Université Marie et Louis Pasteur
UFR Sciences et Techniques

Licence 3 Informatique – Année 2025-2026

RAPPORT DE PROJET DE THÉORIE DES LANGAGES

Module TC/PP : Préparation de rapports et soutenances

Développement d'une structure d'automates finis en C++ dans l'UE Théorie des Langages

Auteurs :
Elouan BOITEUX
Aymeric MARIAUX
Killian MATHIAS

Tuteurs :
M. Julien BERNARD (Tuteur Théorie des langages)
M. Jean-Michel HUFFLEN (Tuteur TC/PP)

Remerciements

Nous tenons à remercier M. Julien Bernard pour la qualité du sujet de travaux pratiques et sa disponibilité. Nous remercions également M. Jean-Michel Hufflen pour ses conseils avisés sur la rédaction de ce rapport ainsi que du support de soutenance associé.

Table des matières

1 Introduction

Dans le cadre de notre troisième année de Licence en Informatique, le module de Théorie des Langages (TL) nous invite à explorer les fondements théoriques de l'informatique. Pour valider ces acquis, il nous a été confié la réalisation d'un projet technique conséquent tout au long du semestre.

L'objectif principal était de développer, en langage C++, une structure capable de manipuler des "automates finis". Ce rapport a pour but de présenter ce travail à un public non expert, en expliquant non seulement les concepts techniques, mais aussi notre méthodologie et notre vécu durant ce projet intense.

Nous commencerons par présenter le contexte universitaire et la mission. Ensuite, nous vulgariserons le concept d'automate. Nous détaillerons ensuite notre réalisation technique et les choix effectués, avant d'aborder la gestion du projet et notre bilan personnel.

2 Présentation du contexte de la mission

L'UFR Sciences et Techniques propose un parcours exigeant dans lequel nous étudions à la fois théorie et pratique. Ce parcours nous permet d'avoir les fondements dans tous les domaines de l'Informatique.

Le module Théorie des Langages vise à comprendre comment les machines interprètent les langages. Le projet pratique associé (TP) sert de validation concrète de ces théories.

2.1 Environnement de travail et Contraintes

2.1.1 Les outils imposés

Nous avons travaillé sous environnement Linux, en utilisant le langage **C++**. La compilation est gérée par l'outil **CMake** et la validation par le framework de test **Google Test**.

2.1.2 La difficulté majeure : la création de la structure

Une spécificité forte de ce projet était que la structure n'était pas imposée comme nous avions eu l'habitude jusqu'ici. Il nous fallait donc réfléchir à une structure de donnée performante où l'accès aux données de l'automate soit facile et pas trop coûteux.

3 Qu'est ce qu'un automate ?

3.1 Définition simplifiée pour non-initiés

Imaginez un portique de métro. Il possède un état initial (bloqué). Si vous insérez un ticket valide (une transition), il change d'état (débloqué). C'est cela, un automate : une machine virtuelle composée d'états et de règles de passage d'un état à un autre.

3.2 Utilité concrète

Ces structures sont utilisées partout :
— Dans les compilateurs (pour comprendre le code que vous écrivez).
— Dans les jeux vidéo (pour gérer l'intelligence artificielle des ennemis).
— Dans la recherche de motifs (quand vous faites CTRL+F dans un texte).

3.3 Le défi technique

Notre mission était de traduire ces schémas graphiques (des ronds et des flèches) en lignes de code performantes capable de réaliser des opérations complexes comme l'intersection ou la minimisation d'automates.

4 Travail réalisé et choix techniques

4.1 Architecture de la solution

4.1.1 Choix de la structure de données

Pour représenter un automate en mémoire, nous avions le choix entre plusieurs structures. Nous avons opté pour un `std::set` d'entier pour les états, un `std::set` de caractères pour notre alphabet et un `std::map` qui associe un `std::pair` d'un entier (état de départ) et d'un symbole à un `std::set` d'entiers (états d'arrivée). En ce qui concerne la gestion des états initiaux et finaux, nous avons choisi d'utiliser un `std::set` d'entier qui contient les états initaux et un autre `std::set` d'entier qui contient les états finaux.

Justification : Ce choix permet un accès rapide et performant aux données importantes de notre automate.

4.2 Implémentation des algorithmes clés

4.2.1 La déterminisation

Un automate est dit "non déterministe" si, pour un état donné, il a plusieurs transitions sortantes avec le même symbole. Nous avons donc implémenté un algorithme capable de transformer tout automate non-déterministe en automate déterministe.

4.2.2 La minimisation (Algorithme de Moore/Brzozowski)

Le but est de simplifier l'automate pour qu'il prenne moins de place en mémoire sans changer son comportement. Pour cela, on s'est d'abord appuyés sur deux algorithmes celui de Moore (étudié en TD) et celui de Brzozowski qui consiste à déterminiser la transposée de l'automate deux fois de suite. Par la suite, un méthode bonus nous propose un troisième algorithme de minimisation nommé Hopcroft.

4.3 Validation et Tests

Nous avons adopté une approche TDD (Test Driven Development). Avant même de coder une fonction, nous écrivions le test qui devait échouer. Cela nous a permis de passer les tests de la "moulinette" d'évaluation automatique et d'obtenir un score de 100%.

5 Organisation et Gestion du projet

5.1 Planification

Le projet était découpé en 12 séances de TP de 1h30 chacune. Nous avions également besoin de fournir un travail personnel afin de suivre la cadence.

— **TP 1-2 :** Structure de base et création.
— **TP 3-5 :** Algorithmes complexes (Intersection, Déterminisation).
— **TP 6 :** Minimisation et bonus.

5.2 Analyse du vécu (Difficultés et Réussites)

La principale difficulté a été la gestion des itérateurs C++ qui peuvent être complexes à manipuler au début. Nous avons également rencontré des difficultés à implémenter des algorithmes complexes mais cela était toute fois formateur.

6 Conclusion

6.1 Bilan technique

Notre structure ‘Automaton‘ est fonctionnelle et passe 100% des tests fournis. Elle permet de créer, manipuler et minimiser des automates efficacement.

6.2 Bilan personnel

Ce projet nous a permis de :

— Maîtriser la librairie standard C++ (STD).

— Comprendre les notions abordées durant l'UE.

« Que retenir de ce travail ? » : *Une bonne structure de données est la clé d'un algorithme performant.*

Références

[1] CppReference, *Documentation C++ Standard*, `http://en.cppreference.com/w/cpp/container`.

[2] Graphviz Project, *Graph Visualization Software*, `http://www.graphviz.org/`.

Résumé

Ce rapport présente la conception et le développement d'une bibliothèque en langage C++ dédiée à la manipulation d'automates finis, réalisée dans le cadre de l'Unité d'Enseignement « Théorie des Langages ». L'objectif principal était de traduire des concepts théoriques (déterminisation, minimisation, intersection) en une structure logicielle performante et modulaire. Nous détaillons ici les choix d'architecture, notamment l'utilisation de la librairie standard, ainsi que la méthodologie de développement dirigé par les tests (TDD) adoptée pour garantir la fiabilité du code.

Abstract

This report presents the design and development of a C++ library dedicated to finite automata manipulation, carried out within the "Language Theory" course. The main objective was to translate theoretical concepts (determinization, minimization, intersection) into a high-performance and modular software structure. We detail the architectural choices, specifically the use of the Standard Library , as well as the Test-Driven Development (TDD) methodology adopted to ensure code reliability.