Compte Rendu

Projet de Théorie des Langages

Etudiant 1 : MEZOUER Amîn
Etudiant 2 : SOUSSI Younous

Groupe : TD1

Année universitaire : 2025–2026

---

1. Objectif du projet

L'objectif de ce projet est de comparer deux expressions régulières afin de déterminer si elles reconnaissent le même langage.

Pour cela, nous construisons les automates associés aux expressions régulières, puis nous appliquons différentes transformations classiques sur les automates :

- suppression des ε-transitions,
- déterminisation,
- complétion,
- minimisation.

Deux expressions régulières sont dites équivalentes si, après ces transformations, leurs automates minimisés reconnaissent le même langage.

---

2. Organisation du projet

Le projet est organisé de la manière suivante :

- regexp.l : analyse lexicale (Flex)
- regexp.y : analyse syntaxique (Bison)
- automate.py : implémentation des automates et des opérations
- Makefile : compilation automatique du projet
- test.1, test.2, test.3 : fichiers de tests
- pdf/ : dossiers contenant les dessins des automates pour chaque test
- README : instructions d'exécution

---

3. Analyse lexicale (Flex)

Le fichier regexp.l permet de reconnaître les symboles suivants :

- lettres de l'alphabet (a, b, c),
- opérateurs + (union), . (concaténation), * (étoile),
- parenthèses (et),
- symbole ε (epsilon).

Chaque symbole reconnu est transformé en un token utilisé par l'analyse syntaxique.

---

4. Analyse syntaxique (Bison)

Le fichier regexp.y contient la grammaire des expressions régulières.

Les priorités suivantes sont respectées :

- l'étoile * est prioritaire,
- la concaténation . est prioritaire sur l'union +,
- l'union + est l'opérateur le moins prioritaire.

Les actions associées aux règles produisent du code Python permettant de construire les automates correspondants à chaque expression régulière.

À la fin de l'analyse, un fichier main.py est généré automatiquement.

---

5. Construction et manipulation des automates

Le fichier automate.py implémente :

- la structure d'un automate,
- la suppression des ε-transitions,
- la déterminisation,
- la complétion,
- la minimisation,
- la comparaison de deux automates (egal),
- les opérations sur automates (union, concaténation, étoile).

Des tests unitaires ont été réalisés directement dans automate.py afin de vérifier le bon fonctionnement de chaque fonction.

---

6. Processus de comparaison de deux expressions régulières

Pour comparer deux expressions régulières, les étapes suivantes sont appliquées :

1. Construction de l'automate avec ε-transitions.
2. Suppression des ε-transitions.
3. Déterminisation de l'automate.
4. Complétion de l'automate.
5. Minimisation de l'automate.
6. Comparaison des automates minimisés.

Si les automates finaux reconnaissent le même langage, les expressions régulières sont équivalentes.

---

7. Tests réalisés

Test 1 – Expressions non équivalentes

Expressions testées :

(a+b)*.c

a*.c + b*.c

Résultat : **NON ÉGALES**

Les deux expressions ne reconnaissent pas le même langage.
La première accepte toutes les suites de a et b terminées par c, tandis que la seconde accepte uniquement des suites composées uniquement de a ou uniquement de b, suivies de c.

Les automates obtenus après minimisation sont différents (voir test.1.pdf).

Test 2 – Expressions équivalentes

Expressions testées :

a*

a*.a*

Résultat : **ÉGALES**

Les deux expressions reconnaissent exactement le langage constitué de suites quelconques de a.
Les automates minimisés sont équivalents (voir test.2.pdf).

Test 3 – Expressions équivalentes plus complexes

Expressions testées :

c + (a+b)*

(b+a)* + c

Résultat : **ÉGALES**

Les deux expressions reconnaissent le même langage, à savoir soit le mot c, soit toute suite de a et b. Après minimisation, les automates obtenus sont équivalents (voir test.3.pdf).

---

8. Génération des automates en PDF

Pour chaque test, les automates intermédiaires ont été dessinés et exportés au format PDF :

- automate avec ε-transitions,
- automate sans ε-transitions,
- automate complété,
- automate minimisé.

Ces documents sont fournis dans le dossier pdf/.

---

9. Compilation et exécution

Compilation du projet : make

Génération du fichier Python à partir d'un test : ./regexp < test.1

Exécution de la comparaison : python3 main.py

Nettoyage des fichiers générés : make clean

---

10. Conclusion

Ce projet a permis de mettre en pratique les notions fondamentales de la théorie des langages, notamment la construction et la manipulation d'automates finis.

L'approche par automates minimisés permet de comparer de manière fiable deux expressions régulières, indépendamment de leur forme syntaxique.

Les résultats obtenus sont cohérents avec la théorie et valident le bon fonctionnement du programme.

Note complémentaire

Dans les schémas de nos tests : test.2 et test.3, lorsque nous indiquons que “l’automate est déjà déterministe”, cela signifie *après suppression des ε-transitions*.

De même, la fonction de minimisation utilisée dans automate.py correspond à une version corrigée conforme à l’algorithme de Moore, sans recours à automate2.py.