TP #2.3 — Automates cellulaires à une dimension

Le but de cet exercice est d'étudier quelques exemples d'*automates cellulaires à une dimension*. De tels automates sont définis par :

— un état : un ensemble ordonné —dans notre cas unidimensionnel (ou « linéaire »)— de N *cellules*, qui peuvent prendre plusieurs valeurs —dans notre cas 0 ou 1. (Dit autrement, on a N cellules $c_0, c_1, \dots c_{N-1}$ à valeur 0 ou 1.) ;
— une règle d'évolution définie sur les ensembles de cellules.

On peut définir une notion de *calcul* pour un automate cellulaire : soit un automate $\mathbb{A}$ dont l'état courant vaut $\mathcal{C}$ et dont la règle d'évolution est φ, une *étape de calcul* pour $\mathbb{A}$ consiste simplement à remplacer $\mathcal{C}$ par l'application de φ à $\mathcal{C}$, ce que l'on note $\mathcal{C} \leftarrow \varphi(\mathcal{C})$.
Dans tout le sujet, on représente l'état d'un automate (un ensemble (unidimensionnel) de N cellules) par un tableau « `state` » de N `int`s `1` ou `0`, où la cellule c_i est stockée à l'indice i.

1. Écrivez une fonction C de signature :
```
void print_state(size_t n, int state[n])
```
qui affiche (sur la sortie standard) la valeur de l'ensemble de cellules représenté par `state`. On utilisera pour cela les conventions d'affichage suivantes :

— on affiche les cellules de gauche à droite par indice croissant, sur une seule ligne
— une cellule à 1 est affichée avec le caractère `*`, et une cellule à 0 avec le caractère ␣ (une espace).

Par exemple, l'état défini par la variable `int state[4] = {1, 0, 1, 1}` doit être affiché comme :
```
* **
```

2. Testez.

Les règles d'évolution que nous allons considérer sont particulièrement simples : pour celles-ci, l'évolution d'une cellule c_i est entièrement déterminée par sa valeur actuelle et celle de ses voisines « de gauche » c_{i-1} et « de droite » c_{i+1}, où l'on prendra la convention que la voisine de gauche de c_0 et la voisine de droite de c_{N-1} valent toujours 0.
Ainsi, on peut entièrement définir une règle d'évolution φ par deux ensembles de triplets (c_{i-1}, c_i, c_{i+1}) : ceux pour lesquels la cellule c_i évolue en 1 et ceux pour lesquels elle évolue en 0.
Par exemple, l'automate défini par la *règle 30* a le comportement suivant :

— si le triplet $(c_{i-1}, c_i, c_{i+1}) \in \{(1,1,1),(1,1,0),(1,0,1),(0,0,0)\}$, alors c_i évolue en 0
— si le triplet $(c_{i-1}, c_i, c_{i+1}) \in \{(1,0,0),(0,1,1),(0,1,0),(0,0,1)\}$, alors c_i évolue en 1

3. Écrivez une fonction C de signature :
   ```
   void evo30_1(size_t n, int state[n])
   ```
   qui prend en entrée un ensemble de cellules représenté par `state` et qui le modifie en appliquant une fois la règle d'évolution « règle 30 » définie ci-dessus à l'ensemble des cellules.

   Prenez garde à ne pas « écraser » prématurément la valeur d'une cellule c_i qui serait nécessaire pour déterminer l'évolution d'une cellule ultérieure (par exemple c_{i+1}).
4. Testez.
5. Écrivez une fonction C de signature :
   ```
   void evo30(size_t n, int state[n], int nsteps)
   ```
   qui prend en entrée un ensemble de cellules représenté par `state`, qui le modifie en appliquant `nsteps` fois la règle d'évolution définie ci-dessus, et qui affiche sur *nsteps* + 1 lignes l'état initial et chacun des états obtenus lors du calcul de cet automate.
6. Testez votre fonction en affichant les états d'une centaine d'étapes de calcul pour des états initiaux variés (par exemple un état initial valant 1 uniquement en son milieu ; sur une dizaine de positions ; sur environ la moitié des positions ; nulle part…).

On remarque maintenant que pour le type d'automate considéré, la règle d'évolution peut être décrite par 8 valeurs 0 ou 1 : il y a $8 = 2^3$ valeurs possibles pour les triplets (c_{i-1}, c_i, c_{i+1}), et il suffit de dire pour chacun d'entre eux si c_i évolue en 0 ou en 1. Soit un automate dont la règle d'évolution φ est ainsi définie par :

— $(0,0,0) \mapsto x_0$
— $(0,0,1) \mapsto x_1$
— $(0,1,0) \mapsto x_2$
— $(0,1,1) \mapsto x_3$
— $(1,0,0) \mapsto x_4$
— $(1,0,1) \mapsto x_5$
— $(1,1,0) \mapsto x_6$
— $(1,1,1) \mapsto x_7$

avec les $x_i \in \{0,1\}$, on dit que φ « est la *règle* x » avec $x := \sum_{i=0}^{7} x_i 2^i$ le nombre dont les x_i's sont les chiffres de l'écriture en base 2. On peut vérifier que cette terminologie est cohérente avec la règle d'évolution qu'on a précédemment appelé « règle 30 ».

7. Écrivez une fonction C de signature :
   ```
   void evoX_1(size_t n, int state[n], uint8_t rule x)
   ```

qui prend en entrée un ensemble de cellules représenté par `state` et la description `x` d'une règle, et qui le modifie en appliquant une fois la « règle *x* ».

8. Testez votre fonction, notamment en comparant son comportement à celui de votre précédente fonction `evo30_1`.
9. Écrivez de même une fonction C de signature :

   ```
   void evoX(size_t n, int state[n], uint8_t rule x,
             int nsteps)
   ```

   qui prend en entrée un ensemble de cellules représenté par `state` et la description `x` d'une règle, qui le modifie en appliquant `nsteps` fois cette règle, et qui affiche sur *nsteps* + 1 lignes l'état initial et chacun des états obtenus lors du calcul de cet automate.
10. Testez votre fonction pour des valeurs d'états initiaux et des règles variées, par exemple les règles 30 ; 77; 90; 110; 177…
11. Écrivez une fonction C de signature :

    ```
    void evoX4ever(void)
    ```

    faisant tourner une boucle infinie demandant à un utilisateur ou utilisatrice d'entrer une règle, et affichant les états d'une centaine d'étapes de calcul pour cette règle depuis un état initial de votre choix.
    Pensez à correctement gérer le cas où la règle demandée n'est pas définie.
12. Testez.