Automates

1 Introduction

Depuis la ligne de commande, naviguez vers un répertoire vous appartenant, et exécutez la commande suivante, qui téléchargera et décompressera deux répertoires nommés `automates` et `turing` contenant quelques sources C et un makefile :

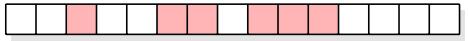
```
curl cdn.sci-phy.org/mp2i/tp4-automates.tgz | tar xvz
```

2 Automates unidimensionnels

2.1 Principe

Dans cette première partie, on s'intéresse à des automates cellulaires unidimensionnels. Ils consistent à faire évoluer un tableau contenants n booléens avec des règles simples.

Par exemple, considérons le tableau initial suivant, où les `false` sont représentés par des cases claires et les `true` sont représentés par des cases sombres.



Considérons par exemple la règle suivante : à l'étape $k+1$, une case contient `true` si et seulement si elle contenait `false` à l'étape k et qu'une et une seule de ses deux voisines (si elles existent) contenait `true`.

Avec cette règle, à l'étape suivante, le tableau contiendra donc :

Et à l'étape qui suit :

2.2 Implémentation

1. Proposer une fonction `void display(bool arr[], int n)` qui affiche, au moyen d'appels à `printf`, le tableau de booléens désigné par `tab` dans la sortie standard. Chaque `true` sera représenté par le caractère « `#` », chaque `false` par le caractère « `.` ». On terminera l'affichage par un retour à la ligne (représenté dans une chaîne par le code `"\n"`).

L'affichage du premier tableau doit donc donner, dans la sortie standard :

```
..#..##.###....
```

2. Écrire une fonction `rule_A(bool src[], bool dst[], int n)` prenant en argument deux tableaux de taille `n` et cette même longueur `n` et construit, dans `dst`, l'état à l'étape $k+1$ en supposant que `src` contient l'état à l'étape k.

3. Compléter la fonction `exec(bool arr_init[], int n, int nb_steps)` prenant un état initial sous la forme d'un tableau `arr_init` et sa taille `n` ainsi qu'un nombre d'itération `nb_steps`, de façon à ce qu'elle affiche l'état initial, puis qu'elle exécute `nb_steps` itérations de l'algorithme et affiche, sur des lignes successives, l'état après chaque itération. La fonction **ne devra pas modifier le tableau qui lui est passé en paramètre**. Elle aura sans besoin de tableaux pour mémoriser l'état courant (et effectuer les calculs). On pourra exceptionnellement utiliser la déclaration

```
int arr[n];
```

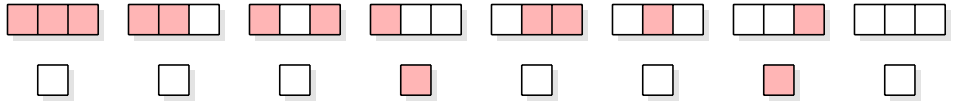
pour créer un « tableau » de taille `n` (même s'il ne s'agit techniquement pas réellement d'un vrai tableau et que le langage C n'impose plus que cette écriture soit valide). Précisons qu'un tel « tableau » ne peut être initialisé que via une boucle.

4. Tester cette fonction en l'appelant depuis `main` afin d'effectuer `nb_steps`$= 10$ itérations sur le tableau fourni en exemple.

5. Modifier le programme précédent pour que le tableau de départ soit un tableau de 79 cases [1] avec des `false` dans toutes les cases excepté dans la case centrale, et afficher les 50 premières itérations.

2.3 Automates de Wolfram

Cet automate fait partie d'une famille de 256 automates étudiés par Stefen Wolfram à la fin du siècle dernier. Pour ces automates, il est possible d'obtenir le contenu de la case `i` à l'étape $k+1$ à partir du contenu des cases $i-1$, i et $i+1$ à l'étape k. On peut donc simplement représenter ce qui se passe pour les $2^3 = 8$ cas possibles pour ces trois cases. Par exemple, pour notre premier automate, les règles sont les suivantes :



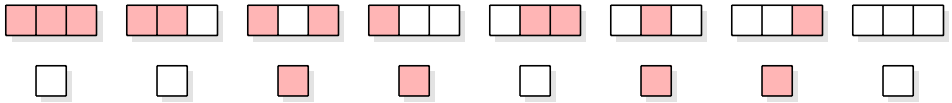
On considère généralement qu'il s'agit de l'automate numéro 18 : le numéro de l'automate est construit en considérant l'ensemble des huit résultats comme huit bits, avec `false`$= 0$ et `true`$= 1$ (soit ici `00010010`), formant ainsi un nombre entre 0 et 255.

1. On pourra modifier la taille de ce tableau en fonction de la longueur des lignes du terminal utilisé. Sur de nombreux terminaux, il est possible d'augmenter ou réduire la taille des caractères, donc de jouer sur le nombre de caractères sur chaque ligne, en utilisant la touche « control » conjointement avec les touches « `+` » et « `-` » du pavé numérique.

Il y a cependant une difficulté en ce qui concerne les extrémités. Dans la suite, on supposera des conditions aux bords périodiques : la case située à gauche de la case la plus à gauche du tableau est la case à l'extrémité droite, et inversement.

6. Écrire une fonction `rule_W(bool src[], bool dst[], int n, bool rules[])` prenant en argument deux tableaux de taille `n` et cette même longueur `n` ainsi qu'un tableau `rules` de taille 8 contenant l'état d'une cellule dans les huit cas possibles (dans l'ordre du dessus) et construit, dans `dst`, l'état à l'étape $k+1$ en supposant que `src` contient l'état à l'étape k.

L'automate 54 (`00110110`) a les règles suivantes :

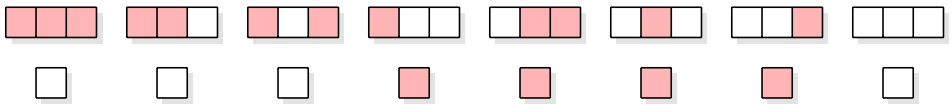


7. Modifier le programme pour qu'il affiche les 50 premières itérations de l'automate 54, pour un état de départ constitué d'une unique case `true` située au centre du tableau.

8. Essayer avec d'autres automates (parmi les cas intéressants, on pourra considérer les automates 13 (`00001101`), 22 (`00010110`), 45 (`00101101`), 57 (`00111001`), 60 (`00111100`), 92 (`01011100`), 105 (`01101001`), 110 (`01101110`)[2], 129 (`10000001`) ou 156 (`10011100`)).

Vous pourrez, si vous le souhaitez, vérifier que le résultat obtenu est bien le bon en le comparant aux images que l'on trouve sur l'encyclopédie en ligne Mathworld, créée par le même chercheur ayant étudié ces automates : `mathworld.wolfram.com/ElementaryCellularAutomaton.html` (mais vous pouvez aussi vous garder le plaisir de la découverte des comportements de ces automates).

Considérons à présent l'automate numéro 30 (`00011110`), régi par les règles suivantes :



Cette règle est particulièrement intéressante car elle présente un comportement chaotique. En fait, les valeurs qu'elle produit sont utilisées comme base d'un générateur aléatoire utilisé par le langage Wolfram, et présente de nombreuses propriétés mathématiques intéressantes.

9. Afficher dans la sortie standard les 50 premières itérations de cet automate, toujours pour un état de départ constitué d'une unique case `true` située au centre du tableau.

10. Avec l'aide du module `basic_ppm`, construire une image noir et blanc de taille 901×450 contenant les 450 premières itérations de cet automate afin de mieux mettre en évidence la nature chaotique de son évolution.

2. Il est à noter que cet automate a la propriété, lorsque l'on choisit bien l'état initial, d'être « Turing-complet », c'est-à-dire en mesure d'exécuter n'importe quel algorithme, ainsi qu'a pu le montrer Matthew Cook.

3 Machines de Turing

3.1 Principe

Une *machine de Turing* est un automate particulier utilisant un ruban, en principe infini, sur lequel il est possible de lire et d'écrire des données grâce à une tête de lecture/écriture mobile.

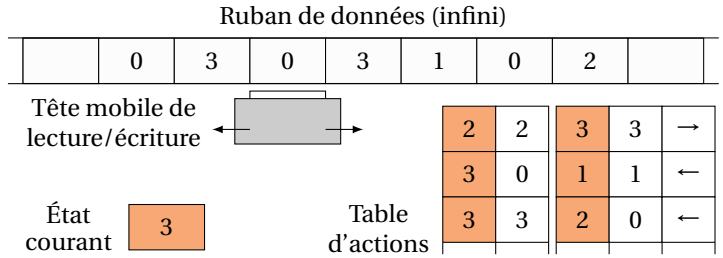
On suppose que les données que l'on peut écrire sur le ruban sont les entiers entre 0 et 10 (inclus), la valeur 10, auquel on peut référer par « `SPACE` » étant une valeur spéciale indiquant que rien n'est inscrit sur le ruban à cet endroit (écrire la valeur `SPACE` à un emplacement du ruban a pour effet d'effacer ce qui pouvait y être écrit préalablement).

Dans la suite, le ruban[3] sera représenté sous la forme d'une suite de cases, vides si elles correspondent à une valeur enregistrée égale à 10 (`SPACE`), et contenant la valeur mémorisée à cet emplacement du ruban sinon sinon.

On dispose d'un module C `ruban` fournissant les fonctions suivantes :

- une fonction `int read(void)` lisant et retournant la valeur inscrite sur le ruban
- une fonction `void write(int data)` écrivant la valeur passée en argument sur le ruban
- une fonction `void left(void)` et une fonction `void right(void)` permettant de se déplacer la tête de lecture/écriture respectivement d'un emplacement vers la gauche et vers la droite sur le ruban
- une fonction `void init(int tape[], int length)` prenant un tableau et sa longueur n et initialisant le contenu du ruban, aux positions 0 à $n-1$, avec les valeurs fournies
- une fonction `void show_tape(int imin, int imax)` prenant deux arguments entiers `imin` et `imax`, et affichant dans la sortie standard le contenu du ruban entre les positions `imin` et `imax` et mettant en évidence la case « zéro » et la position courante de la tête de lecture/écriture.

Le ruban se comporte comme un ruban infini dans les deux directions, les seules limites à ce que vous pourrez y écrire sont liées à la mémoire disponible sur votre ordinateur.



3. ou, plus précisément, la partie du ruban qui nous intéresse, puisqu'il s'agit d'un ruban infini.

A tout instant, la machine de Turing se trouve dans un état particulier, identifié par un entier entre 0 et 255 (inclus). Les trois derniers états, numérotés 253 à 255, sont particuliers : lorsque la machine se trouve dans l'un des ces derniers états, elle s'arrête. L'état 255, auquel on peut faire référence par le code `ERROR`, correspond à une erreur lors de l'exécution du programme. Les états 253 et 254, désignés respectivement par `ACCEPT` et `REJECT`, sont un moyen pour la machine de retourner un résultat au programme.

À chaque instant, si la machine n'est pas dans un des trois derniers états, elle effectue les opérations suivantes :

- elle lit la valeur v à la position actuelle du ruban ;
- puis, en fonction de cette valeur et de son état e, elle effectue trois opérations :
  — elle écrit une valeur sur le ruban ;
  — elle déplace la tête de lecture d'un cran vers la gauche ou la droite ;
  — elle modifie son état.

Les trois opérations sont gouvernées par trois tableaux de taille 252×11 :

- un tableau `prog_new_value[e][v]` d'entiers (`int`) (dont les valeurs doivent être entre 0 et 10 inclus) qui indique la valeur à écrire sur le ruban, si l'état actuel est `e` et la valeur lue est `v` ;
- un tableau `prog_new_state[e][v]` d'entiers (`int`) (dont les valeurs doivent être entre 0 et 255 inclus) qui indique le nouvel état de la machine, si l'état actuel est `e` et la valeur lue est `v` ;
- un tableau `prog_move_right[e][v]` de booléens (`bool`) qui indique s'il faut déplacer le ruban vers la droite (`true`) ou vers la gauche (`false`), si l'état actuel est `e` et la valeur lue est `v`.

L'état courant de la machine est mémorisé dans la variable globale [4] `current_state`.

Dans le fichier `turing.c`, quelques fonctions sont déjà disponibles :

- une fonction `void show(void)` qui affiche sur la sortie standard [5] la stuation courante (état du ruban et état de la machine) ;
- une fonction `void clear_program(void)` qui efface le programme courant ;
- une fonction `void load_program_to_1(void)` qui charge un programme d'exemple (voir ci-dessous).

11. Compléter la fonction `set_instruction(int, int, int, int, bool)` pour qu'elle ajoute dans le programme une instruction indiquant que dans l'état `state`, si elle lit la valeur `value`, alors la machine écrira la valeur `new_value` sur le ruban, effectuera un déplacement selon la valeur du booléen `move_right` et se placera dans l'état `new_state` en modifiant les trois tableaux précédemment décrits.

12. Modifier la fonction `execute` pour qu'elle exécute le programme de la machine de Turing jusqu'à atteindre un état provoquant son arrêt.

On définit un *mot* sur le ruban comme une séquence contigue de valeurs entre 0 et 9 non-interrompue par une espace, le nombre de valeurs constituant la *longueur* du mot. Par exemple, ci-dessous, on a deux mots de longueur respective 5 et 3 :

			0	1	0	1	1		2	3	1			

Le programme d'exemple `program_to_1` remplace tous les valeurs constituant le premier mot situé à droite de la position initiale de la tête de lecture par des 1, puis s'arrête (en se plaçant dans l'état `ACCEPT`).

13. Vérifier la bonne exécution du programme en appelant `execute` depuis la fonction `main` (après avoir initialisé le ruban et chargé le programme).

3.2 Écriture de programmes

Il vous est à présent demandé d'écrire et tester des programmes effectuant les opérations suivantes. **Dans tous les cas, on pourra supposer qu'à la position initiale de la tête de lecture se trouve une espace**, et qu'il y a au moins un mot (si nécessaire plusieurs) à la droite du ruban.

14. `program_move_start` : déplacer la tête de lecture sur la valeur la plus à gauche du premier mot (situé à la droite de la position de départ sur le ruban) puis s'arrêter.

15. `program_move_end` : déplacer la tête de lecture sur la valeur la plus à droite du premier mot puis s'arrêter.

16. `program_append_1` : ajouter un 1 à la fin du premier mot (à son extrémité droite, dans la première case contenant `SPACE` suivant les cases occupées par le mot [6]).

17. `program_delete` : effacer le premier mot situé à droite de la position initiale de la tête de lecture.

18. `program_swap` : on suppose que le premier mot est uniquement constitué de 0 et de 1 ; remplacer les 0 par des 1 et inversement.

19. `program_is_length_even` : déterminer si la longueur du premier mot est paire (si c'est le cas, on arrêtera la machine en se plaçant dans l'état `ACCEPT`, sinon se se plaçant dans l'état `REJECT`, la position finale de la tête de lecture est sans importance).

20. `program_shift_left` : décaler le premier mot d'un cran vers la gauche.

21. `program_concatenate` : on suppose que le ruban contient, au moins, à la droite de la position initiale de la tête de lecture, deux mots séparés par une unique espace ; concatener les deux mots.

22. `program_is_palindrome` : On suppose le premier mot constitué uniquement de 0 et de 1 ; déterminer si ce mot est un palindrome (la fonction peut détruire le mot sur le

4. Cette fois encore, que le programme, l'état courant, la position du ruban et son contenu soient des variables globales n'est guère satisfaisant, mais pour ce que nous en feront aujourd'hui, ce sera suffisant.

5. En fonction de la largeur du terminal que vous utilisez, n'hésitez pas à changer les dimensions du ruban affiché.

6. Il est possible que cela ait pour conséquence de « coller » le mot au mot suivant.

ruban si besoin).

23. `program_Collatz` : on suppose que le ruban contient un unique mot constitué uniquement de 1, toujours placé initialement strictement à droite de la position initiale ; si la longueur de ce mot est paire, réduire la longueur du mot de moitié, sinon tripler sa longueur (toujours avec des 1) et ajouter un 1 supplémentaire à la fin ; il est permis de changer la position du début du mot si c'est utile ; dans les deux cas, la machine doit s'arrêter avec la tête de lecture placée strictement à gauche du mot, afin qu'il soit possible d'exécuter le programme plusieurs fois de suite.