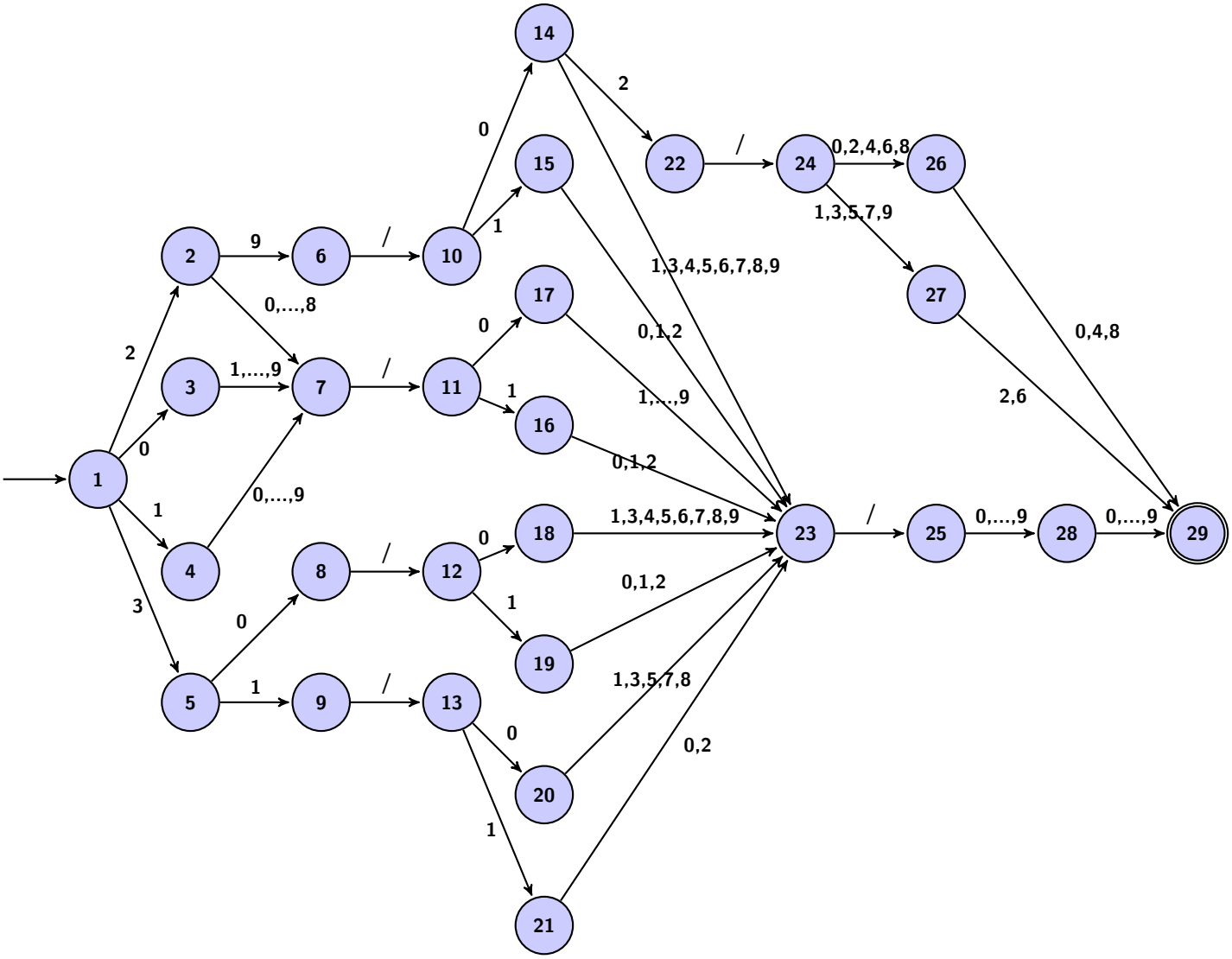
La variante ci-dessous tient compte des secondes : on attend un format `HH:MM:SS`, avec des valeurs allant de 00 : 00 : 00 à 23 : 59 : 59.



Vérificateur de date On obtient bien sûr un automate très similaire au précédent si on s'intéresse à la vérification d'une date (au format `JJ/MM/AA` par exemple) au lieu d'un horaire. Si la nature était coopérative et si tous les mois avaient 31 jours, on obtiendrait le schéma ci-dessous, et il ne serait pas nécessaire de s'attarder.



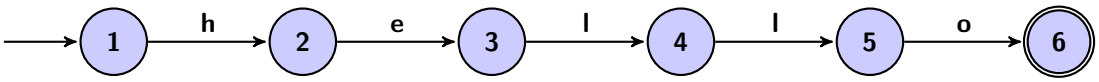
Cependant, le calendrier réel est bien plus compliqué. Il faut tenir compte des mois qui ont 30 jours, du mois de février qui n'a que 28 jours, et 29 jours les années bissextiles. L'année 2000 étant bissextile, contrairement aux autres années multiples de 100 (1900 et 2100 par exemple), on va se limiter à un calendrier valable du 1er janvier 2000 au 31 décembre 2099.



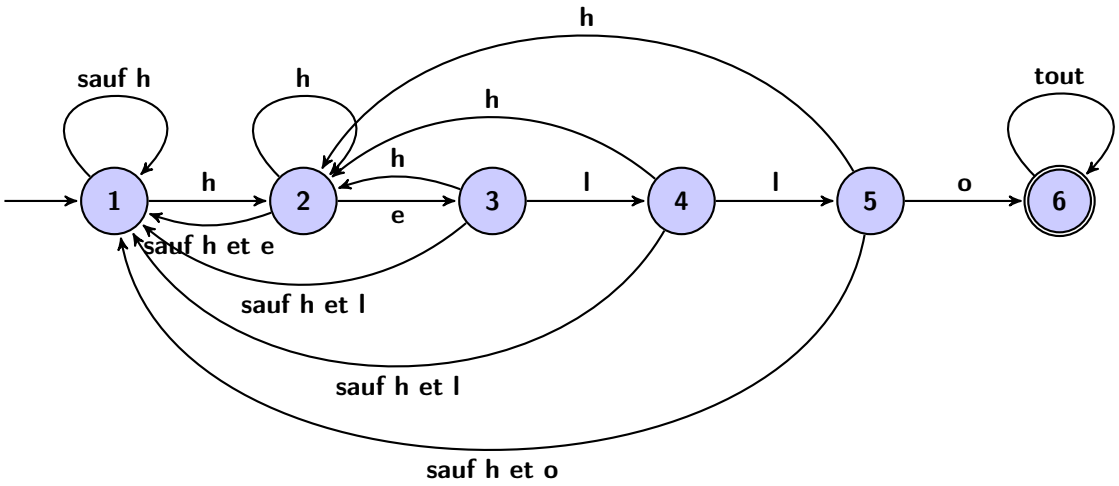
Pour comprendre ce schéma, on repère les passages obligés par les signes / qui correspondent à la séparation entre les jours et les mois d'une part (entre les états 6 et 10, 7 et 11, 8 et 12 et 9 et 13) et entre les mois et les années d'autre part (entre les états 22 et 24 et 23 et 25).
La partie la plus simple de l'automate est celle qui va de l'état 1 à l'état 7, puis de l'état 11 à l'état 23 et enfin de l'état 25 à l'état 29. Il s'agit du cas le plus général : les jours 01 à 28 se produisent pour les mois 01 à 12 et les années 00 à 99. Tout le reste de l'automate correspond au traitement des cas particuliers.
Le numéro de jour 30 se produit pour tous les mois sauf celui de numéro 02, et ceci pour n'importe quelle année : c'est la partie qui part de l'état 1, et atteint l'état 23 en passant par l'état 8, puis rejoint l'état 29. Par contre, le numéro de jour 31 ne se produit que certains mois (passage par l'état 9), et ceci pour n'importe quel numéro d'année (passage par l'état 23).
Il reste à traiter le numéro de jour 29 (passage par l'état 6) : pour n'importe quel mois sauf celui de numéro 02, il se produit pour n'importe quelle année (état 23). Pour le mois 02, on passe par l'état 22, et il reste à restreindre les numéros d'années permettant d'atteindre l'état terminal 29 à ceux des années bissextiles (c'est-à-dire ceux qui sont des multiples de 4).

Chercher `hello` Le second type d'exemples dans cette série concerne la fonction recherche d'un traitement de texte. Imaginons qu'on veuille savoir si la chaîne de caractères `hello` se trouve dans un fichier texte. Pour cela, on va parcourir le fichier caractère par caractère. Comme dans le cas du digicode, on va utiliser une série d'états pour mémoriser les lettres du début du mot recherché qui ont déjà été rencontrées (et par conséquent

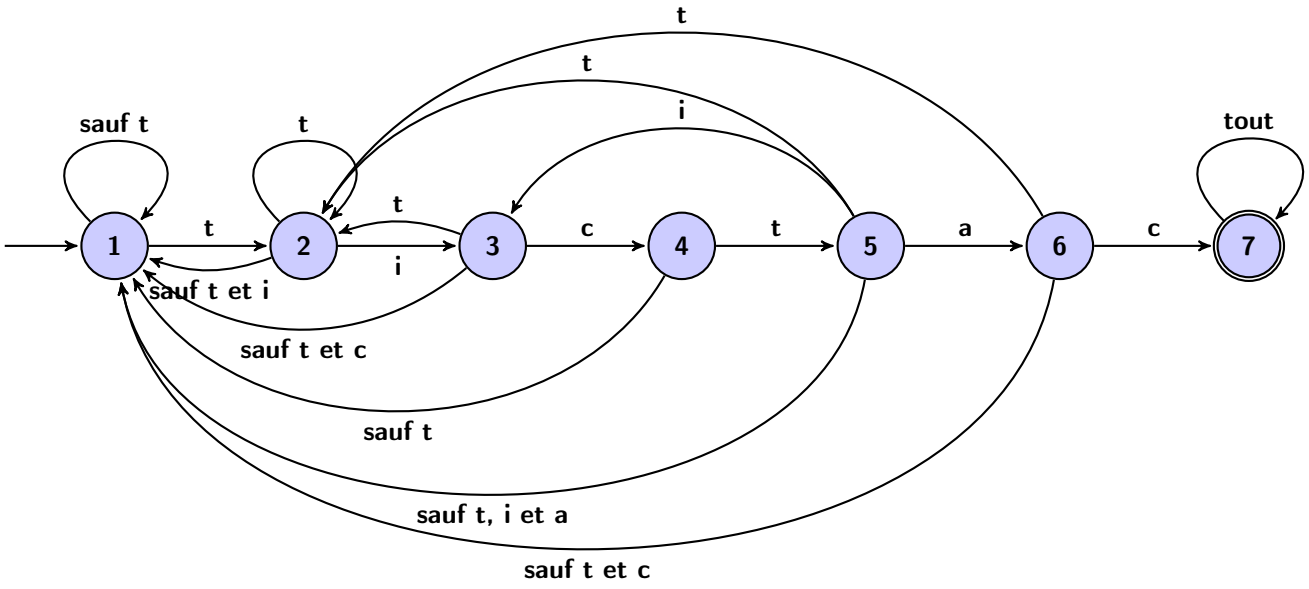
celles qu'on attend encore). On commence donc par le schéma ci-dessous :



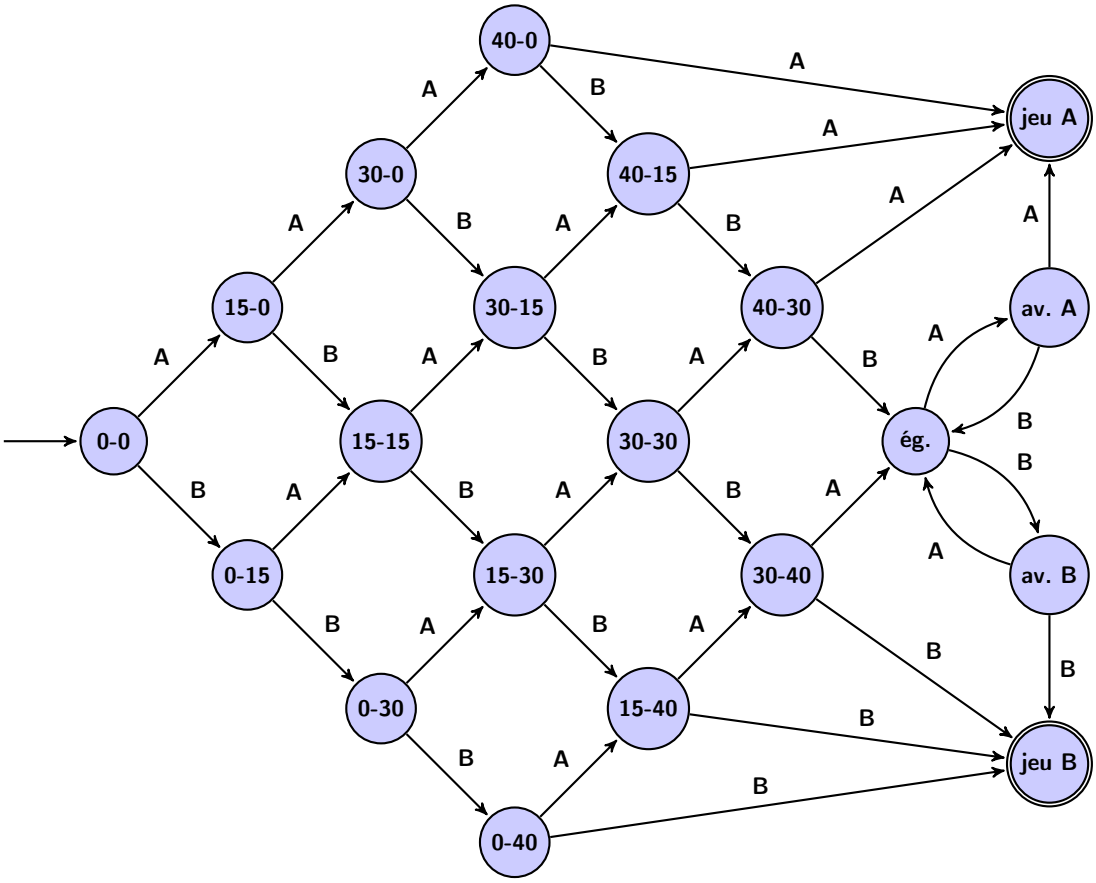
Les transitions restantes sont ajoutées en tenant compte du fait que chaque fois qu'on rencontre un `h`, on reprend la recherche à partir de la seconde lettre de `hello`. On attend dans l'état initial jusqu'à rencontrer un `h`. De chacun des états 2, 3, 4 et 5, il part trois transitions : l'une fait avancer d'une lettre dans le mot recherché, une autre revient à l'état 2 lorsqu'on rencontre un `h` et la troisième renvoie à l'état initial lorsque la lettre qu'on rencontre n'est aucune des deux précédentes. Enfin, lorsque le mot `hello` a été trouvé, il ne reste qu'à parcourir le reste du fichier sans plus se préoccuper de ce qui y est écrit, d'où la boucle étiquetée `tout` sur l'état terminal 6.



Chercher `tictac` On a une situation un tout petit peu plus compliquée si le début de l'expression à chercher se retrouve aussi à l'intérieur de l'expression elle-même : par exemple la lettre `t` dans le mot `tictac`. En effet, quand on rencontre le second `t` de `tict` (état 5), il peut être suivi d'un `a`, auquel cas il faut continuer vers l'état 6, ou bien d'un `i`, auquel cas il faut revenir vers l'état 3 (et non 2), puisqu'on vient de rencontrer `ti`. Il y a donc dans ce cas une transition supplémentaire. Il y a des cas encore plus compliqués, comme par exemple la recherche du mot `ananas`.



Le tennis Le dernier type d'exemples de modélisation d'un système à l'aide d'automates finis concerne des processus complètement abstraits, comme les différentes étapes du scénario d'un jeu. On illustre ce domaine par un automate modélisant l'évolution du score au cours d'un jeu pendant une partie de tennis. Les états correspondent aux différents scores possibles (comme $40-15$, ou *avantage* A), et les lettres A ou B représentent un point marqué par le joueur A ou B respectivement. On obtient le schéma ci-dessous.



Concrètement, lors de la conception d'un nouveau jeu (par exemple un futur jeu sur console, mais pas seulement), on peut modéliser à l'aide d'automates de ce type les différents scénarios envisagés, ou certaines parties de ces scénarios. D'une façon plus générale, les diagrammes d'utilisation que produisent les concepteurs de logiciels pour analyser les futures interactions avec les usagers, ou bien entre les différents modules d'un logiciel, sont aussi très souvent des automates finis construits selon un raisonnement similaire à celui que nous venons de présenter.

3 À quoi servent les automates ?

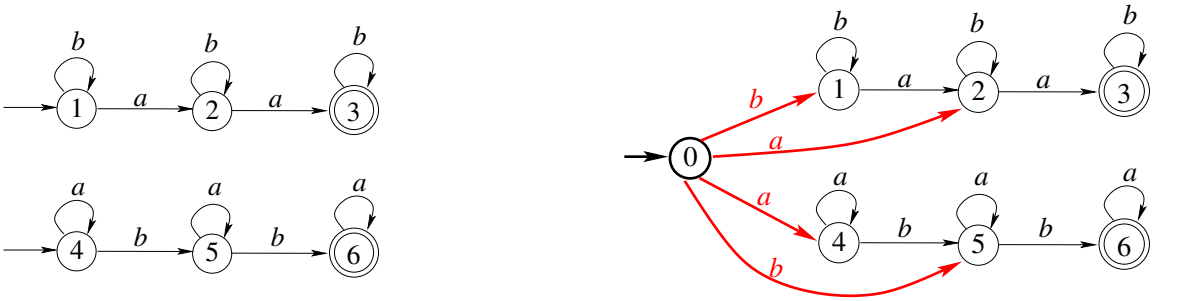
On a vu que le modèle automate fini est très souple et s'adapte à des domaines très différents. Ce qu'on n'a pas encore évoqué, c'est l'objectif de ces modélisations. Le but de notre dernier paragraphe est de donner quelques éléments à ce propos. On aborde ici des sujets plus techniques, aussi on se contentera d'évoquer des pistes très générales sans entrer dans des explications techniques.
Le premier point à prendre en compte est le fait que les automates correspondant à des situations réelles sont en général beaucoup plus gros et compliqués que les exemples que nous avons présentés. Il en résulte qu'on ne peut en règle générale pas les visualiser globalement, ni les analyser à la main. On utilise à la place des logiciels qui effectuent automatiquement les différentes opérations décrites ci-dessous.

Aide à la fabrication Tout d'abord, on peut réaliser concrètement un automate : il existe des méthodes permettant de produire automatiquement, à partir de la description d'un automate (par exemple sous forme de tableau), un programme reconnaissant les mêmes mots. On peut ainsi écrire un logiciel qui compte les points au tennis et fait retentir un signal sonore quand un des joueurs a gagné le jeu ou fabriquer un véritable digicode. À titre d'illustration, voici un programme en langage Python correspondant au digicode détaillé au début du paragraphe 2. Pour notre propos, il n'est pas nécessaire de l'étudier en détail, mais une observation attentive

permet de retrouver les transitions de l'automate : par exemple, lorsque le système est dans l'état 2, si la lettre tapée est un b, il passe dans l'état 3 (lignes 11 à 13).

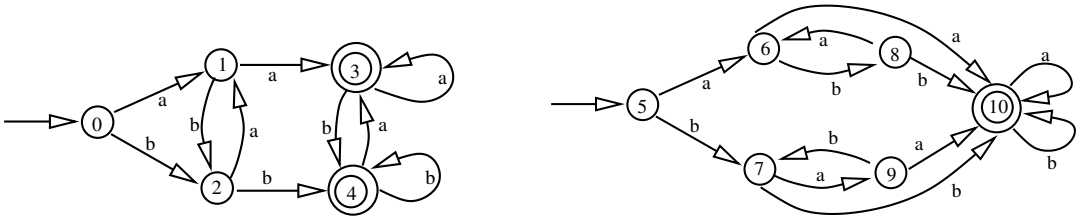
```
etat=1
lettre=input("Tapez a ou b : \n")
continuer=1
while continuer== 1 :
    if etat==1 :
        if lettre=="a" :
            etat=2
            lettre=input("Tapez a ou b : \n")
        else :
            lettre=input("Tapez a ou b : \n")
    elif etat==2 :
        if lettre=="b" :
            etat=3
            lettre=input("Tapez a ou b : \n")
        else :
            lettre=input("Tapez a ou b : \n")
    elif etat==3 :
        if lettre=="b" :
            etat=4
            lettre=input("Tapez a ou b : \n")
        else :
            etat==2
            lettre=input("Tapez a ou b : \n")
    elif etat=4 :
        if lettre=="a" :
            etat=5
        else :
            etat=2
            lettre=input("Tapez a ou b : \n")
    else :
        continuer=0
print("Vous avez tape abba")
```

Réutilisation Chaque set au tennis se joue en plusieurs jeux, et chaque match en plusieurs sets. Pour modéliser entièrement les évolutions possibles du score au cours d'une partie de tennis, il est intéressant de réutiliser l'automate qu'on a déjà construit pour le cas d'un seul jeu, au lieu de reprendre entièrement la construction depuis le début.
Pour illustrer comment on peut se servir d'automates déjà conçus pour en créer d'autres plus compliqués, sur le schéma ci-dessous, on a dessiné à gauche en haut un automate qui reconnaît les mots qui contiennent exactement deux fois la lettre a (et des lettres b en nombre quelconque, avant, après ou entre les deux a). À gauche en bas, l'automate reconnaît les mots qui contiennent exactement deux fois la lettre b. Sur le schéma de droite, on montre la première étape de la combinaison de ces deux automates pour en obtenir un qui reconnaît les mots qui contiennent ou bien exactement deux fois la lettre a, ou bien exactement deux fois la lettre b. On retrouve bien sur le schéma de droite les deux automates de gauche. On voit toutefois qu'il y a un problème, puisque à partir de l'état initial partent deux flèches étiquetées a et b au lieu d'une seule. Le reste du travail consiste à modifier le schéma pour éviter ce phénomène, mais nous ne l'aborderons pas ici.



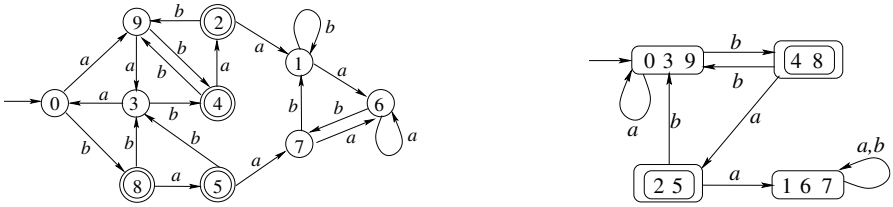
De telles méthodes systématiques de combinaisons d'automates permettent de construire des modèles de plus en plus volumineux à partir de briques initiales relativement simples.

Comparaison Supposons qu'on ait obtenu par deux méthodes différentes les deux automates ci-dessous pour la modélisation d'un même système.



On a évidemment besoin de savoir si l'un des deux est incorrect ou bien si on a simplement affaire à des automates qui reconnaissent exactement les mêmes mots. Ce n'est souvent pas évident, même sur des petits exemples comme ici, et on imagine facilement la difficulté de cette question pour des automates de quelques centaines d'états. Il est donc très utile qu'il existe une méthode programmable permettant de répondre à la question. Dans ce cas particulier, les deux automates reconnaissent bien exactement les mêmes mots, mais nous demanderons au lecteur de bien vouloir nous croire sur parole.

Simplification Après avoir modélisé un système, on se demande souvent s'il n'aurait pas été possible de faire plus simple (c'est-à-dire d'utiliser moins d'états). Il peut en effet arriver que certains états soient inutiles car inaccessibles, ou encore que certains états soient en réalité identiques, même si on ne s'en est pas rendu compte à la construction. Par exemple l'automate ci-dessous à gauche peut-être simplifié en celui de droite car certains états sont en réalité les mêmes. Comme on le voit sur le schéma de droite, il s'agit des états 0, 3 et 9 d'une part, des états 4 et 8, mais aussi 2 et 5 et enfin des états 1, 6 et 7. Ici encore, une procédure automatique, que nous ne détaillerons pas, permet de trouver l'automate le plus simple possible équivalent à un automate donné.



Vérifier des propriétés Il existe enfin des méthodes permettant de tester diverses propriétés du système à partir de son modèle automate. Par exemple, si on a écrit le scénario d'un jeu, il est souhaitable de s'assurer qu'il n'a pas un défaut qui fait que les parties ne se terminent jamais, ou qu'un des joueurs ne peut jamais gagner une partie. Ici encore, il est possible de s'en assurer en effectuant certains calculs sur l'automate correspondant au scénario. On peut par exemple vérifier que l'automate reconnaît au moins un mot (le contraire voudrait dire que l'état terminal "le jeu est fini" ne serait jamais atteint). De même, on peut savoir si un automate possède des boucles dont il est impossible de sortir. Pour mesurer l'intérêt de cette propriété, il suffit d'imaginer que l'automate modélise le fonctionnement d'un répondeur téléphonique interactif qui n'aboutirait jamais à vous mettre en relation avec un interlocuteur humain.

Ce qui est trop compliqué pour un automate Le dernier point qu'il faut évoquer ici est très important mais beaucoup plus subtil. Il se trouve qu'il est possible de déterminer si un système est ou n'est pas modélisable par un automate fini. On n'a parlé jusqu'ici que de ce qu'il est possible de modéliser avec un automate. Mais une des forces de ce modèle est qu'on connait aussi précisément ses limites.
Sans entrer dans les détails, signalons par exemple qu'on peut montrer qu'il n'existe aucun automate fini qui reconnaisse tous les mots qui contiennent autant de a que de b, quelle que soit leur longueur. Ainsi, les mots *ababbbaa* ou *aaaaaabbbbbb* seraient reconnus, mais pas *aab*, ni *bbbbabab*. Plus généralement, à cause de sa mémoire finie (correspondant à l'ensemble des états), un automate de ce type n'est pas capable de compter au-delà d'une certaine valeur fixée. Concrètement, cela montre qu'on ne peut pas modéliser à l'aide d'un automate fini les machines qui vérifient (disons) à la piscine que le nombre de sorties à la fin de la journées est bien égal au nombre d'entrées. On aurait besoin pour cela d'un modèle plus puissant que celui des automates finis, mais ce serait le début d'une autre histoire.