

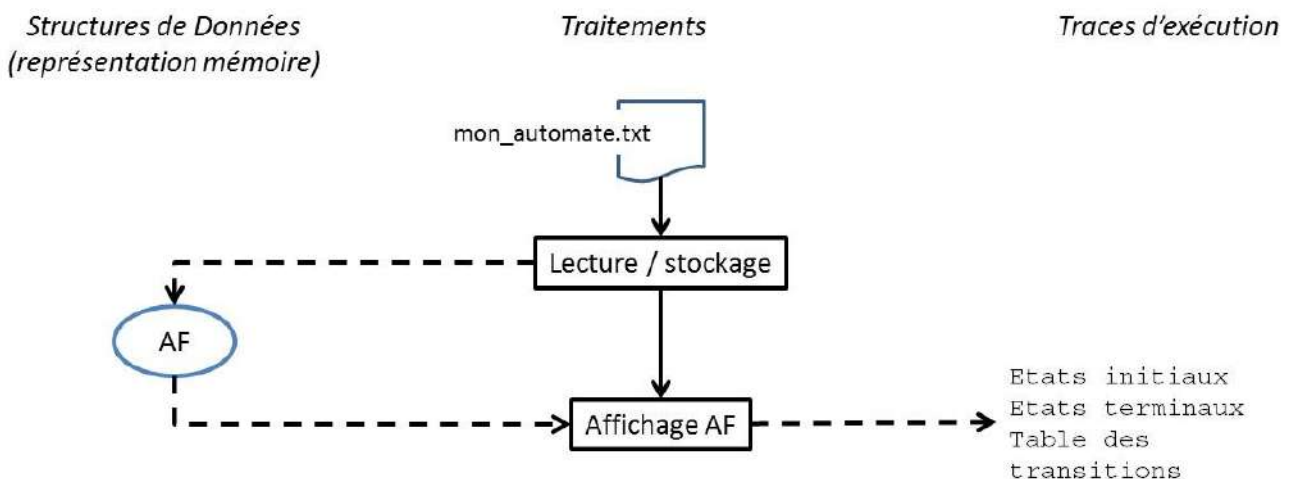
Traitement d'automate fini **Reconnaissance de mot**

Prenez le temps de lire attentivement ce document. Il contient notamment certaines instructions que vous devez absolument respecter. Tout manquement influencera fatalement votre note.

D'autres éléments d'informations (clarification, complément) pourront vous être fournis ultérieurement.

Programme à développer

Votre programme doit dans un premier temps lire la description d'un automate dans un fichier texte, et le sauvegarder en mémoire.



On retrouve, sur le schéma précédent comme sur les suivants, les traitements à effectuer, avec les structures de données à créer et manipuler (sur la gauche) ainsi que les traces d'exécution à produire (sur la droite).

Lecture / stockage	Lire un automate contenu dans un fichier et le stocker en mémoire (« AF »).
--------------------	-----------------------------------------------------------------------------

Voir plus loin les automates à prendre en compte, ainsi que la structure du fichier de données.

Le choix du fichier à lire se fait au travers de l'interface utilisateur (pendant l'exécution de votre programme).

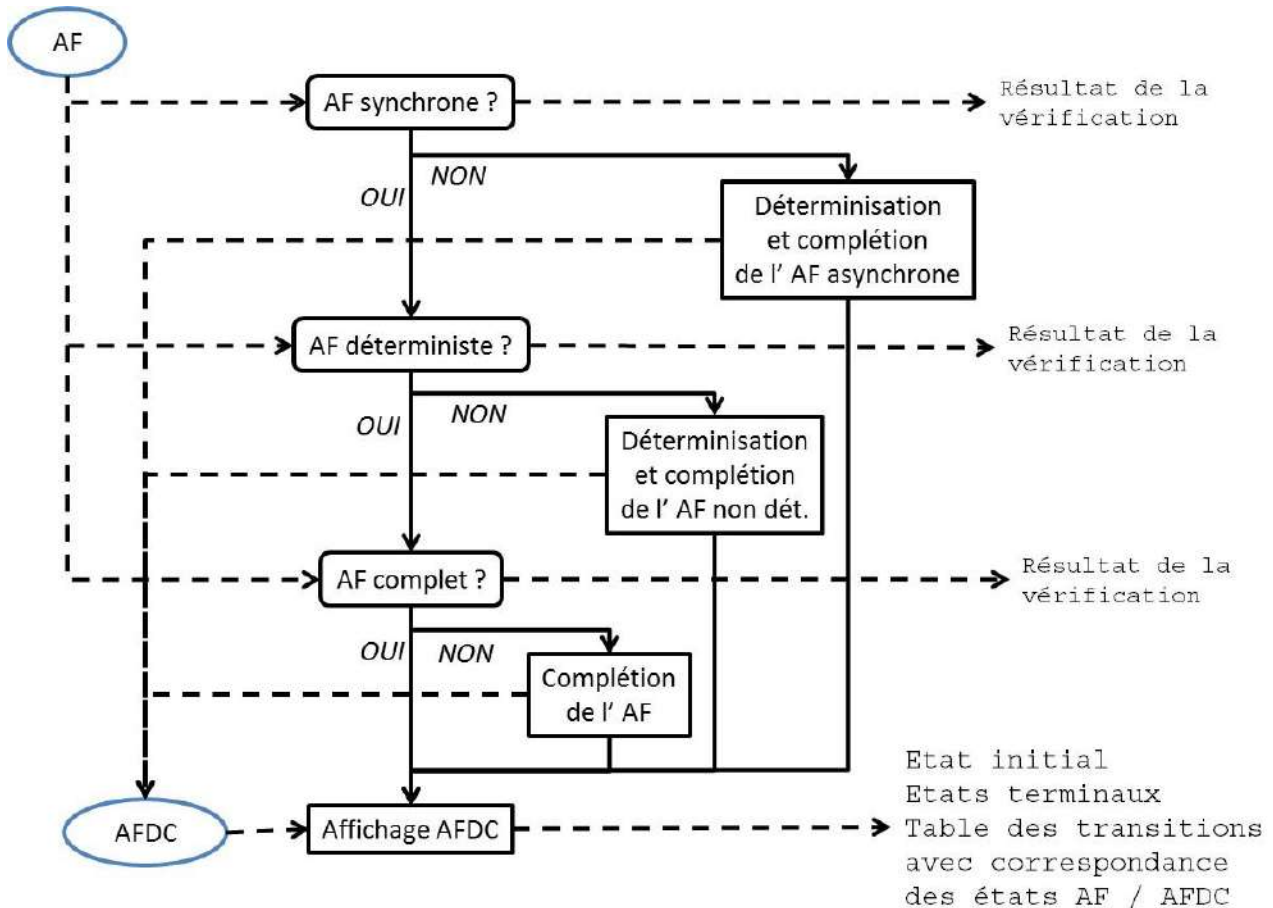
La saisie clavier de l'automate (sans fichier) est exclue, de même que la définition « en dur » dans le programme.

Affichage AF

Affichage de l'automate sauvegardé en mémoire, en indiquant explicitement :

- l'état initial, ou les états initiaux ;
- les états terminaux ;
- la table des transitions.

Une fois l'automate initial stocké en mémoire, vous devez effectuer les traitements de détermination et complétion, comme décrit dans le diagramme suivant :



AF synchrone ?

Vérifier s'il s'agit d'un automate asynchrone ou pas.

Le résultat du test est affiché. Si l'automate est asynchrone, votre programme doit également afficher les éléments qui font que l'automate est asynchrone.

Détermination et complétion de l'AF asynchrone

Construction d'un automate synchrone, déterministe et complet à partir de l'automate initial asynchrone (AF).

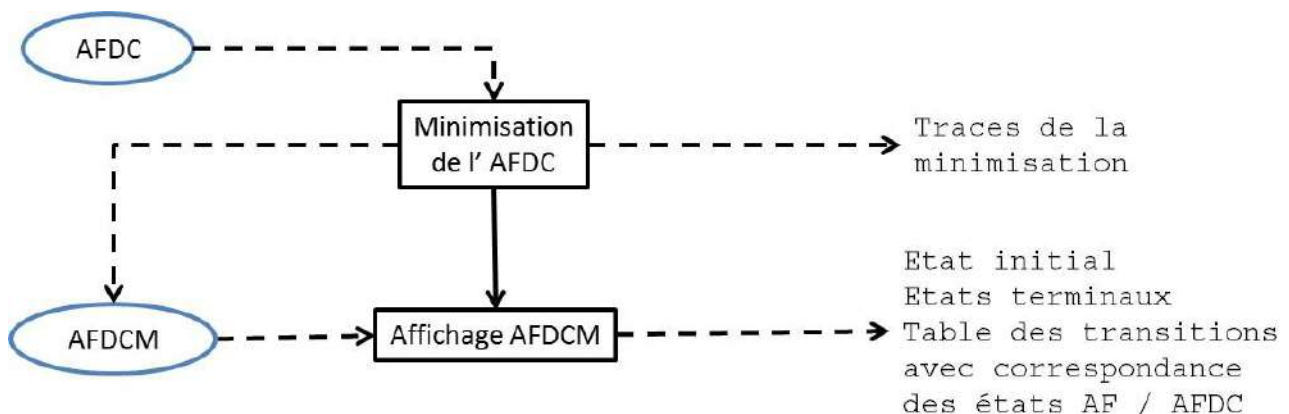
AF déterministe ?

Vérifier si l'automate AF synchrone est déterministe ou non.

Le résultat du test est affiché. Si l'automate est non déterministe, votre programme doit en afficher les raisons.

Déterminisation et complétion de l'AF non dét.	Construction de l'automate déterministe et complet (AFDC) à partir de l'automate synchrone non déterministe (AF).
AF complet ?	Vérifier si l'automate synchrone et déterministe (AF) est complet. Le résultat du test est affiché. Si l'automate n'est pas complet, votre programme doit en afficher les raisons.
Complétion de l'AF	Construction de l'automate déterministe et complet (AFDC) à partir de l'automate synchrone et déterministe (AF).
Affichage AFDC	Affichage de l'automate, sous un format équivalent à ce qui a été utilisé pour l'automate initial. En plus de l'affichage de l'automate déterministe complet (AFDC), votre programme doit explicitement indiquer à quels états de l'automate non déterministe en entrée du traitement (AF) correspond chacun des états de l'automate déterministe complet (AFDC). Note : Il est préférable de garder dans la notation des états composés de l'AFDC l'ensemble des états correspondant de l'automate asynchrone AFA, ou bien effectuer la déterminisation en termes des ϵ -clôtures de certains états de l'automate asynchrone, en affichant la liste des ϵ -clôtures en tant qu'états composés.

Traitement suivant : minimisation de l'automate obtenu :



Minimisation de l'AFDCM	Construction de l'automate synchrone, déterministe, complet et minimal (AFDCM) à partir de l'automate synchrone, déterministe et complet (AFDC).
-------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------

On notera qu’il n’existe pas à ce stade de « vérification » préalable.

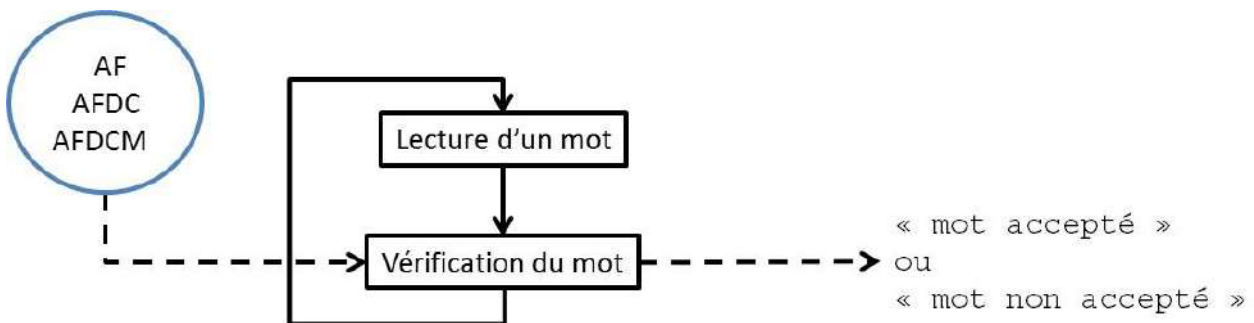
Votre programme doit afficher les partitions successives, ainsi que les transitions exprimées en termes de parties, tout au long du processus de minimisation.

Affichage AFDCM

Affichage de l’automate sous une forme similaire aux précédentes.

Votre programme doit notamment afficher de façon explicite à quels états de l’automate déterministe complet (AFDC) chaque états de l’automate minimal (AFDCM) correspond. Cette correspondance peut soit être directement visible dans la table de transition, soit effectuée au moyen d’une table de correspondance distincte (en cas de renommage des états dans l’AFDCM).

Enfin, votre programme procède à l’analyse de mots fournis au clavier par l’utilisateur.



Lecture d’un mot

Récupération d’une chaîne de caractères donnée par l’utilisateur au clavier de l’ordinateur.

Attention :

- Vous devez absolument prévoir dans votre programme un moyen pour l’utilisateur de saisir le mot vide.
- Le mot est lu en entier sur une ligne avant vérification. Il ne doit pas y avoir une lecture / vérification caractère par caractère.

Vérification du mot

Utilisation de l’automate (AF, AFDC ou AFDCM) pour vérifier si un mot appartient ou non au langage.

En résultat : « oui » ou « non ».

Option : indiquer le premier caractère du mot où une erreur est rencontrée.

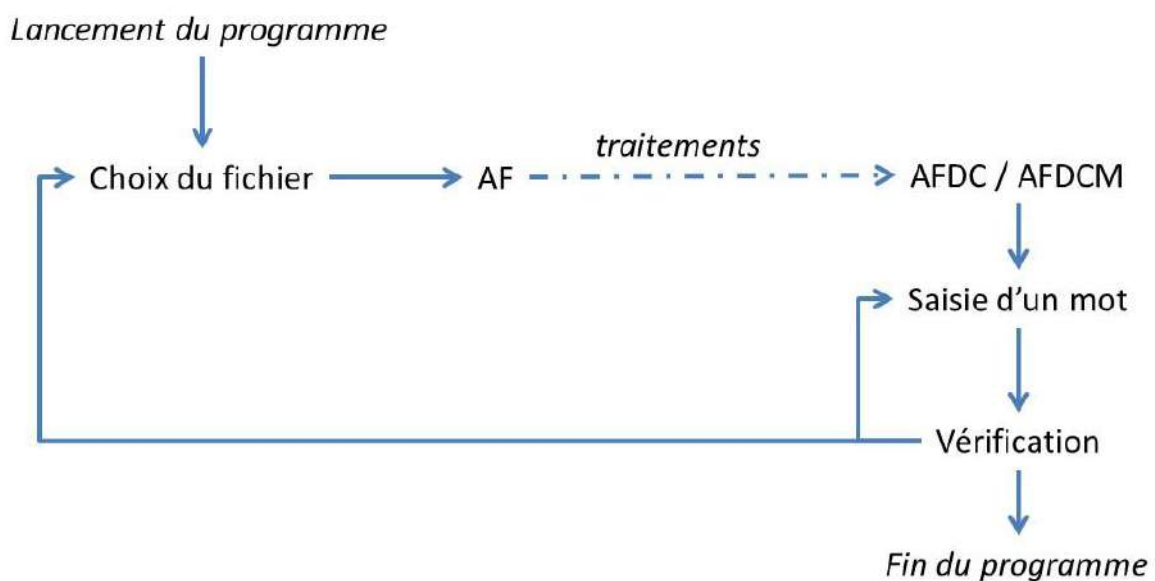
Vous êtes libre de mettre en œuvre la vérification avec n’importe quel automate.

C'est plus facile avec un automate déterministe qu'avec un automate non déterministe.

Si votre code utilise un automate déterministe complet, il pourra utiliser indifféremment un automate minimal ou non.

Attention : votre programme doit IMPÉRATIVEMENT prévoir une boucle permettant de tester plusieurs mots sans être obligé de redémarrer votre programme.

De même, il est fortement conseillé de mettre tous les traitements identifiés ci-dessous dans une boucle générale permettant de traiter plusieurs automates AF sans relancer votre programme.



L'interface utilisateur doit être assez simple et guidée par des messages écran de façon à ce que l'enseignant puisse ne pas vous demander comment faire ceci ou cela. Le choix de la représentation mémoire pour un automate est libre.

Partie bonus

Programmer l'obtention du langage reconnu par un automate fini (déterministe ou non) en utilisant la méthode d'élimination d'états. Le résultat : une chaîne de caractères représentant une expression rationnelle.

Toutes les étapes intermédiaires (l'état à éliminer, l'automate après l'élimination d'un état...) doivent être affichées sur l'écran et figurer dans la trace d'exécution.

Environnement de programmation

Vous pouvez utiliser les langages C ou C++.

Votre programme doit pouvoir être compilé et s'exécuter sous Unix/Linux et Windows. Evitez des instructions n'appartenant pas au langage standard (non reconnu par le

compilateur gcc). A vous de vérifier sur les machines de l'école, et de n'utiliser que des mécanismes suffisamment standards. Au cas de besoin, l'enseignant doit pouvoir compiler et exécuter chez lui, ce qui veut dire que tout programme doit marcher en utilisant CodeBlocks standard, ou VS Express 2012.

Automates à prendre en compte

Les tests seront effectués sur des automates :

- ayant pour alphabet des lettres de 'a' à 'z' :
par exemple, un automate dont l'alphabet contient 3 symboles utilisera les lettres 'a', 'b' et 'c' ;
- dont les états sont numérotés à partir de '0' :
par exemple, un automate contenant 5 états contiendra les états numérotés de 0 à 4, sans rupture de séquence.

Il n'y a pas de limite concernant le nombre d'états que les automates de test pourront contenir.

Le fichier de données représentant l'automate lu par votre programme doit avoir la syntaxe suivante :

Ligne 1 : nombre de symboles dans l'alphabet de l'automate.

Ligne 2 : nombre d'états.

Ligne 3 : nombre d'états initiaux, suivi de leurs numéros.

Ligne 4 : nombre d'états terminaux, suivi de leurs numéros.

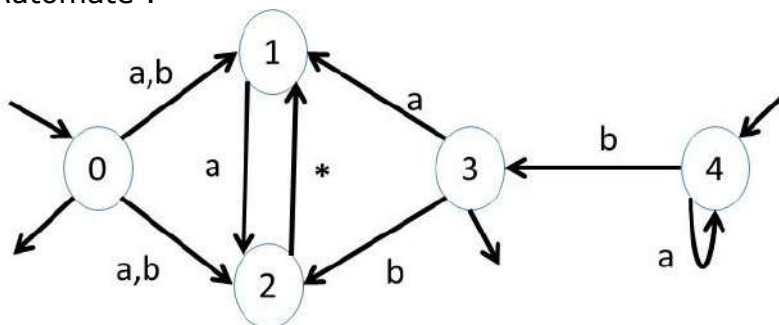
Ligne 5 : nombre de transitions.

Lignes 6 et suivantes : transitions sous la forme
<état de départ><symbole><état d'arrivée>

En cas d'automate asynchrone, une transition « epsilon » sera représentée par le symbole « * ».

Par exemple :

Automate :



Fichier :

```

2
5
2 0 4
2 0 3
10
0a1
0a2
0b1
0b2
1a2
2*1
3a1
3b2
4b3
4a4
    
```

2 symboles dans l'alphabet

$A=\{a,b\}$

5 états

$Q=\{0,1,2,3,4\}$

2 états initiaux

$I=\{0,4\}$

2 états terminaux

$T=\{0,3\}$

10 transitions (dont une 'epsilon')

Déroulement du TAI

Constitution des équipes

Par équipes de 5 étudiants. Lorsque le nombre d'étudiants d'un groupe TD l'impose, il y aura des équipes de 4 étudiants.

Date limite de constitution des équipes : vendredi 08/04.

La constitution des équipes devra être envoyée par le délégué à aux adresses suivante :

Pour les groupes de M Velikson : velikson.efrei@gmail.com

Pour les groupes de M Barbot : herve.barbot@efrei.fr

Dans le cas où toutes les équipes ne seraient pas constituées à cette date, il est demandé à chaque délégué de communiquer la liste des équipes en sa possession. Si toutes les équipes ne sont pas constituées et communiquées à l'enseignant à la date indiquée ci-dessus, celui-ci se chargera de constituer lui-même les équipes, sans discussion possible de la part des étudiants.

La constitution de toutes les équipes sera diffusée sur Campus. Aucun changement ne sera accepté au-delà de cette date, sauf cas de force majeure qui devra être explicitement accepté par l'enseignant.

Tests (préparation à la soutenance)

Les automates de test qui seront utilisés pour la soutenance vous seront communiqués ultérieurement.

Remise de votre travail

Contenu de l'envoi par email : votre programme (code source uniquement, bien commenté) et vos fichiers de test.

Date limite : sera fixée ultérieurement.

Adresse email : voir section concernant la constitution des groupes.

Les noms des fichiers doivent impérativement être préfixés par les noms des membres de chaque équipe par ordre alphabétique, par exemple :

Dupond-Dupont-Durand-monprogramme.cpp (ou .c)

Dupond-Dupont-Durand-monprogramme.h

Dupond-Dupont-Durand-test1.txt

Conseil: nommez correctement vos fichiers dès le début de votre travail !

Pas de fichier archive autre qu'au format zip. Vous pouvez envoyer autant de pièces jointes que nécessaire.

Pas de fichier .layout, .cbp, .exe etc.

Soutenance

Calendrier : sera fixée ultérieurement.

Durée : 50 minutes par équipe.

Date limite d'inscription des équipes dans les créneaux ouverts : sera communiquée ultérieurement.

Il n'y aura pas d'ordinateur mis à votre disposition. Vous devez impérativement venir avec le vôtre.

Déroulement : exposé, démonstration, discussion complémentaire éventuelle.

Exposé

Durée de l'exposé : 20 minutes.

Chaque membre de l'équipe présentera une partie de l'exposé, sans intervention des autres. Sachez à l'avance qui présente quoi !

Les temps alloués à chaque membre de l'équipe, ainsi que la difficulté des traitements présentés, doivent être équilibrés entre les membres de l'équipe. A vous de vous en assurer.

Vous devez impérativement préparer un support écrit (à remettre sur papier en début de soutenance) pour votre exposé.

Ce support doit clairement présenter vos structures de données et vos algorithmes.

*Conseil : préférez un schéma clair et précis ou du pseudo-code bien structuré que vous commenterez, plutôt qu'un texte long qui ne pourra pas être lu durant la soutenance ! Votre structure de données et vos algorithmes doivent en sortir de façon absolument claire, et vous n'utiliserez pas directement le code C ou C++ pour la présentation. Soyez complet et précis : dire par exemple qu'un automate est représenté par une classe d'objet n'est pas suffisant si on ne connaît pas plus précisément les structures de données utilisées et les fonctions permettant leur manipulation ; dire que c'est un tableau n'est pas suffisant non plus si on ne sait pas ce que les cases du tableau contiennent... **Expliquer une structure de données veut dire faire comprendre sans qu'on doive poser des questions supplémentaires comment l'automate est représenté dans la mémoire de la machine.** Il faut aussi préciser spontanément, sans attendre qu'on vous le demande, si vous utilisez une même structure de données pour des automates non déterministe et déterministes ou ce sont deux structures de données différentes.*

Attention :

Si vous voulez utiliser une présentation formelle de type « powerpoint », vous devrez vous équiper d'un vidéoprojecteur. Ceci n'est cependant pas conseillé dans la mesure où sa mise en place prend du temps. Un PC portable avec un écran lisible est généralement suffisant.

Démonstration

Vous aurez préalablement placé dans votre environnement de travail tous les fichiers correspondant aux automates de test fournis.

Vous compilerez votre programme, puis vous lancerez son exécution. L'examineur vous indiquera le ou les fichiers à lire ou fournira un autre automate, et les chaînes à vérifier. Il pourra également demander la modification d'un automate existant, voire la saisie d'un nouvel automate.

Pour effectuer des tests complémentaires, des modifications des fichiers d'entrée pourront être demandées en séance.

Assurez-vous que votre ordinateur est opérationnel (batterie chargée, système lancé, ...) avant d'entrer dans la salle de soutenance !

Discussion / Q&R

L'enseignant pourra bien entendu vous poser des questions à tout moment. Il pourra demander à une personne en particulier de lui répondre, sans aide des autres. Bien que vous ne travaillerez pas tous directement sur tous les composants de votre programme (ce qui est tout à fait normal pour un travail de groupe), chacun d'entre vous doit donc être capable de répondre à des questions relativement générales sur l'ensemble du TAI. Nous vous conseillons donc vivement de vous réunir régulièrement, et vous présenter les uns aux autres le travail effectué.

Outre le fait que cela donnera à chacun les informations nécessaires pour la soutenance, cela vous forcera à apprendre comment mettre en œuvre les mécanismes principaux que vous avez appris en cours et TD.

Éléments de notation

Vous serez bien entendu jugés sur le fonctionnement de votre programme ainsi que sur la qualité de votre exposé.

Sachez que les éléments suivants seront également pris en compte :

- clarté de l'interface de votre programme permettant un bon déroulement et un suivi simple de la démonstration (enchaînement des actions, trace d'exécution, ...)
- facilité de vérification du bon fonctionnement de votre programme ;
- choix / justification des structures de données pour représenter les automates ;
- lisibilité du code (cela inclut les commentaires).

Une note globale sera donnée au groupe, mais sera modulée de quelques points en fonction de la prestation orale de chacun.