Projet – Métasimulation

Version du 18 Mars 2025

Le but de ce projet est de simuler des automates cellulaires à une dimension et de s'en servir pour simuler une machine de Turing.

1 Automate cellulaire

Un **automate cellulaire** est un modèle de calcul qui travaille sur une grille infinie de cellules, chacune pouvant prendre un état parmi un ensemble fini. L'évolution de l'automate s'effectue selon des règles locales appliquées à chaque cellule en fonction de son état actuel et de celui de ses voisines. Attention, ces automates travaillent sur de la mémoire (la grille) et sont différents des automates finis vus au premier semestre.

Dans un **automate cellulaire en une dimension**, la grille est réduite à une ligne de cellules évoluant au fil du temps. Chaque cellule a plusieurs états possibles, souvent 0 ou 1, et son état futur est déterminé par une règle impliquant son état actuel ainsi que ceux de ses voisines immédiates.

Un automate cellulaire unidimensionnel est défini par :

— Un ensemble fini d'états S ;

— Une fonction de transition locale $f : S^3 \to S$ qui détermine l'état futur d'une cellule en fonction de son état actuel et de celui de ses deux voisines immédiates (gauche et droite).

Pour définir l'évolution de l'automate, on donne

— Une configuration initiale finie $c^0 = (c_1^0, c_2^0, ..., c_n^0)$, où chaque $c_i^0 \in S$ pour $i \in \{1, ..., n\}$

— On calcule c^{t+1}, la configuration au temps t par la relation suivante pour chaque cellule i :

$$c_i^{t+1} = f(c_{i-1}^t, c_i^t, c_{i+1}^t). \tag{1}$$

Pour pouvoir avoir des configurations de taille arbitraire, on peut considérer que les cases non définies ont une valeur par défaut $\square$ comme les machines de Turing. Dans ce cas, il faut avoir la transition $f(\square, \square, \square) = \square$ pour que les modifications n'aient lieu que sur un mot fini (qui peut néanmoins grossir sur ses bords).

Exemple

La **règle 110** est un automate cellulaire élémentaire défini par la fonction de transition suivante :

Configuration $(c_{i-1}^t, c_i^t, c_{i+1}^t)$	Nouvel état c_i^{t+1}
(1,1,1)	0
(1,1,0)	1
(1,0,1)	1
(1,0,0)	0
(0,1,1)	1
(0,1,0)	1
(0,0,1)	1
(0,0,0)	0

Donnons l'évolution de la règle 110, sur la configuration initiale 0001000 :

$$0001000 \to 0011000 \to 0111000 \to 1101000$$

2 Simulation de l'exécution d'un automate cellulaire

Le code du projet est à faire en Python en C ou en RUST. Vous devez automatiser votre travail (par exemple avec make, ou un script ou en le codant directement en Python) afin que l'utilisateur puisse en une ligne de commande lancer le code qui s'exécute pour répondre à toutes les question. Par ailleurs, on doit pouvoir aussi lancer les tests de chaque question séparément le plus simplement et rapidement possible. Vous donnerez un fichier README qui explique comment utiliser votre code.

Question 1 : Proposer une structure de données qui permet de représenter un automate cellulaire. Attention, l'espace d'états des cellules doit être préciser (ça n'est pas forcément $\{0, 1\}$ comme dans l'exemple ou le jeu de la vie).

La configuration d'un automate cellulaire est la suite finie d'états obtenue à un temps t de la simulation.

Question 2 : Proposer une structure de données pour représenter la configuration d'un automate cellulaire.

Un automate cellulaire sera stocké dans un fichier texte, une ligne par transition.

Question 3 : Écrire une fonction qui lit un fichier texte contenant le code d'un automate cellulaire et un mot d'entrée et qui initialise la structure de données pour représenter cet automate.

Question 4 : Donner une fonction qui prend en argument un automate cellulaire et une configuration et qui donne la configuration obtenue après un pas de calcul de l'automate.

Question 5 : Écrire une fonction qui prend comme argument un mot et un automate cellulaire et qui simule le calcul de l'automate. Vous proposerez plusieurs modes pour arrêter le calcul :

- après un nombre de pas de calcul donné
- après l'application d'une transition particulière
- quand il n'y a pas de changements entre deux configurations successives

Question 6 : Modifier la fonction précédente pour que, à chaque pas de simulation, la configuration de l'automate s'affiche de manière compréhensible (soit graphiquement, soit sur le terminal).

Question 7 : Donner les automates cellulaires suivants :

- Un automate qui fait grandir sa configuration à l'infini en propageant l'information sur ses bords.
- Un automate qui fait cycler les valeurs de ses cases (les cases doivent prendre toutes les valeurs de S).
- Deux automates de votre choix qui ont un comportement intéressant.

Exécutez ces machines sur des exemples à l'aide de votre simulateur.

3 Une machine de Turing, juste pour comparer

Nous allons implémenter la simulation d'une machine de Turing, de la même manière que pour un automate cellulaire. Toutes les machines de Turing seront sur l'alphabet de travail $\{0, 1, \square\}$ et n'auront qu'un seul ruban.

Question 8 : Proposer une structure de données qui permet de représenter une machine de Turing.

Question 9 : Proposer une structure de données pour représenter la configuration d'une machine de Turing.

Une machine de Turing sera stockée dans un fichier texte, une ligne par transition. Il peut y avoir plus d'informations dans ce fichier et si certaines informations sont implicites (état initial, acceptant ...), vous les préciserez en commentaire.

Question 10 : Écrire une fonction qui lit un fichier texte contenant le code d'une machine de Turing et un mot d'entrée et qui initialise la structure de données pour représenter cette machine.

Question 11 : Donner une fonction qui prend en argument une machine de Turing et une configuration et qui donne la configuration obtenue après un pas de calcul de la machine.

Question 12 : Écrire une fonction qui prend comme argument un mot et une machine de Turing, qui simule le calcul de la machine sur ce mot et qui s'arête sur un état ACCEPT ou REJECT.

4 La simulation finale

L'objectif de ce devoir est d'implémenter le théorème qui montre que les automates cellulaires peuvent simuler les machines de Turing. Nous allons simuler une machine M qui travaille sur l'alphabet $\Sigma = \{0, 1, \square\}$, avec des états Q et des transitions $T = (T_1, \ldots, T_m)$. Pour cela nous créons un automate cellulaire A sur l'espace d'états $(Q \cup \{\star\}) \times \Sigma$ et avec la fonction de transition f.

Une configuration de A aura la forme $(\star, l_0), \ldots, (q, l_i), \ldots, (\star, l_n)$, cela représente la configuration de la machine de Turing M : $(q, i, l_0 l_1 \ldots l_n)$, c'est à dire que M est dans l'état q avec la tête de lecture sur la $i$ème position et le mot sur la bande est $l_0, \ldots, l_n$. Sur une configuration, on aura une seule valeur de la forme (q, l), avec $q \in Q$, toutes les autres auront la forme $(\star, l)$.

Pour faire évoluer les configurations de A comme celles de M, il faut simuler les transitions données par T. Quand on aura un triplet $((\star, l_1), (\star, l_2), (\star, l_3))$ on ne fait aucun changement, car la tête de lecture n'est pas là. Par contre, si on a $((\star, l_1), (q, l_2), (\star, l_3))$, on va simuler la transition $(q, l_2, q', l', D) \in T$. Par exemple, si on a $(q_1, 0, q_2, 1, R) \in T$ alors on va définir $f((\star, 0), (q_1, 0), (\star, 1)) = (\star, 1)$ et $f((q_1, 0), (\star, 1), (\star, 1)) = (q_2, 1)$.

Question 13 : Programmer une fonction qui prend en entrée le code d'une machine de Turing et construit l'automate cellulaire qui la simule. Tester sur plusieurs exemples que la machine de Turing et sa simulation par un automate cellulaire font le même calcul.

Au regard de ce que vous venez de programmer, répondez à la question théorique suivante.

Question 14 : Soit le problème HALTING-CELLULAR-AUTOMATON : étant donné $< A >$ le code d'un automate cellulaire, $s \in S$ et un mot $w \in S^*$, décider si A sur l'entrée w va avoir une configuration contenant s lors de son calcul. Est-ce que HALTING-CELLULAR-AUTOMATON est décidable ?

5 Modalités pratiques

Merci de respecter les consignes suivantes :

— le projet est à faire en binôme ;

— n'importe quel membre du binôme doit pouvoir expliquer en détail n'importe quel partie du projet ;

— le projet est à rendre sur ecampus, au plus tard le dimanche 4 mai à 23h59 ;

— une soutenance aura lieu la semaine du 5 mai.

— vous devez déposer un fichier `XY_NOM1_Prenom1-NOM2_Prenom2.zip` qui est le zip du dossier `XY_NOM1_Prenom1-NOM2_Prenom2` contenant votre projet.
`XY` sont les initiales de votre chargé de TD (`PC`, `ND`, `YS`). Pour être évalué, votre travail doit s'exécuter quand on tape make dans le terminal. Si le nom de fichier remis sur ecampus ne respecte pas ces règles, le projet ne sera pas évalué.

— Un outil de détection de plagiat sera utilisé sur vos codes.

Le retard de la remise du projet entraîne 1 point de moins par heure de retard.