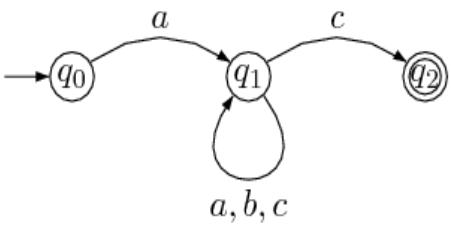
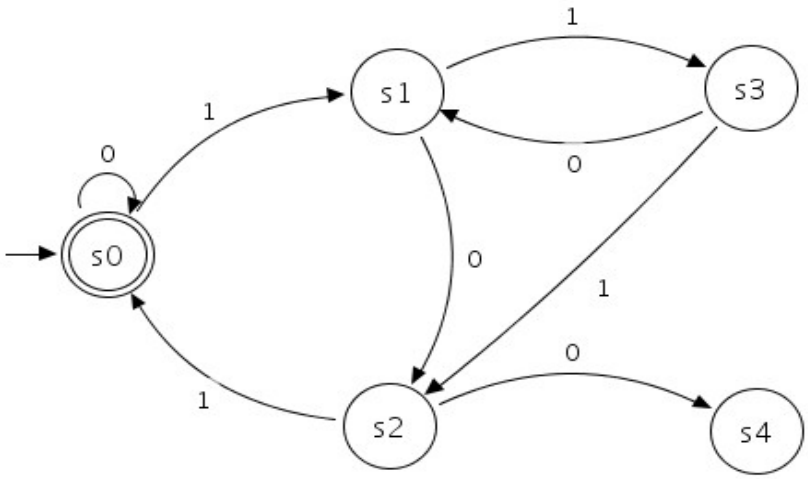


CAHIER DES CHARGES

Projet Automates finis: simulation, opérations



Responsables : Béatrice Bérard & Nathalie Sznajder
Étudiants : Jessica Lourenço & Ervin Milandou

UPMC SORBONNE UNIVERSITÉS

Sommaire

I. Introduction

Les automates finis sont des objets mathématiques, très utilisés en informatique, qui permettent de modéliser un grand nombre de systèmes (informatiques). Ils peuvent ainsi être utilisés pour modéliser des ordinateurs, pour comprendre ce qu'il peut faire et ce qu'il sait faire efficacement. Mais les automates peuvent aussi être utilisés de manière plus pratique pour décrire des machines physiques (circuits électroniques, machines à calculer, distributeurs d'objets …), pour la recherche d'occurrence dans un texte …

II. Présentation du projet

1. Intitulé et intérêts du projet

Notre projet est intitulé « Projet Automates finis : simulation, opérations ». Les automates finis sont étudiés en deuxième année de licence. Cependant certains étudiants ont des difficultés à percevoir la nature informatique de ces objets. Dans le but d'améliorer leur compréhension, il est envisagé de concevoir une séquence de TME.

2. Contraintes du projet

Dans le cadre de notre projet, il nous est demandé de concevoir une séquence de TME pour les étudiants de L2 en Informatique mais aussi en Mathématiques. Ce travail nécessite donc de :

- Maîtriser au mieux les automates finis ainsi que les opérations telles que la déterminisation, les opérations ensemblistes (intersection, union, complémentaire), les opérations rationnelles (produit, étoile) et minimisation.
- Se renseigner sur le langage python
- Adapter notre code à des étudiants qui n'ont jamais coder

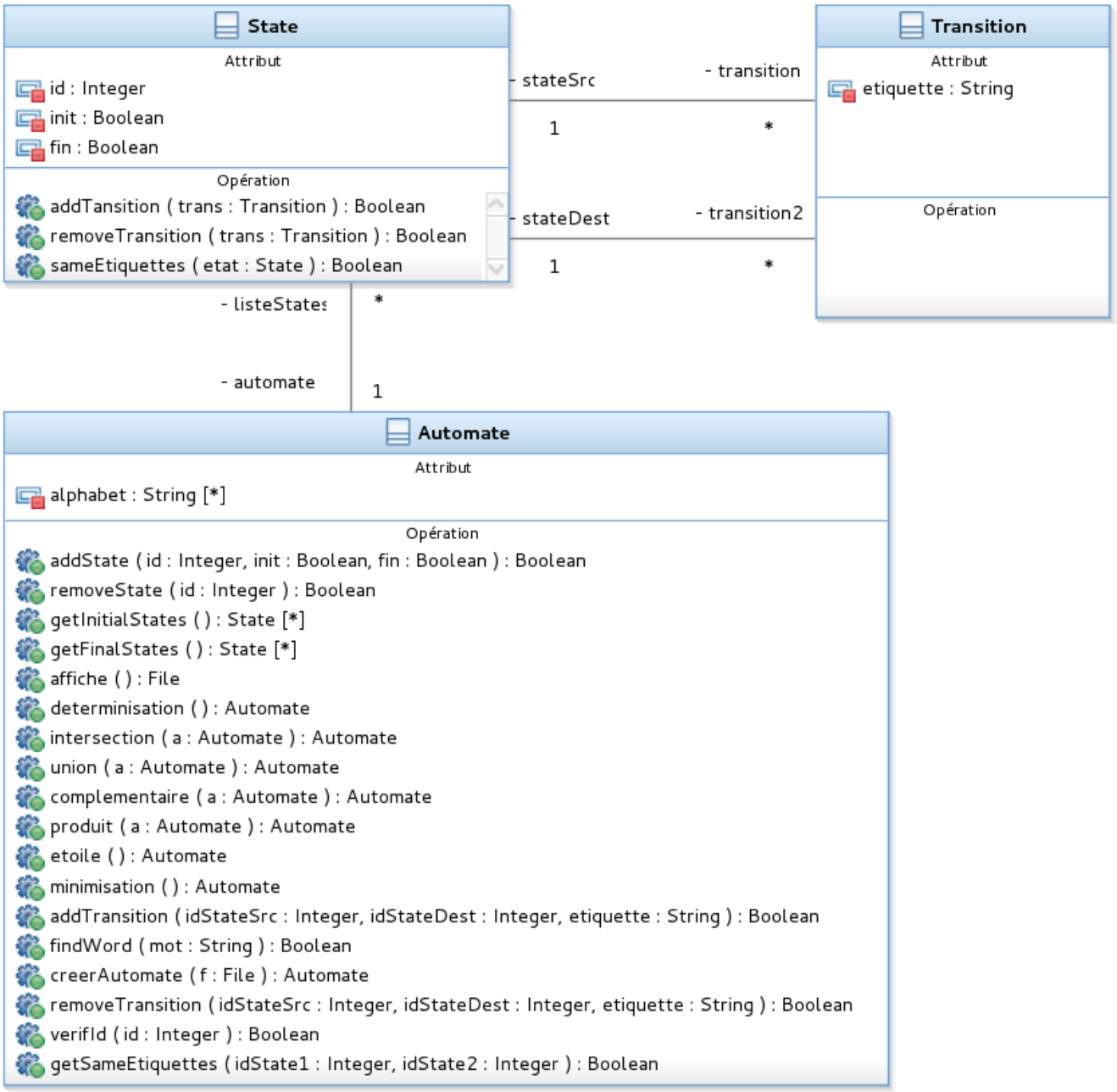
3. Contexte

Notre projet rentre dans le cadre de l'UE « Structures discrètes »(2I005) proposée en L2 et qui est dirigée par Béatrice Bérard. Afin d'aider les étudiants à mieux comprendre la nature informatique des automates, on nous demande de programmer à la fois une structure adaptée à la manipulation des automates finis mais aussi de programmer les opérations sur ceux-ci.

III. Réalisation du projet

Pour réaliser notre projet, nous commencerons par programmer une structure représentant un automate. Nous continuerons avec la programmation d'une interface graphique à l'aide de l'outil « Graphviz ». Puis nous programmerons les opérations, dans l'ordre : déterminisation, intersection, union, complémentaire, produit, étoile et enfin minimisation.
Ensuite nous sélectionnerons certaines parties de code à effacer (celles qui seront compléter par les étudiants lors de leur TME). Enfin nous finirons par rédiger le rapport final.

Pour la structure représentant l'automate, nous proposons :



Nous avons tout d'abord pensé qu'il faudrait trois classes principales : une classe définissant une transition d'abord, une autre définissant un état et finalement une autre définissant un automate :

- Les transitions sont définies par un nom d'étiquette, un état source et un état destination. Nous n'avons pas jugé nécessaire d'ajouter des opérations dans cette classe.

- Les états sont définis par un identifiant (unique dans chaque automate), deux booléens permettant de savoir si c'est un état initial et/ou final et une liste de transitions. Nous avons ajouté les opérations :
  - → addTransition : permettant d'ajouter une transition avec état source ce même état vers un autre état destination ;
  - → removeTransition : permettant de supprimer une transition d'un état source ;
  - → sameEtiquettes : permettant de connaître la liste des étiquettes communes entre deux états. Cette opération sera utile quand on essayera de déterminiser un automate.

- Les automates sont définis par un alphabet (ie. une liste de chaîne de caractères) et une liste d'états. Nous avons décidé d'ajouter les opérations suivantes :
  - → addState : permettant d'ajouter un état à l'automate
  - → removeState : permettant de supprimer un état de l'automate
  - → getInitialStates : permettant de récupérer la liste des états initiaux de l'automate
  - → getFinalStates : permettant de récupérer la liste des états finaux de l'automate
  - → affiche : permettant de générer un fichier qui affichera un automate à l'aide de l'outil graphviz
  - → determinisation : retournant l'automate déterministe
  - → intersection : retournant l'automate qui est l'intersection des deux automates
  - → union : retournant l'automate qui est l'union des deux automates
  - → complementaire : retournant l'automate complémentaire
  - → produit : retournant l'automate qui est le produit des deux automates
  - → etoile : retournant l'automate qui résulte de l'opération étoile
  - → minimisation : retournant l'automate minimisé
  - → addTransition : permettant d'ajouter une transition à un état de l'automate
  - → findWord : permettant de savoir si un mot est accepté par l'automate
  - → creerAutomate : permettant à partir d'un fichier de créer un automate
  - → removeTransition : permettant de supprimer une transition
  - → verifId : permettant de savoir si l'identifiant d'un automate est déjà pris
  - → getSameEtiquettes : permettant de connaître la liste des étiquettes communes entre deux états. Cette opération sera utile quand on essayera de déterminiser un automate.

IV. Conditions de validation

Notre projet sera validé si les opérations que nous aurons programmé répondent aux résultats attendus. Donc si nous programmons par exemple l'opération « déterminisation », on s'attend à ce que l'automate passé en paramètre soit déterministe à la fin. Il en est de même pour les autres opérations. Un des tests que nous pourrons faire après chaque opération ayant pour résultat un automate (donc determinisation, intersection, union, complementaire, produit, etoile et minimisation) est de vérifier qu'un mot accepté par l'automate initial, sera aussi accepté par l'automate final, retourné par l'opération.

En outre, nous prévoyons un jeu de test. On disposera d'un fichier qui aura la définition d'un automate ayant par exemple l'alphabet [« a », « b », « c »], 3 états dont un initial et un final. On commencera donc par créer cet automate grâce à la commande creerAutomate. On se propose ensuite de supprimer l'état qui n'est ni initial ni final à l'aide de la commande removeState. Puis on testera l'affichage avec la commande affiche qui générera le fichier monFic.dot . Pour visualiser ensuite le graphe, il faudra entrer la commande :

dot -Tpn monFic.dot -o monGraph.png

On s'attend donc à voir deux cercles distincts représentant les états et la disparition de toutes les transitions provenant ou à destination de l'état supprimé. On pourra après tester l'ajout d'un état à l'aide de la commande addState (qui sera identique à celui supprimé). Puis on pourra ajouter une transition à l'aide de l'opération addTransition à partir de l'état que l'on vient de créer en direction de l'état initial avec l'étiquette « d ». On s'attend à ce qu'il y est une erreur puisque l'étiquette « d » n'est pas censée être reconnue par l'automate. On réitère l'opération avec l'étiquette « a ». Puis on testera la suppression de cette transition à l'aide de l'opération removeTransition. On pourra ensuite demander à obtenir la liste des états initiaux à l'aide de getInitialStates et la liste des états finaux à l'aide de getFinalStates. On espère avoir comme résultat une liste contenant un état correspondant à notre seul état initial pour getInitialStates et une liste contenant un état correspondant à notre seul état final pour getFinalStates. Enfin on pourra tester si par exemple le mot « abc » est accepté par l'automate à l'aide de la fonction findWord.

Dans une seconde partie, on pourra essayer de déterminiser notre automate à l'aide de l'opération déterminisation. En résultat, on aura l'automate déterministe. Puis on pourra demander l'automate complémentaire de notre automate grâce à l'opération complementaire. On pourra aussi appliquer l'étoile sur notre automate avec l'opération etoile.

Enfin, pour tester nos dernières opérations, on créera un nouvel automate a2 avec l'opération creerAutomate (et en passant donc en paramètre un fichier décrivant notre automate a2). On testera ensuite l'intersection entre notre premier automate et le deuxième a2 avec l'opération intersection. On aura en retour un automate résultant de l'intersection de ces deux ci. Puis nous testerons l'union entre ces deux automates grâce à l'opération union, on s'attend donc en résultat à avoir un automate résultant de l'union des deux premiers. Enfin, nous finirons par tester le produit entre ces deux automates à l'aide de l'opération produit. Nous pourrons utiliser, comme pour les autres opérations aussi l'affichage à l'aide de l'opération affiche pour visualiser que l'automate obtenu résulte bien du produit entre les deux premiers automates.

V. Planning prévisionnel

Nous nous accorderons 3 semaines pour déterminer la structure de l'automate et programmer l'interface graphique à l'aide de l'outil « Graphviz ». Puis nous nous accorderons 5 semaines pour programmer les différentes opérations ainsi que la structure d'un automate à l'aide d'un éditeur de texte du type « Emacs ». Et enfin le reste du temps sera consacré à la rédaction du rapport.