Automates

Benjamin Monmege

2021/2022

Jusqu'à maintenant, nous avons surtout étudié des algorithmes et des structures de données permettant de résoudre des problèmes concrets. Mais, dans l'introduction, nous avons évoqué le problème de savoir si un problème possède une solution : nous avons vu le problème géométrique de la quadrature du cercle qui s'est avéré ne pas avoir de solution. Existe-t-il de telles impossibilités en informatique également, c'est-à-dire des problèmes tels qu'aucun calcul, aucun algorithme n'est capable de le résoudre ? Pour pouvoir répondre à cette question, il nous est nécessaire de préciser ce que nous entendons par *impossible*, et donc ce qui est *possible* également, à savoir *calculable*. Ce chapitre évoque dans un premier temps une notion assez faible de calculabilité, les arbres de décision, que nous enrichirons ensuite avec les automates, puis finalement avec les machines de Turing.

1 Arbres de décision

Commençons donc par utiliser des arbres pour prendre des décisions. C'est très naturel et c'est ce que nous faisons sans même le savoir tous les jours. Par exemple, imaginons un médecin qui doit diagnostiquer un patient entrant dans son cabinet. Comment procède-t-il ? Une façon de faire est de commencer par poser une première question, par exemple « Avez-vous des douleurs ? » : suivant la réponse, le médecin apprend de l'information et il peut continuer à poser des questions, jusqu'à avoir appris suffisamment d'informations pour prendre une décision, c'est-à-dire déclarer si le patient est malade et si oui, quelle maladie il a probablement. Les différents états de connaissance du médecin au cours du diagnostic sont reliés par des

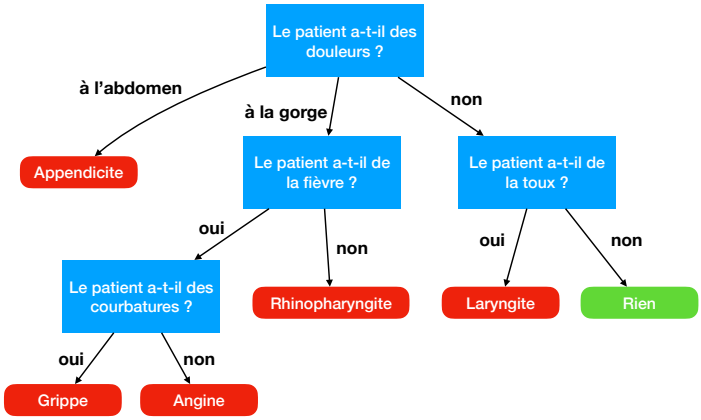


FIGURE 1 – Un arbre de décision pour décider la maladie du patient

FIGURE 2 – Le jeu du *Qui est-ce ?*

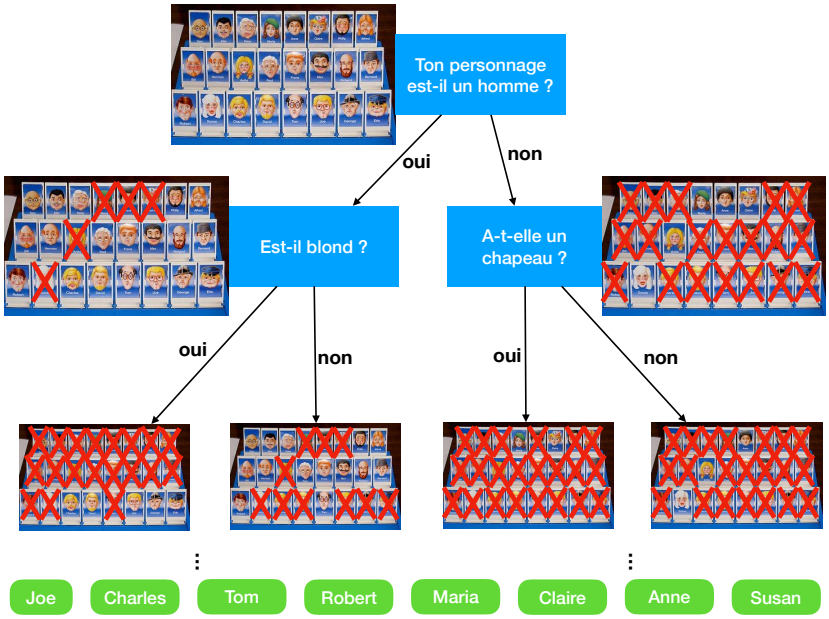


FIGURE 3 – Un extrait de l'arbre de décision pour représenter une stratégie au *Qui est-ce ?*

arcs selon les réponses aux questions qu'il pose : cela forme un arbre (en effet, un cycle dans ce diagramme de diagnostic n'aurait pas beaucoup de sens puisque cela voudrait dire que le médecin pose deux fois la même question), qu'on appelle *arbre de décision*. Un exemple très simplifié de tel arbre de décision est représenté en Figure 1.

Lorsque nous jouons au jeu du *Qui est-ce ?* (cf Figure 2), nous procédons par élimination successive de personnages en posant à l'autre joueur des questions auxquelles il répond par oui ou non. On peut à nouveau représenter une stratégie de l'un des joueurs à l'aide d'un arbre de décision : chaque nœud de l'arbre est à nouveau une question qu'il pose à son adversaire et selon la réponse apportée (oui ou non), on continue dans le sous-arbre gauche ou le sous-arbre droit. Le début d'un arbre de décision possible pour ce jeu est représenté en Figure 3 : les feuilles de cet arbre sont les situations où un seul visage reste possible auquel cas le joueur a deviné (sauf erreur ou tricherie de l'adversaire...) le personnage choisi par l'adversaire.

Un dernier exemple d'arbre de décision que nous utilisons tous les jours sans le savoir

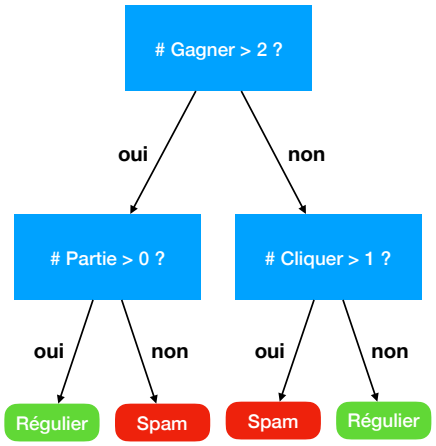


FIGURE 4 – Un arbre de décision simplifié pour détecter les spams

consiste en la détection de spams dans les applications d'emails. Par exemple, considérons les deux emails suivants :

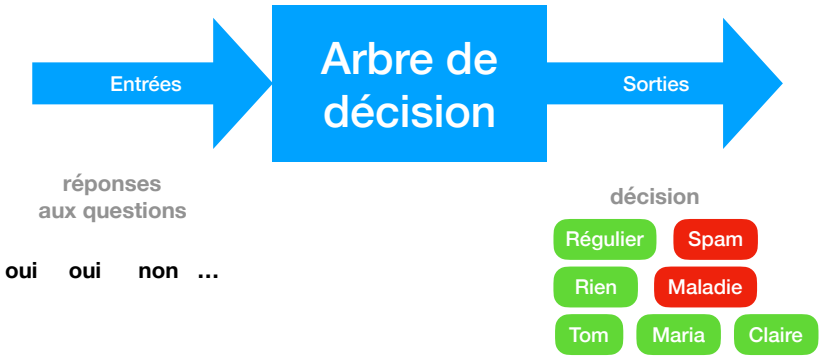
> Bravo,
> Vous venez de gagner à notre grand tirage internet. Cliquez ici pour recevoir 1.000.000 de dollars !!! Et pour gagner d'autres cadeaux, cliquez ici.

> Salut Benoît,
> Demain je joue contre Bruno. Si je gagne la partie, je me qualifie directement. Si je ne gagne pas demain mais que je gagne la suivante, je monte en pool 3 l'année prochaine !
> Bises, Sandra

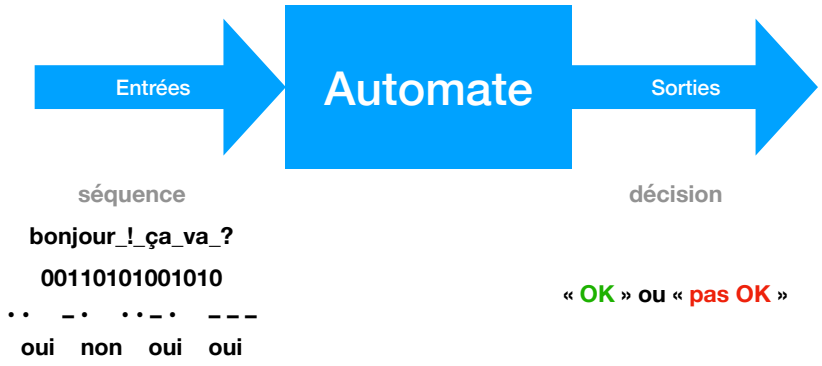
Comment détecter si ce sont des spams ou des messages réguliers qu'il ne faut pas filtrer ? La méthode la plus simple, très efficace, consiste à utiliser des arbres de décision. Les questions sont des critères qu'on peut vérifier sur chaque email, par exemple la comparaison du nombre d'occurrences d'un mot (ou sa fréquence d'apparition dans l'email) avec une constante. Un tel arbre de décision simplifié est donné en Figure 4.

Pour le message de gauche, le nombre de fois que le mot « gagner » apparaît est inférieur ou égal à 2, mais le mot « cliquer » (sous une forme conjuguée) apparaît strictement plus d'une fois : l'arbre de décision conclut donc que c'est un spam (la troisième feuille en partant de la gauche). Pour le second message en revanche, le mot « gagner » apparaît strictement plus de deux fois, mais le mot « partie » apparaît aussi, donc l'arbre de décision permet de détecter qu'il s'agit probablement d'un email régulier. En pratique, les arbres de décision utilisées pour détecter les spams sont bien plus grands et ils sont mis au point à l'aide de méthodes d'apprentissage automatique, à partir d'une grande quantité d'emails qui sont déjà classifiés comme étant réguliers ou spams : on parle d'*apprentissage supervisé*.

Essayons désormais de préciser ce qu'on entend par « calculer avec un arbre de décision ». Dans les exemples que nous avons vus jusqu'alors, un arbre de décision est un processus de calcul qui prend en entrée des réponses (oui ou non) aux questions qu'il pose, et les utilise afin de produire une sortie, la décision attendue (régulier/spam, malade/pas malade, le nom du personnage du *Qui est-ce ?*) :

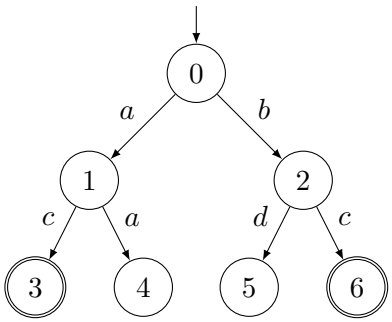


Si on s'abstrait un peu des exemples, on peut considérer qu'on prend en entrée une séquence de réponses prises dans un ensemble fini de réponses possibles, et qu'on prend une décision « OK » ou « pas OK », suite à la séquence de réponses observée. Cette étape de généralisation de ce que peut prendre un arbre de décision en entrée nous invite à changer de nom : désormais, nous appellerons un tel processus prenant des séquences en entrée et renvoyant « OK »/« pas OK » en sortie un *automate* :

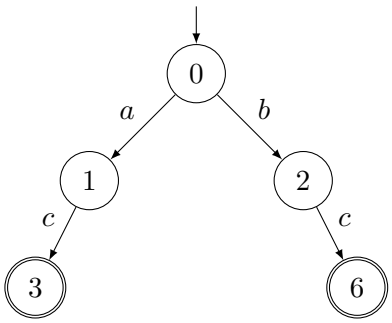


En entrée, on peut donc imaginer que l'automate prend des caractères pour lire des séquences de caractères (du texte, donc), ou bien des séquences de 0 et de 1 (pour représenter des entiers en binaire par exemple), ou bien encore des points et des traits, éléments de base du code Morse. Les arbres de décision sont donc des automates particuliers, qui lisent des séquences de « oui »/« non ». Dans le cas général, les sommets de l'arbre de décision portent le nom d'*états* dans les automates : l'état d'où l'on part au début du processus (la racine de l'arbre de décision) est appelé *état initial*. Les éléments qu'on lit (des caractères, des 0/1, des oui/non...) sont appelés des *lettres* et les arcs d'un état à l'autre, étiquetés par une lettre, sont appelés des *transitions*. Certains états sont les états où l'automate prend une décision : certains sont *acceptants* (ceux où on déclare la séquence de lettres lues comme étant « OK ») et les autres sont non acceptants.

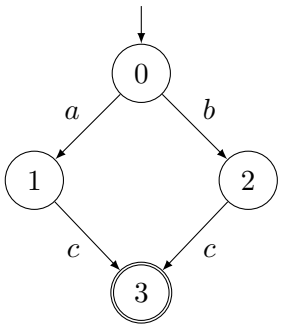
Voici un premier exemple d'automate sur l'alphabet $\{a, b, c, d\}$:



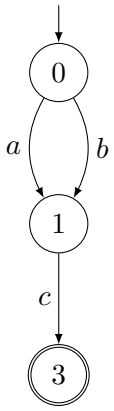
Cet automate possède sept états numérotés de 0 à 6. L'état 0 est l'état initial. Seuls les états 3 et 6 sont acceptants (parmi les feuilles $\{3, 4, 5, 6\}$) : on les représente par un double cercle. Un automate décrit un ensemble de séquences « valides », c'est-à-dire celles pour laquelle il décide « OK ». Pour l'exemple ci-dessus, seules deux séquences de lettres permettent d'aller de l'état initial à un état acceptant : la séquence ac et la séquence bc. Les séquences aa et bd parviennent à des feuilles mais ne sont tout de même pas acceptées par l'automate. On dit que le *langage reconnu par l'automate* est $\{ac, bc\}$. Notez que la séquence a n'est pas reconnue par l'automate, mais on peut l'étendre de façon à atteindre un état acceptant. En revanche, non seulement aa n'est pas reconnue, mais aucune suite possible ne mène à un état acceptant. C'est la même chose que pour la séquence c ou d : elles ne sont pas acceptées et aucune suite possible ne permet d'atteindre un état acceptant. C'est la raison pour laquelle on ne les a même pas représentées sur le dessin. De même, on peut donc ignorer les transitions menant de 1 à 4 et de 2 à 5. L'automate ci-dessous est donc équivalent au précédent, au sens où il accepte le même langage :



Maintenant qu'on a supprimé des états inutiles, il apparait qu'on peut encore simplifier cet automate. En effet, on peut essayer de *partager* des états, quitte à ne plus insister sur le fait que l'arbre de décision soit un *arbre*. Par exemple, il ne rime à rien de distinguer les états 3 et 6 qui sont tous deux acceptants mais ne permettent pas de continuer à lire des lettres. On peut donc les fusionner sans crainte :



Mais alors, on s'aperçoit qu'il est aussi inutile de distinguer les états 1 et 2 qui sont tous deux non acceptants, mais permettent de lire uniquement la lettre c en se rendant dans le même état 3 : ils réagissent de la même manière dans le futur, donc on peut sans crainte les fusionner également. On obtient donc l'automate suivant :



Cela n'a plus rien d'un arbre, mais on appelle toujours cela un automate. Voyons donc une définition formelle de ces automates finis (car on se concentre sur les automates qui possèdent un ensemble fini d'états).

2 Automates finis

Un automate permet de décrire des ensembles de séquences sur un alphabet A donné :

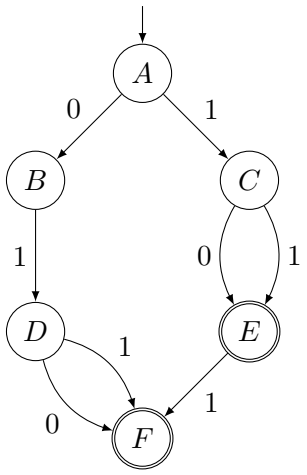
— si l'alphabet est $A = \{0, 1\}$, l'automate décrit un ensemble de séquences de 0 et 1 : un automate peut ainsi accepter des codes binaires d'entiers ;

— si l'alphabet est $A = \{a, b, c, \ldots, y, z\}$, l'automate décrit un ensemble de mots sur l'alphabet latin : un automate peut ainsi accepter l'ensemble des mots du dictionnaire français ;

— si l'alphabet est $A = \{oui, non\}$, l'automate accepte des séquences de décisions à certaines questions : un automate permet alors de représenter une stratégie au jeu du *Qui est-ce ?* dans un arbre de décision.

Un automate est composé d'*états* et de *transitions* qui sont des arcs menant d'un état (potentiellement acceptant, contrairement aux exemples vus jusqu'à maintenant) à un autre (potentiellement le même), étiquetés par des lettres de l'alphabet A. Un automate possède également un état *initial* et des états *acceptants*.

On représente ci-dessous un automate $\mathcal{A}$, sur l'alphabet $\{0, 1\}$:



Ses états sont A, B, C, D, E et F. L'état initial est A, distingué par une flèche entrante. Les états acceptants sont E et F distingués par un double cercle. Cet automate possède 8 transitions : par exemple, il y en a une de l'état A vers l'état B étiquetée par la lettre 0, et une autre de l'état E à l'état F étiquetée par la lettre 1.

Notons que tous les automates vus jusqu'alors vérifient la propriété suivante :

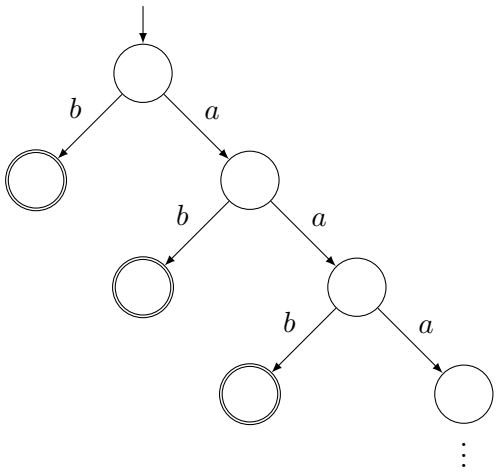
> « Pour tout état e de l'automate et pour toute lettre ℓ de l'alphabet, il existe au plus une transition sortant de l'état e étiquetée par la lettre ℓ. »

On dit que l'automate est *déterministe* en cela qu'à tout moment il n'y a aucun choix pour continuer la lecture d'une séquence de lettres fixée à l'avance : soit on bloque car il n'y a pas de transition étiquetée par la lettre voulue (par exemple, si on veut lire deux lettres 0 dans l'automate du dessus, on bloque en B qui ne peut pas lire une lettre 0 en suivant une transition), soit on suit l'unique transition étiquetée par la lettre voulue. Dans ce cours, on considèrera uniquement des automates déterministes.

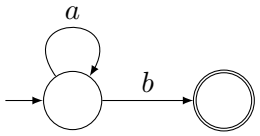
Un automate accepte un *langage*, qui est un ensemble de séquences. Une séquence s est *acceptée* par l'automate s'il existe un chemin (et s'il existe, il est unique) de l'état initial à un état acceptant dont la séquence des étiquettes des transitions visitées (dans l'ordre) est s. Ainsi, la séquence 011 est acceptée par l'automate $\mathcal{A}$ ci-dessus, mais pas la séquence 01 qui ne termine pas dans un état acceptant, ni la séquence 110 pour laquelle il n'y a pas de chemin avec cette étiquette (on bloque en E lorsqu'il s'agit de lire la lettre 0). L'ensemble des séquences acceptées par l'automate $\mathcal{A}$ est $\{010, 011, 10, 11, 101, 111\}$: ce sont l'ensemble des codages binaires sur moins de 3 bits représentant des entiers premiers (2, 3, 5, 7).

Jusque-là, nous n'avons vu que des automates qui acceptent un langage fini de séquences. Comment peut-on accepter un langage infini ?[1] Prenons un exemple : peut-on trouver un automate acceptant toutes les séquences sur l'alphabet $\{a, b\}$ qui ne contiennent que des a sauf la dernière lettre qui doit être un b, c'est-à-dire le langage $\{b, ab, aab, aaab, aaaab, \ldots\}$. Clairement, ce langage contient un ensemble infini de séquences. Notre première tentative pourrait être de dessiner un arbre de décision : depuis l'état initial, soit on lit un b et on accepte directement, soit on lit un a et on va dans un nouvel état qui, à nouveau soit lit un b et accepte directement, soit lit un a et poursuit une étape de plus, etc.

1. Attention, c'est bien le langage dont on veut qu'il contienne un ensemble infini de séquences, pas les séquences elles-même qui sont toujours finies dans ce cours.

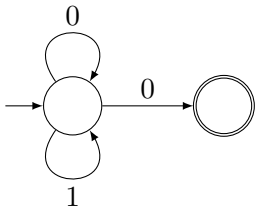


Malheureusement, cet automate n'est pas fini : il possède un nombre infini d'états. Plus précisément, il a un état acceptant par séquence à accepter... Comme précédemment cependant, on se rend compte que tous les états acceptants peuvent être aisément fusionnés puisqu'ils sont tous acceptants et ne permettent de lire aucune lettre supplémentaire. Une fois qu'on a fait cela, on peut se rendre compte que tous les états non acceptants sont alors « similaires » : il permettent tous de partir vers l'unique état acceptant en lisant un b, ou continuer vers eux en lisant un a. On peut donc tous les fusionner et on obtient finalement l'automate fini (car il possède un nombre fini d'états) suivant :

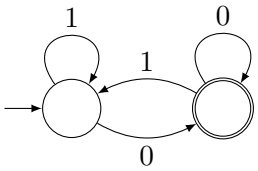


Dans les dessins, il n'est pas toujours nécessaire de donner des noms aux états, comme dans l'exemple de l'automate $\mathcal{B}$. Mais on peut évidemment donner des noms si on le souhaite, pour aider à comprendre la signification de l'état : ici, l'état de gauche de $\mathcal{B}$ pourrait s'appeler « que des a » et l'état de droite « b à la fin », ou simplement « A » et « B » si on veut des noms plus courts...

Revenons sur l'alphabet $\{0, 1\}$ pour un nouvel exemple de langage infini. Les séquences sur cet alphabet représentent des entiers codés en binaire. Peut-on trouver un automate acceptant l'ensemble des codes des entiers pairs ? Rappelons-nous qu'un entier est pair si et seulement si son codage en binaire termine par un 0. On souhaite donc reconnaître l'ensemble des séquences de 0 et de 1 qui terminent par un 0 : cela ressemble au langage précédent (à renommage près du a en 1 et du b en 0), mais cette fois-ci, on doit pouvoir accepter n'importe quelle séquence avant de lire le *dernier* 0. On pourrait donc se dire qu'un automate permettant d'accepter ce langage est :



On lit n'importe quelle séquence de 0 et de 1 en restant dans l'état de gauche, avant de passer dans l'état de droite en lisant un 0. Malheureusement, cet automate n'est pas *déterministe* : dans l'état de gauche, en lisant un 0, l'automate a le choix entre rester dans l'état de gauche, ou passer dans l'état de droite. On ne s'autorise pas de tels automates à choix dans ce cours. Il nous faut donc modifier notre premier essai. En fait, notons qu'une fois qu'on vient de lire un 0, on doit nécessairement se trouver dans un état acceptant puisqu'il se pourrait que ce 0 soit le dernier bit du codage binaire. Par contre, lorsqu'on vient de lire un 1, on doit se trouver dans un état non acceptant. Cela nous amène directement à cette seconde tentative :



Cet automate est déterministe et il accepte bien toutes les séquences de bits qui terminent par un 0, c'est-à-dire les codages binaires des entiers pairs.

3 Applications des automates finis

Les automates finis sont très utiles pour modéliser des situations de la vie courante. Ils sont d'ailleurs intensivement utilisés à ces fins dans l'industrie. Prenons un exemple : modéliser le comportement normal d'un distributeur de café. Voici les spécifications données par l'industriel qui les construit :

— le café est à 30 centimes d'euro ;
— le thé est à 50 centimes d'euro ;
— la machine n'accepte que les pièces de 10 et 20 centimes d'euro ;
— on peut insérer 50 centimes d'euro au maximum dans la machine ;
— on peut annuler à tout moment (et récupérer la monnaie).

Décrivons alors l'ensemble des fonctionnements normaux du distributeur à l'aide d'un automate. Il admet comme alphabet les cinq lettres suivantes : « insertion d'une pièce de 10 centimes », « insertion d'une pièce de 20 centimes », « appui sur le bouton *Café* et distribution du café », « appui sur le bouton *Thé* et distribution du thé », « appui sur le bouton *Cancel* et récupération de la monnaie ». On ne considère donc pas comme un fonctionnement normal l'appui sur le bouton *Café* ou *Thé* sans distribution de café ou de thé : dis autrement, cela veut dire que le distributeur ne doit rien faire dans le cas où l'utilisateur appuierait sur l'un de ces deux boutons dans un autre cas que ceux considérés dans le fonctionnement normal. Afin de pouvoir savoir quand le distributeur doit délivrer du café ou du thé, il nous faut retenir le montant déjà introduit dans la machine : ce sont le rôle des états dans un automate, qui sont l'unique mémoire disponible. On créé donc 6 états : 0 centimes (début), 10 centimes, 20 centimes, 30 centimes, 40 centimes et 50 centimes. Il ne reste plus qu'à ajouter les transitions. Une représentation de l'automate est donné en Figure 5. On y a représenté l'état début en vert pour signifier qu'il pourrait être considéré comme état acceptant de l'automate, c'est-à-dire un état où l'on peut sans souci éteindre le distributeur sans léser un consommateur qui a déjà inséré de l'argent dedans.

Une autre application possible des automates concerne la recherche de motif dans un texte. Par exemple, dans un navigateur web ou dans un traitement de texte, vous avez toujours la

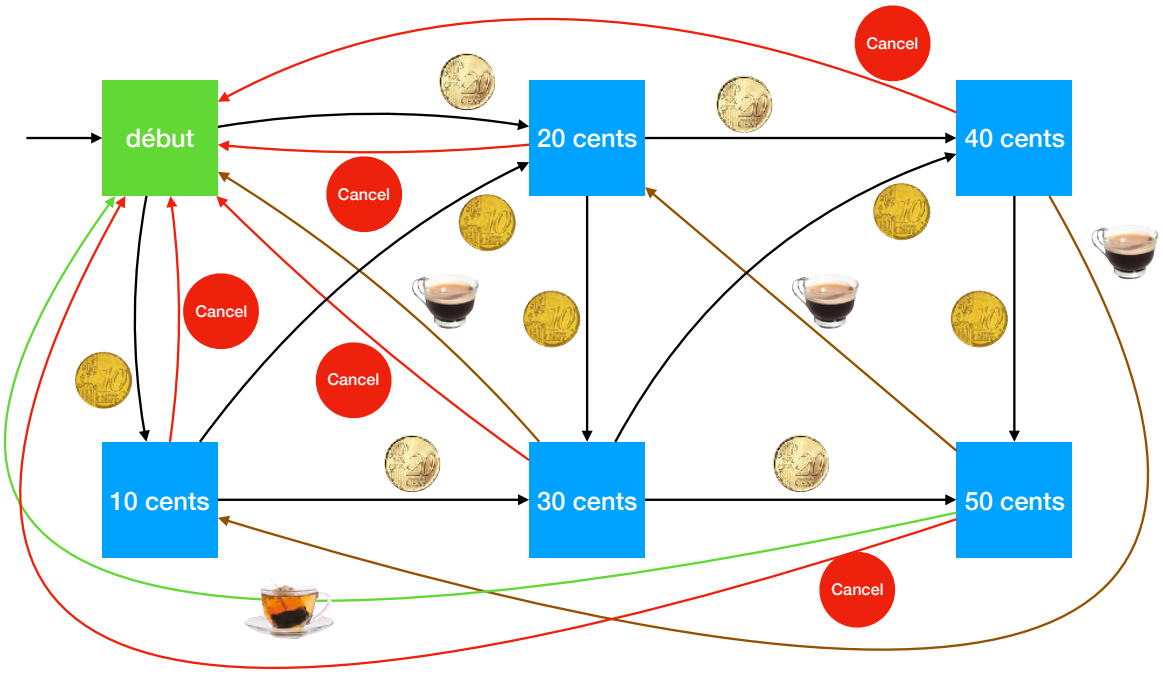


FIGURE 5 – Un automate pour le distributeur de café

possibilité de rechercher un mot dans le document. Cette tâche est exécutée très efficacement par un automate (à tel point que c'est souvent à l'aide d'automates que les algorithmes de recherche sont effectivement décrits en pratique). Trouver un mot dans un texte, cela revient à décrire un automate qui accepte la partie initiale du texte qui termine par le mot recherché. Typiquement, considérons le texte :

> Du journal « Le petit bachelier » : comme les professeurs
> se plaisent à le rabâcher, le baccalaureat est important... !

Si on recherche le mot « bac » dans ce texte, en ignorant les majuscules et les accents, on trouve trois occurrences :

> Du journal « Le petit bachelier » : comme les professeurs
> se plaisent à le rabâcher, le baccalaureat est important... !

Un automate à qui l'on donnerait ce texte devrait donc accepter les trois *préfixes* suivant du texte, tous ceux qui finissent par le motif « bac » :

- Du journal « Le petit bac
- Du journal « Le petit bachelier » : comme les professeurs se plaisent à le rabâc
- Du journal « Le petit bachelier » : comme les professeurs se plaisent à le rabâcher, le bac

Repérer une occurrence du motif revient donc à accepter dans l'automate le préfixe du texte qui termine sur cette occurrence.

Considérons toujours la recherche du motif « bac » mais dans des textes sur l'alphabet simplifié $\{a, b, c\}$, pour éviter d'obtenir un automate difficile à représenter visuellement. Dans ce cas, les mots *aabbbac* et *aabacbbac* doivent être acceptés, mais pas les mots *aabacbba* (même s'il contient *bac* puisqu'il ne termine pas par *bac*), *acababbaa* ou *babc*. L'automate doit accepter

le mot *bac* lui-même donc il est raisonnable de commencer par créer quatre états permettant de lire successivement les lettres b, a puis c. Le premier état doit être initial, et le dernier acceptant. Il faut ensuite *compléter* les autres transitions de l'automate en maintenant sa correction. Par exemple, dans l'état initial, il faut pouvoir également lire les lettres a et c : celles-ci ne font pas avancer dans la reconnaissance d'une occurrence du motif, donc, en les lisant, on boucle sur l'état initial. De même, si on lit un c dans le second état, on remet à zéro notre avancée dans la recherche du motif *bac*, donc on retourne dans l'état initial. En revanche, si on lit un b dans le second état, on doit rester dans cet état, puisqu'on continue à voir la première lettre du motif *bac*. En continuant de compléter les transitions jusqu'au dernier état, on obtient l'automate suivant :