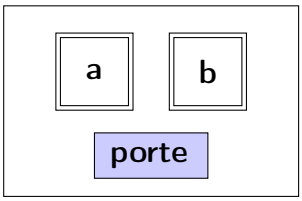
Automates finis

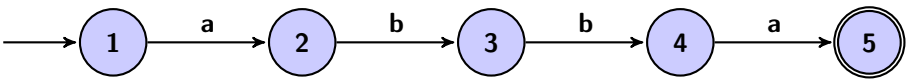
Il s'agit d'un modèle très souple, qui s'adapte à des domaines très différents en informatique. D'une façon générale, il sert à représenter les divers états d'un système (mécanique, électronique ou abstrait) et les transitions entre ces états. L'intérêt de ce modèle est qu'il s'accompagne de nombreuses méthodes pour analyser le comportement du système modélisé.

1 Quelques digicodes

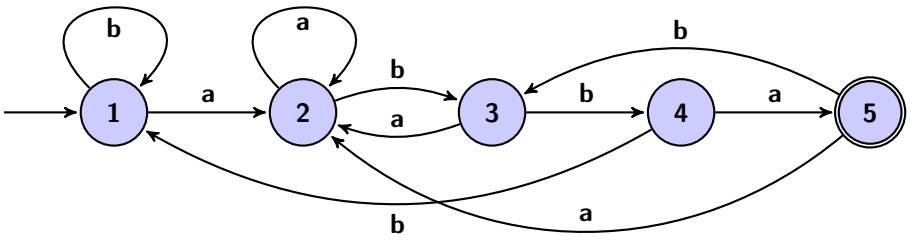
Un premier digicode On imagine une sorte de digicode à deux touches (marquées a et b), qui débloque le bouton d'ouverture d'une porte lorsqu'on frappe n'importe quelle suite de lettres qui se termine par *abba*, et seulement l'une de celles-ci. Ainsi, les suites *aaabba* ou *abababba* ou évidemment *abba* vont convenir, mais pas les suites *abb* ou *aabbaba*. Par conséquent, on peut dire que, parmi toutes les suites de lettres a et b possibles, ce digicode a pour rôle de **reconnaître** celles qui se terminent par *abba*. Nous allons dessiner un schéma appelé un **automate fini** pour modéliser le comportement de cette machine.



Chaque **état** de l'automate symbolise la partie de la suite *abba* qui a été frappée jusque là (ou, de façon équivalente, celle qui reste à frapper). Lorsqu'aucune touche n'a encore été frappée, le système se trouve dans l'**état initial** 1, et attend la totalité de la suite *abba*. Puis l'état 2 symbolise 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 la suite *abba* a été frappé en entier, on n'attend alors plus rien. L'état 5 sera donc l'unique **état terminal** de l'automate. Les **transitions** permettant de passer d'un état au suivant correspondent à la frappe d'une touche, et sont **étiquetées** par la lettre correspondante.
Sur le plan fonctionnel, le bouton d'ouverture de la porte est bloqué lorsque l'automate se trouve dans les états 1 à 4 et débloqué dans l'état 5.
À ce stade, on obtient le schéma suivant :

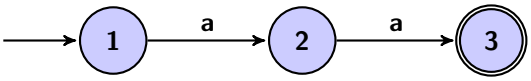


Il reste à compléter les transitions correspondant aux cas où les lettres frappées ne forment pas la suite *abba*. Au départ, le système attend dans l'état 1 qu'on tape un a, qui est la première lettre de *abba*. C'est pourquoi on met une transition étiquetée b qui boucle de l'état initial sur lui-même : 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 que des a et pas un b, l'écriture de *abba* 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 avait recommencé à écrire *abba* 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. Enfin, on rend compte de ce qui se produit si l'utilisateur continue à taper des lettres alors que l'automate se trouve dans l'état 5 : un a envoie vers l'état 2 (la suite a a été tapée, et la suite *bba* est attendue), et un b envoie vers l'état 3 (la suite *ab* a été tapée, et la suite *ba* est attendue).
Finalement, voici l'automate qui modélise le comportement de ce premier digicode :

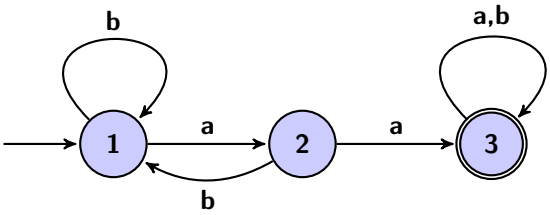


Notons que les états et les transitions de l'automate n'ont pas forcément une contrepartie matérielle dans l'appareil, comme des pièces mécaniques ou des composants électroniques. En effet, un automate modélise le comportement d'un système, mais pas nécessairement son architecture interne.

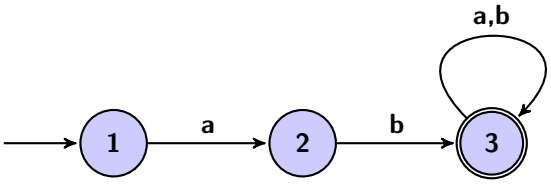
Un autre digicode On s'intéresse maintenant à un digicode de même type, qui débloque le bouton d'ouverture de la porte pour toutes les suites de lettres a et b qui contiennent (au moins) deux a consécutifs, et seulement pour celles-ci. Ainsi, les suites $abaab$, ou $aaaaa$ ou $bbabaabaab$ vont convenir, mais pas les suites $ababab$ ou $bbba$. Tout d'abord, la suite aa doit être reconnue. On commence donc avec le schéma ci-dessous :



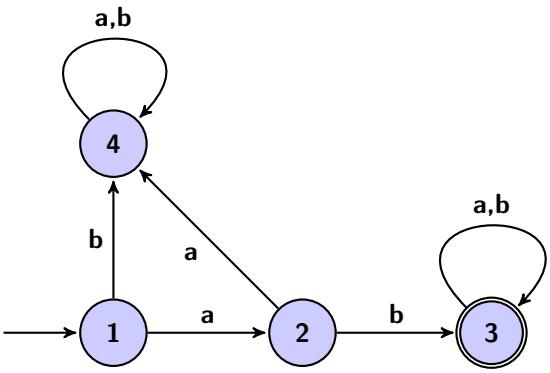
Puis on complète l'automate avec les autres transitions. Notons que, dès que la suite aa a été tapée une fois, l'utilisateur doit pouvoir continuer à taper d'autres lettres sans que le bouton d'ouverture de la porte ne se bloque de nouveau : c'est le sens de la boucle étiquetée a, b sur l'état terminal de l'automate.



Un dernier digicode Pour terminer cette série d'exemples introductifs, on modélise le comportement d'un digicode à deux touches qui débloque le bouton d'ouverture de la porte pour toutes les suites de lettres a et b qui commencent par ab, et seulement pour celles-ci. Ainsi, les suites $abaab$, ou abb vont convenir, mais pas les suites baa ou $aaba$.
Pour commencer, la suite ab doit évidemment être reconnue. De plus, dès que la suite ab a été tapée, l'utilisateur doit pouvoir continuer à taper d'autres lettres sans que le bouton d'ouverture de la porte ne se bloque de nouveau : comme dans le cas précédent, on modélise ce phénomène par une boucle étiquetée par a, b sur l'état terminal de l'automate. On obtient à ce stade le schéma ci-dessous :



On observe que deux événements n'apparaissent pas sur ce schéma : frapper un b lorsque le système se trouve dans l'état 1 et frapper un a dans l'état 2. Ces deux événements mènent à une impasse, puisqu'aucune suite commençant par b ou par aa ne permet de débloquer le bouton d'ouverture de la porte. Il y a deux façons d'en tenir compte sur l'automate. La première consiste simplement à ne pas dessiner les transitions interdites : on se contente alors du schéma ci-dessus. Dans ce cas, par exemple, la suite de lettres aab n'est pas reconnue, puisqu'elle « se bloque » après la première lettre, et ne permet donc pas d'atteindre l'état terminal. La deuxième méthode pour tenir compte de ces impasses consiste à introduire un **état rebut**, dont on ne peut pas s'échapper, et vers lequel on envoie les transitions interdites. On obtient alors l'automate ci-dessous :

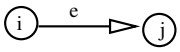


2 Qu'est-ce-qu'un automate ?

Dans cette partie, on précise les notions introduites dans les exemples précédents.

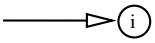
- **État** 

Chaque état du sytème est symbolisé par un état de l'automate et représenté par un cercle. On attribue souvent un nom ou un numéro aux états pour pouvoir les repérer.

- **Transition** 

Chaque transition entre deux états est représentée par une flèche étiquetée : quand le système est dans l'état i, la transition e le fait passer dans l'état j.

- **Étiquette** e
Les étiquettes des transitions symbolisent les événements subis par le système et sous l'effet desquels il évolue.

- **État initial** 

Un état particulier, repéré par une flèche sans origine, modélise l'état dans lequel se trouve le système au début de son évolution.

- **État(s) terminal(aux)** 

Un ou des état(s) particulier(s) représente(nt) l'objectif atendu de l'évolution du système. On utilise un double cercle pour signaler un état terminal.

Notons que l'adjectif « fini » accolé au nom « automate » souligne simplement le fait que les états et les étiquettes pour un automate donné doivent obligatoirement être en nombre fini.

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

- On commence dans l'état initial.
- À partir de cet état initial, toute suite finie d'événements aboutit à un certain état.
- On distingue deux sortes de suites d'événements : celles qui aboutissent à un état terminal et les autres.
- Par exemple, pour le premier digicode, on a fait en sorte que les suites de lettres qui aboutissent à l'état 5 (terminal) soient celles qui se terminent par *abba* et seulement elles. Par conséquent, on peut considérer que cet automate décrit une méthode permettant de *reconnaître* à coup sûr, parmi tous les suites de lettres (constituées de a et de b) possibles, celles qui se terminent par *abba*.
- C'est le concept sous-jacent à la notion d'automate fini : il s'agit d'une description visuelle d'un procédé permettant de reconnaître à coup sûr un ensemble de suites d'événements possédant une propriété commune parmi toutes les suites possibles (composées des mêmes événements).

Dans notre exemple, il y a exactement deux événements possibles : l'utilisateur peut taper la lettre a ou la lettre b. Naturellement, les « événements » dépendent du système modélisé : 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, détecter qu'un joueur avance un pion, etc.

3 À quoi servent les automates ?

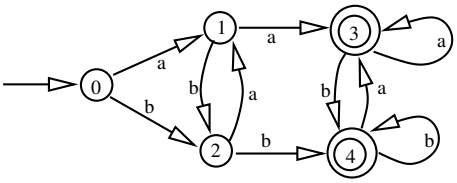
On a dit que le modèle des automates finis est très souple et s'adapte à des domaines très différents. Avant de donner de nouveaux exemples, nous allons évoquer l'objectif de ces modélisations. On aborde dans cette partie des sujets plus difficiles, aussi on se contente d'évoquer des pistes très générales sans entrer dans des explications techniques.

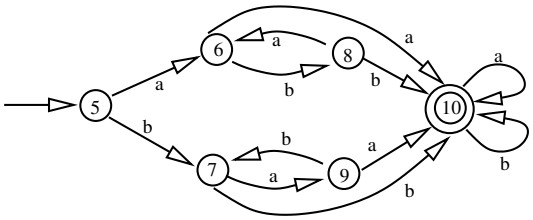
Le premier point à prendre en compte est le fait que les automates correspondant à des systèmes réels sont souvent beaucoup plus gros et compliqués que les exemples présentés ici. Il en résulte qu'on ne peut en règle générale pas les visualiser globalement, ni les analyser ou les traiter à la main. On utilise à la place des logiciels pour effectuer les différentes opérations décrites ci-dessous.

Aide à la fabrication À partir de la description d'un automate, on peut en fabriquer un équivalent électronique, ou écrire un programme reconnaissant les mêmes suites d'événements. Il existe des logiciels qui le font automatiquement à partir de la liste des états et des transitions avec leurs étiquettes. On peut ainsi fabriquer un véritable digicode ou, comme on le verra plus loin, écrire un logiciel qui compte les points au tennis et fait retentir un signal sonore quand un des joueurs a gagné un jeu.

Par conséquent, la modélisation par un automate peut être la première étape de la conception d'une machine ou d'un programme. Cette étape sera ensuite suivie d'une étape de réalisation concrète. Nous avons réalisé une telle étape de modélisation dans la première partie à partir de la description fonctionnelle d'un digicode qui « débloque le bouton d'ouverture d'une porte lorsqu'on frappe n'importe quelle suite de lettres qui se termine par *abba*, et seulement l'une de celles-ci ».

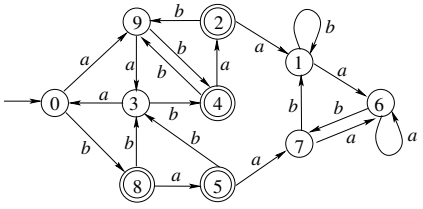
Équivalence Supposons qu'on ait obtenu par deux méthodes différentes les deux automates ci-dessous pour la modélisation d'un même système.

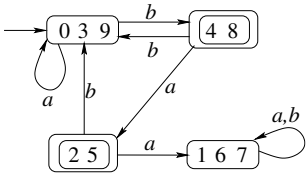




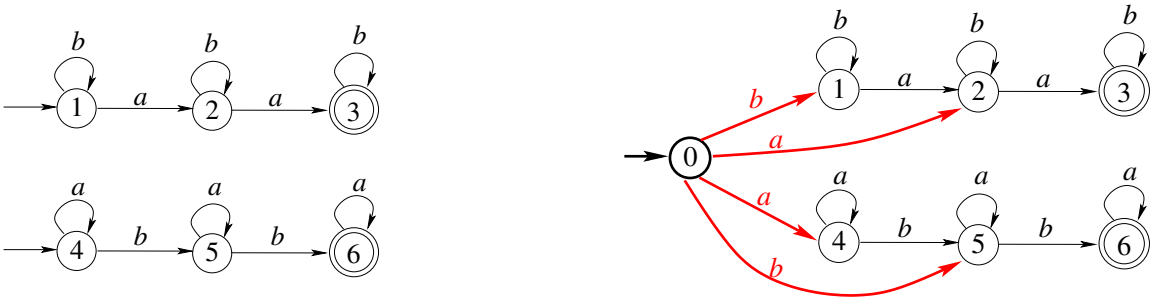
On a évidemment besoin de savoir si on a bien affaire à des automates équivalents, c'est-à-dire qui reconnaissent exactement les mêmes mots. Ce n'est en général pas visible, même sur des petits exemples comme ici, et on imagine facilement la difficulté de cette question pour des automates de quelques centaines d'états. Il est donc très utile de savoir 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 permet de trouver l'automate le plus simple possible équivalent à un automate donné.





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 serait intéressant de réutiliser l'automate qu'on construit dans les exemples qui suivent 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.

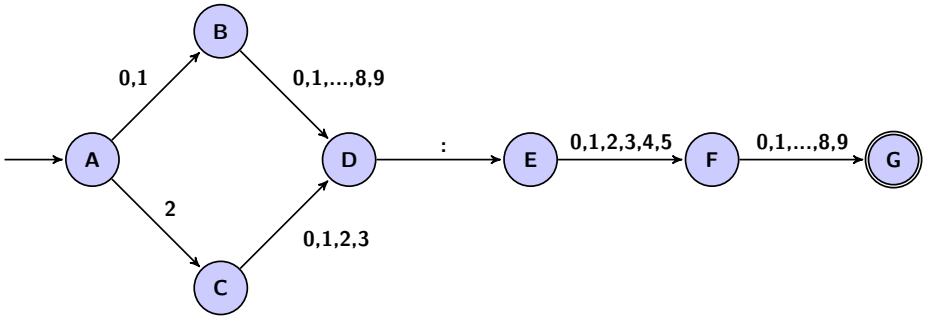
Vérifier des propriétés Il existe enfin des méthodes permettant de tester diverses propriétés du système à partir de son modèle automate. Par exemple, si on a écrit le scénario d'un jeu, il est souhaitable de s'assurer qu'il n'a pas un défaut qui fait que certaines parties ne se terminent jamais, ou qui fait 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 comportement 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.
Sans entrer dans les détails, signalons par exemple qu'un automate, dans la mesure où il ne possède qu'un nombre fini d'états, n'est pas capable de compter au-delà d'une certaine valeur fixée. Ainsi, on peut montrer qu'il n'existe aucun automate fini qui reconnaisse les mots qui contiennent autant de a que de b (quelle que soit leur longueur), et uniquement ceux-ci. Ainsi, les mots *ababbbaa* ou *aaaaaabbbbbb* seraient reconnus, mais pas *aab*, ni *bbbbabab*. 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 total de sorties à la fin de la journées est bien égal au nombre total 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.

4 D'autres automates

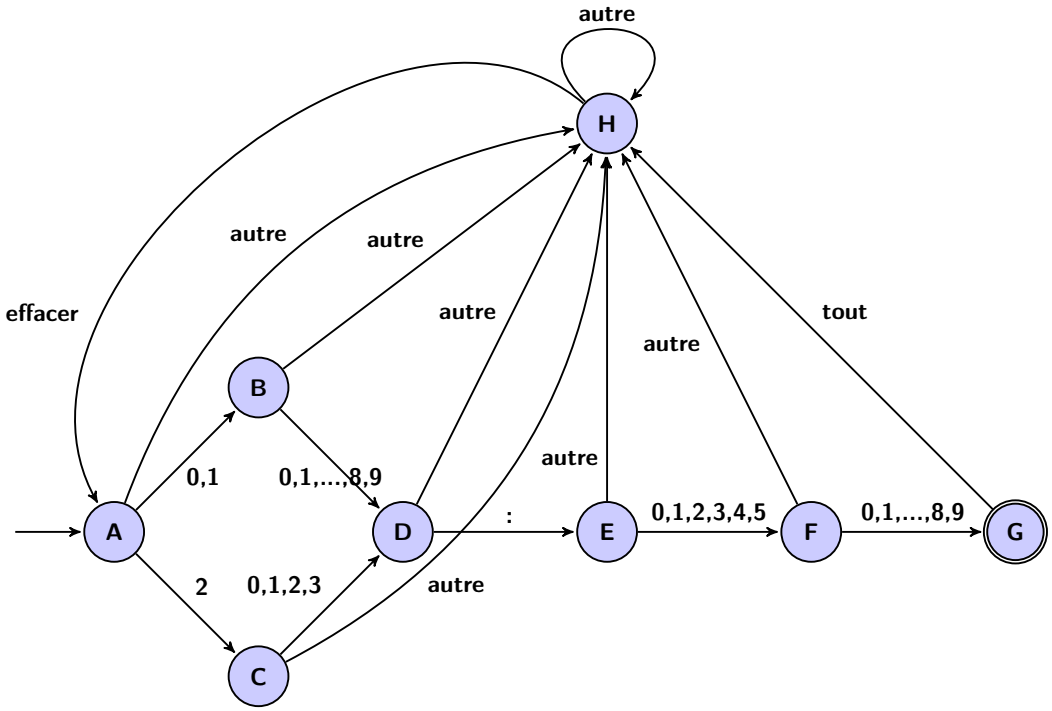
Vérificateur de format d'horaire Un des domaines où la modélisation à l'aide d'automates finis est très utilisée est ce qu'on appelle l'algorithmique du texte au sens large : recherche documentaire, traitement de texte, génomique, compilation, etc. On parle alors de recherche de motifs ou d'analyse syntaxique selon les domaines d'application. On détaille ci-dessous un exemple simple utilisé dans les activités proposées pour les élèves. Dans ce paragraphe, les noms des états sont des lettres, pour ne pas créer de confusion avec les étiquettes dont certaines sont des chiffres.

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 comportement du programme de vérification de la valeur donnée par l'internaute peut être modélisé par l'automate ci-dessous.

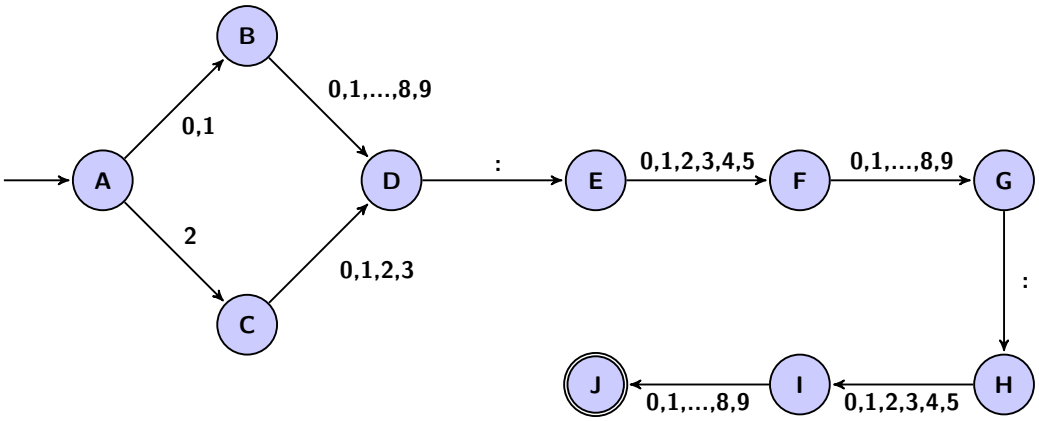


Détaillons le schéma : il faut impérativement deux chiffres, suivis du symbole `:`, puis encore deux chiffres. Le premier chiffre (celui des dizaines des heures) ne peut prendre que les valeurs 0, 1 ou 2. Si c'est un 0 ou un 1, alors le chiffre des unités correspondant peut prendre n'importe quelle valeur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

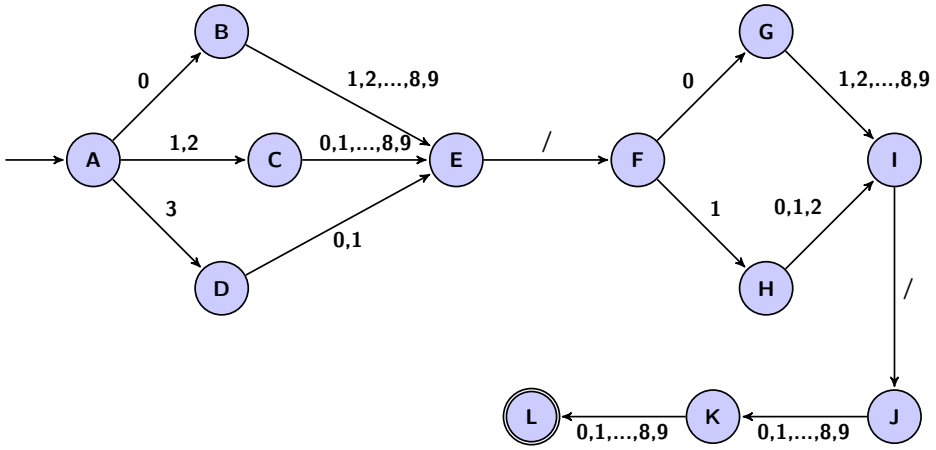
Pour aboutir à un modèle plus réaliste, il nous reste à tenir compte du fait que le clavier de l'ordinateur possède d'autres touches que les chiffres et le symbole `:`, d'une part, et que le formulaire propose probablement une possibilité d'effacer en cas de fausse manœuvre, d'autre part. On introduit un état supplémentaire symbolisant 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).
Pour ne pas trop compliquer le schéma, on a simplement étiqueté les transitions avec `tout` pour indiquer que toutes les touches ont le même effet et avec `autre` pour indiquer que toutes les touches autres que celles indiquées ont le même effet.
On aboutit alors à l'automate suivant :



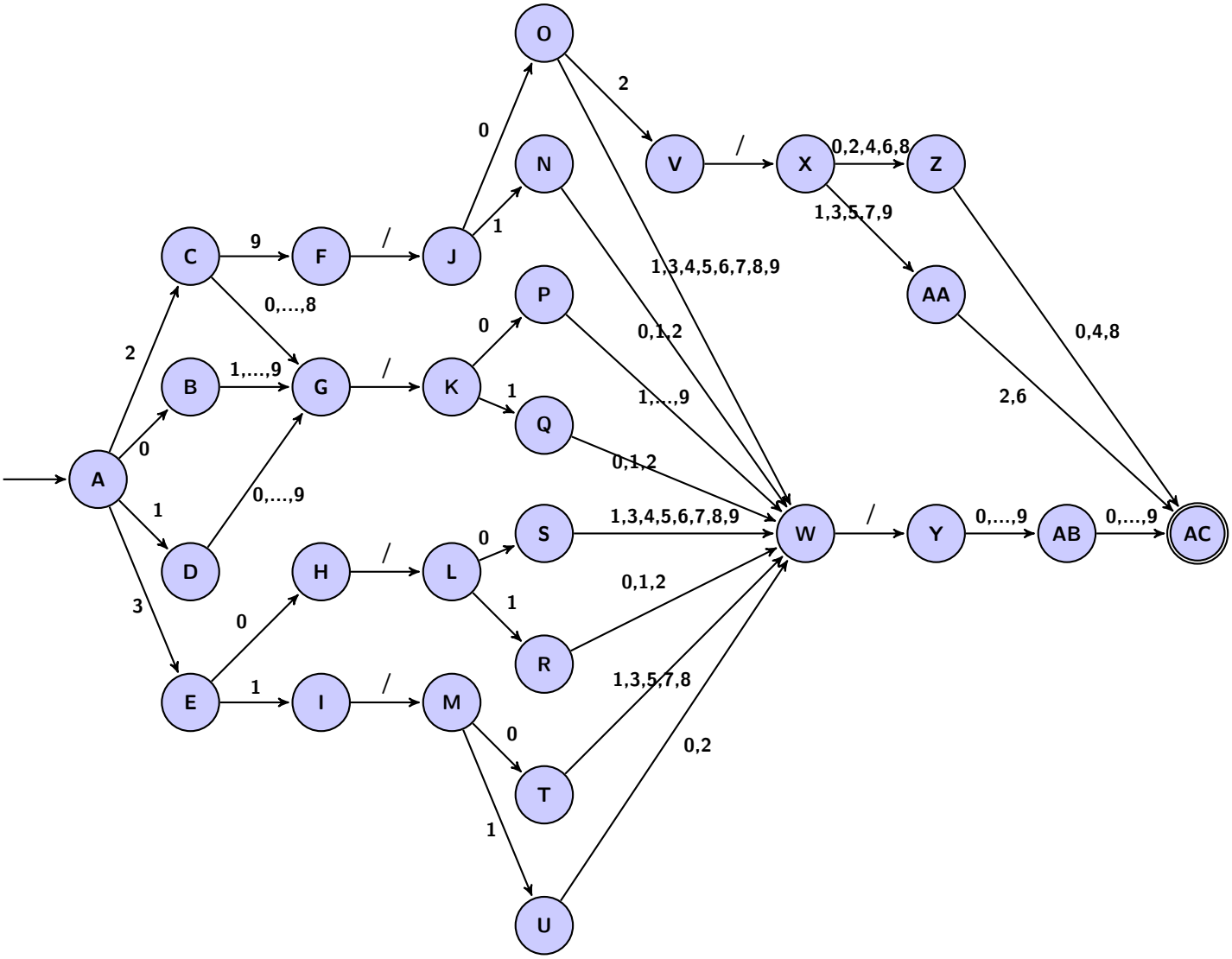
La variante ci-dessous tient compte des secondes : on attend un format `HH:MM:SS`, avec des valeurs allant de 00 : 00 : 00 à 23 : 59 : 59.



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.

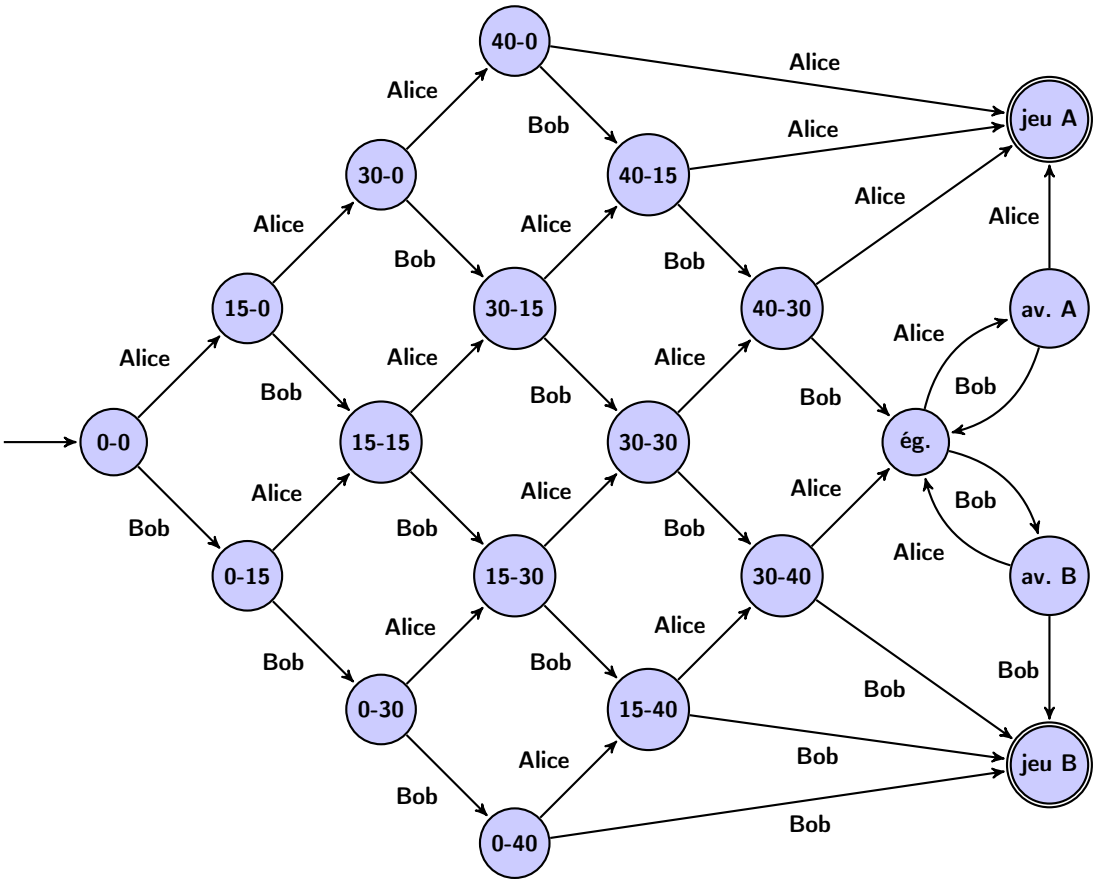


Cependant, le calendrier réel est bien plus compliqué. Il faut tenir compte des mois qui ont 30 jours, du mois de février qui n'a que 28 jours, et 29 jours les années bissextiles. L'année 2000 étant bissextile, contrairement aux autres années multiples de 100 (1900 et 2100 par exemple), on va se limiter 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 F et J, G et K, H et L et I et M) et entre les mois et les années d'autre part (entre les états V et X et W et Y).
La partie la plus simple de l'automate est celle qui va de l'état A à l'état G, puis de l'état K à l'état W et enfin de l'état Y à l'état AC. 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 A, et atteint l'état W en passant par l'état H, puis rejoint l'état AC.
Par contre, le numéro de jour 31 ne se produit que certains mois (passage par l'état I), et ceci pour n'importe quel numéro d'année (passage par l'état W).
Il reste à traiter le numéro de jour 29 (passage par l'état F) : pour n'importe quel mois sauf celui de numéro 02, il se produit pour n'importe quelle année (état W). Pour le mois 02, on passe par l'état X, et il reste à restreindre les numéros d'années permettant d'atteindre l'état terminal AC à ceux des années bissextiles (c'est-à-dire ceux qui sont des multiples de 4).

Le tennis Le dernier type d'exemples de modélisation d'un système à l'aide d'un automate fini 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 entre *Alice* et *Bob*. Les états correspondent aux différents scores possibles (comme `40-15`, ou `avantage Alice`), et les évènements `Alice` ou `Bob` représentent un point marqué par le joueur *Alice* ou *Bob* 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 modélisables par des automates finis.

Dernier exemple : une machine à café De nombreuses machines mécaniques ou électroniques peuvent être représentées par des automates finis. 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 ferme la fente permettant d'introduire des pièces, débloque un bouton permettant d'obtenir le café, rend éventuellement la monnaie et retourne dans l'état initial si on appuie sur le bouton `café`. 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.