



UE : Théorie des langages et compilation

# Autosim

Simulateur d'automates finis

As

## Membres du groupe

Filière : ISN L3 Année : 2022-2023

- AZEUFACK NDOMO Aurelle
- BAKEHE William Stève
- ELLA Kris David Steeve
- EMANE BILE Félicien Davy
- ENGBWANG NGO'O Éric Hans
- IBOUANGA Alvin
- LIEUMO NJOPNU Marcelle A.
- MATAKON NGAROUA Modeste
- MAVAIWA DÉSIRÉ Désiré *chef*
- MENYE Pegguy Flora
- NDONGA Isept Dannielle
- NDJESSE LETERE Emmanuel A.
- NOUMA Ludovic
- OUSMANE MERIGA
- SANDJIO NGUENANG Patrick
- VOUNDI ABESSOLO Paul Yvan

Enseignant : Dr KOUOKAM Étienne



## Introduction

Depuis les années 50, le monde du numérique a connu plusieurs évolutions parmi lesquelles la création de plusieurs langages de programmation. Parmi ces derniers, existent des langages qui ont besoin d'un compilateur dont le rôle est de traduire le code source en un langage compréhensible par une machine. La reconnaissance du dit langage et des règles (syntaxe) qui lui sont appliquées se fait de façon automatisée grâce à des machines abstraites appelées automates finis. Un automate fini est un modèle mathématique capable de reconnaître l'appartenance ou non d'un mot à un langage régulier donné. Il en existe plusieurs types dont le choix dépend du besoin de l'utilisateur. Le but de ce projet est donc de mettre sur pied un outil logiciel qui dans un premier temps permettra de simuler un automate fini quelconque et dans un second temps va permettre de déterminer ou de minimiser un automate fini à partir d'un automate fini donné.



### 1. Description générale du projet

Le projet **AutoSim** (*Automata Simulator*) consiste en la réalisation d'un simulateur d'automate fini, avec la possibilité d'effectuer d'autres opérations sur un automate fini donné. Il a été réalisé en utilisant l'approche orienté objet. Quant au langage utilisé, nous avons opté pour le langage Java car c'est le langage connu par la plupart des membres de l'équipe d'autant plus qu'il a un fort support communautaire. Pour la réalisation de notre projet, nous proposons dans notre logiciel 3 principales fonctionnalités : La simulation d'un automate, la détermination d'un automate et la minimisation d'un automate.

- **La simulation d'automates finis** : il est question ici de mettre sur pied un programme qui va permettre à un utilisateur donné d'entrer les données de l'automate fini qu'il dispose. Le système devra concevoir cet automate et lui donner la suite la possibilité de tester l'appartenance ou non de certaines chaînes au langage de l'automate entré.
- **La détermination d'un automate fini** : un automate est dit déterministe s'il possède un unique état initial et pour chaque état de cet automate, il existe au maximum une transition issue de cet état et possédante le même symbole. Afin de réaliser cela, notre programme va recueillir le quintuplé d'informations sur un automate fini non déterministe et sur la base de ses



informations, le système renverra un nouveau quintuplet correspondant à l'automate déterministe équivalent.

- **La minimisation d'un automate fini déterministe** : la minimisation d'un AFD consiste à améliorer la reconnaissance des chaînes en supprimant les états inutiles jusqu'à obtenir un AFD minimal. Pour cela, nous mettons sur pied un programme qui va recueillir les informations nécessaires sur l'AFD considéré. Ensuite, il va alors renvoyé (dans le cas où la minimisation est possible) le quintuplet équivalent au nouvel automate minimal obtenu.



## II. Présentation des différentes fonctionnalités

### 1. Simulation des automates finis

#### a) Description processuelle de la solution

Notre programme est structuré en classes et en méthodes :

- On récupère notre quintuplet  $(\Sigma; Q; q_0; F; \delta)$  où  $\Sigma$  est l'alphabet (une liste de caractères),  $Q$  est la liste des états du système (une liste de caractères),  $q_0$  est l'état initial (un caractère),  $F$  est la liste des états finaux (une liste de caractères) et  $\delta$  est la table des états de transitions (qui est une matrice où chaque ligne représente l'état actuel, chaque colonne représente l'état suivant et chaque cellule représente l'étiquette de chaque transition).
- Premièrement, on récupère l'état initial et le caractère consommé, puis on vérifie si le caractère appartient à notre alphabet en parcourant la liste des symboles de notre alphabet. S'il n'appartient pas, alors on conclut que le caractère n'appartient pas à notre alphabet si oui, on entre alors dans une boucle pour déterminer l'état suivant correspondant qui devrait correspondre. Pour cela, on considère la table des états de transitions, on choisit l'état actuel et on parcourt toutes les colonnes de la matrice pour vérifier s'il existe une transition de notre état actuel vers un autre état avec pour étiquette le caractère considéré. Si oui on récupère l'état suivant correspondant et on le stocke, sinon on conclut alors qu'il n'existe pas de transitions correspondant au caractère saisi.



- Pour vérifier l'appartenance d'une chaîne au langage d'un automate, on entre dans une boucle qui va parcourir le mot entré par l'utilisateur. Dans la boucle, on récupère successivement chaque caractère du mot qu'on place en paramètre avec l'état qui va nous retourner l'état suivant. Puis on va écraser l'ancienne valeur de notre variable **etatActuel** et stocker le résultat retourné par la méthode précédente. La boucle va donc se répéter jusqu'à la fin du mot. Après être sorti de la boucle, on vérifie si la valeur de l'état actuel appartient à la liste de nos états finaux. Si oui alors on conclut que le **mot appartient au langage reconnu par l'automate** sinon on conclut que le mot **n'appartient pas au langage reconnu par l'automate**. Dans le cas où le mot n'est pas reconnu, le système affiche l'erreur à l'écran.

#### a) Limites de la solution

- L'automate ne reconnaît que des mots de langages appartenant à un alphabet constitué uniquement de caractères.
- L'utilisateur devra entrer des automates avec un seul état initial

## 2. Détermination d'un automate fini

#### b) Description processuelle de la solution

Après avoir recueilli les informations de l'AFN à déterminer (ses différents états et ses différentes transitions), on procède comme suit :

- **Créer un tableau** (qui est en réalité la matrice de transition de notre AFD) à 2 dimensions de type chaîne ayant comme nombre de colonnes celui de la matrice initiale (de l'automate à déterminer) et comme nombre de lignes un nombre très grand.
- **Initialiser le tableau avec la valeur vide.**
- **Remplir la première ligne avec les valeurs de la table de transition** de la matrice de transition de l'automate à déterminer.
- Parcourir la ligne du tableau correspondant à l'état initial, lire successivement les états suivants (les colonnes du tableau) vers lesquels on se déplace lorsqu'on lit un symbole de l'alphabet.



- Vérifier que l'état n'existe pas encore dans la nouvelle matrice de transition. Dans le cas où un état existe ou est vide, **incrémenter une variable x** qui initialement était à 0.
- Si l'état n'existe pas encore dans la nouvelle matrice de transition, **Vérifier que l'état est simple ou composé en cherchant s'il contient une virgule.**
- Si un état suivant est simple, le rechercher dans la table de transition de la matrice de transition initiale et recopier ses valeurs sur une nouvelle ligne dans la nouvelle matrice de transition. Si l'état est composé, **le décomposer en états simples**, rechercher ces états dans la table de transition de la matrice de transition initiale et concaténer les états vers lesquels on se déplace. Si on lit un caractère de l'alphabet précis en séparant les états par une virgule, **recopier ce nouvel état et les valeurs de transition obtenues** sur une nouvelle ligne dans la nouvelle matrice de transition.
- Après avoir parcouru toute la ligne, **réinitialiser la variable x à 0.**
- Répéter le processus sur chaque ligne de la nouvelle matrice de transition en **réinitialisant à chaque fois la variable x lorsqu'on passe à une nouvelle ligne** jusqu'à ce que la valeur de la variable x soit égale au nombre de colonnes du tableau moins.
- **Marquer chaque état contenant un état final de l'automate initial comme état final.**

### c) Les limites

- Nous avons exclusivement géré la détermination des AFN à un seul état initial.

## 3. Minimisation d'un automate fini déterministe

### a) Description processuelle de la solution

Après avoir recueilli les informations de l'AFN à déterminer (ses différents états et ses différentes transitions), on procède comme suit :

- On regroupe les états en deux grandes partitions : **la première constituée de l'ensemble des états finaux et la seconde est constitué**



**des états non finaux.** Pour cela, on enlève dans l'ensemble Q des états de notre automate l'ensemble des états finaux, ainsi on obtient ces deux ensembles.

- Par la suite on vérifie **la cohérence de chaque partition** c'est-à-dire si pour chaque transition (en consommant la même étiquette), tous les états d'un ensemble aboutissent à des états d'une seule et même partition. Cela se fait en vérifiant si oui ou non pour une cellule donnée, toutes les valeurs d'une même colonne de notre matrice transition appartiennent au même ensemble. Si c'est le cas l'ensemble est cohérent, dans le cas contraire il ne l'est pas. Pour un ensemble non cohérent, on résout le problème en étalant l'ensemble (la décomposition de l'état non cohérent est faite en regroupant pour cet état tous les états qui aboutissent aux états d'un même ensemble entre eux).
- On recommence le processus (à partir de la deuxième étape) jusqu'à l'obtention des partitions uniquement cohérentes. Chaque partition devient donc comme un nouvel état, et considérant les étiquettes qui les lient, on peut générer l'automate minimal correspondant.

#### b) Les limites

- Il est recommandé à l'utilisateur d'entrer les états de son automate (AFDC) sous forme d'entiers positifs partant de zéro (0), ces entiers sont utilisés dans l'algorithme.



### III. Conception et modélisation UML

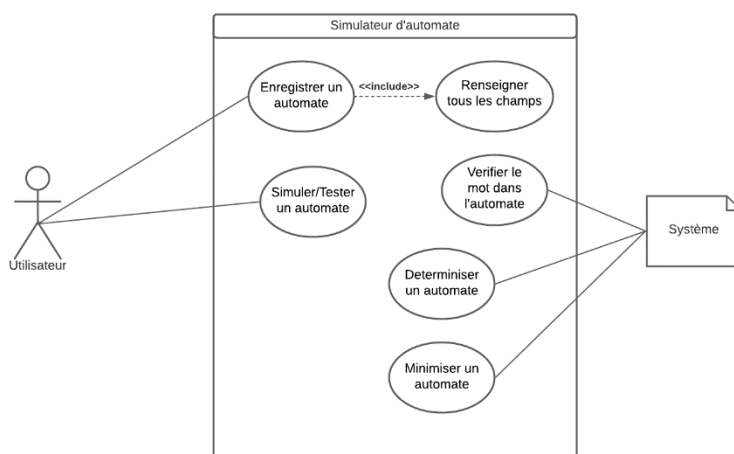


Diagramme de cas d'utilisation

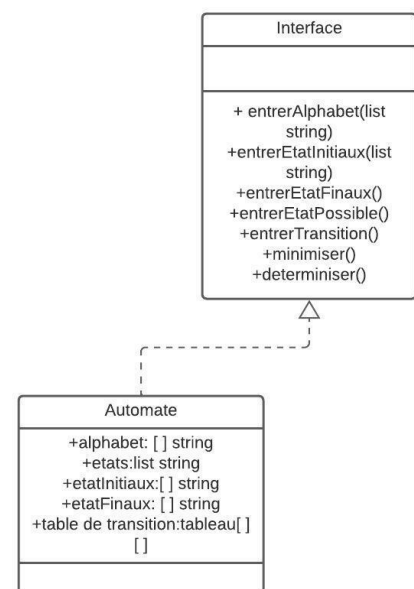
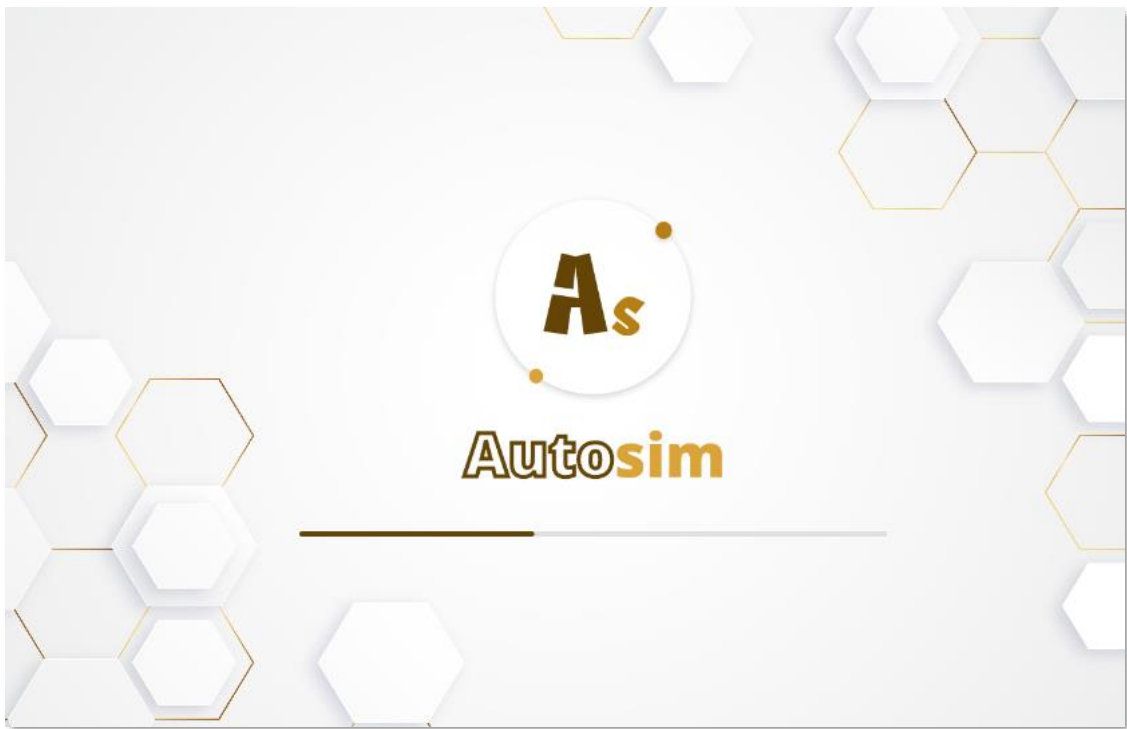


Diagramme de classe



## IV. Interfaces logicielles



Page de chargement

**Entrez un automate**

type d'automate AFD

$\Sigma$

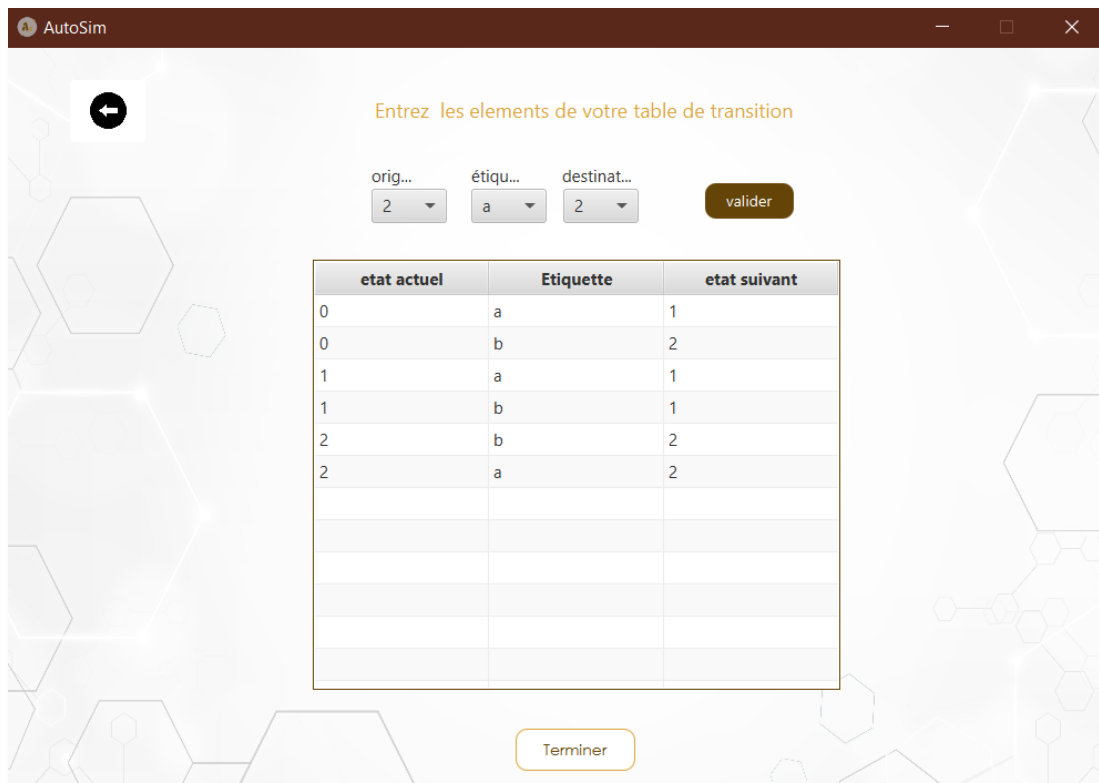
$Q$

$I$

$F$

Poursuivre

Entrer un automate



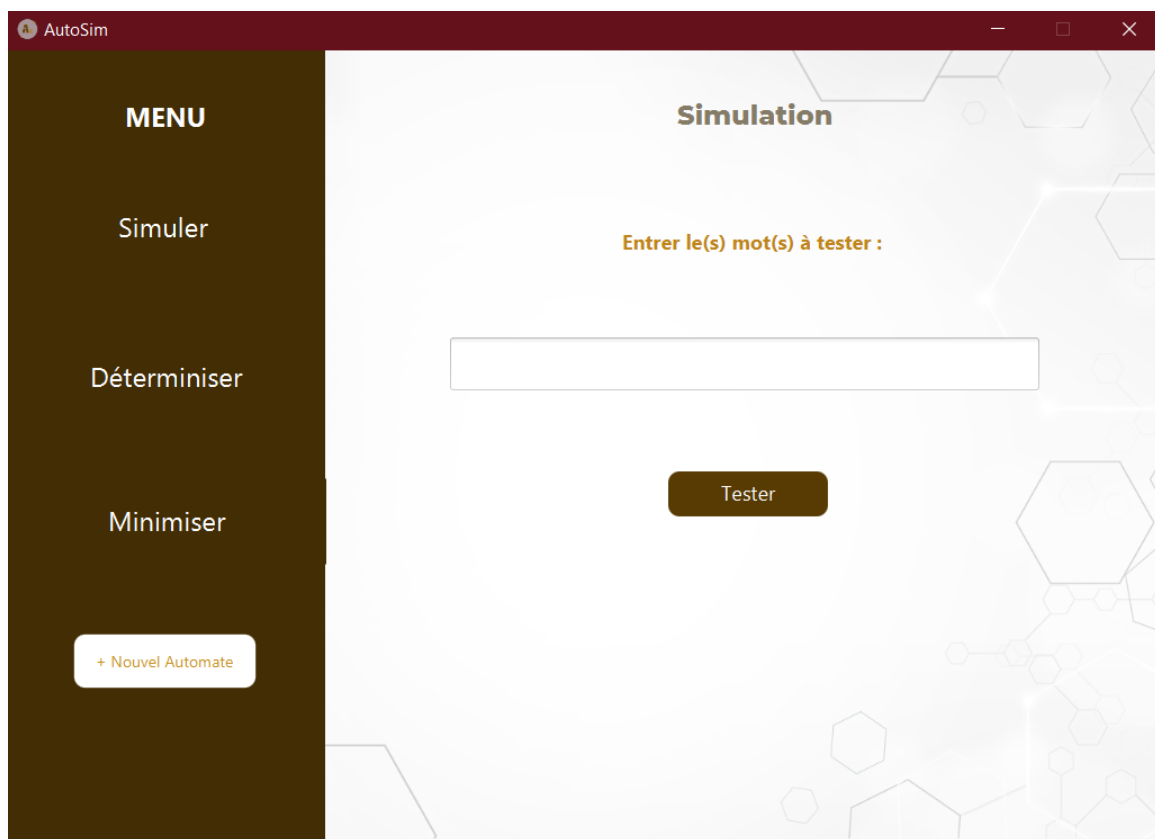
Entrez les elements de votre table de transition

orig... étiqu... destinat... valider

etat actuel	Etiquette	etat suivant
0	a	1
0	b	2
1	a	1
1	b	1
2	b	2
2	a	2

Terminer

Entrer la table de transition de l'automate



**MENU**

Simuler

Déterminiser

Minimiser

+ Nouvel Automate

**Simulation**

Entrez le(s) mot(s) à tester :

Tester

Simulation d'un automate





MENU

Simuler

Déterminiser

Minimiser

+ Nouvel Automate

Détermination

$\Sigma : a,b,$   
 $Q : \{0\},\{1\},\{2\},$

$I : \{0\}$   
 $F : \{1\},\{2\},$

etat initial	etiquette	etat suivant
0	a	1
0	b	2
1	a	1
1	b	1
2	a	2
2	b	2

Détermination d'un automate

MENU

Simuler

Déterminiser

Minimiser

+ Nouvel Automate

Minimisation

$\Sigma : a,b,$   
 $Q : [1, 2] , [0] ,$

$I : \{0\}$   
 $F : [1, 2] ,$

etat de depart	etiquette	etat d'arrivé
[1, 2]	[a]	[1, 2]
[1, 2]	[b]	[1, 2]
[0]	[a]	[1, 2]
[0]	[b]	[1, 2]

Minimisation d'un automate