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 se déroule en plusieurs étapes :

- lecture d'un automate (dans un fichier) ;
- si l'automate n'est pas déterministe complet, obtention de l'automate déterministe et complet correspondant ;
- minimisation de l'automate ;
- reconnaissance de mots ;
- création d'un automate reconnaissant le langage complémentaire et test de reconnaissance de mots ce langage-ci ;
- standardisation de l'automate complémentaire et reconnaissance de mots.

Lecture d'un automate dans un fichier

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

Pseudo_code :

```
AF←lire_automate_sur_fichier(nom_fichier)
afficher_automate(AF)
```

Voir plus loin les automates à prendre en compte, ainsi qu'une description de la structure du fichier de données contenant un automate.
Pendant l'exécution de votre programme, le choix du fichier à lire se fait au travers de l'interface utilisateur.).

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

La représentation d'un automate en mémoire va dépendre de votre choix de structures de données. Vous êtes libre de choisir les structures de données qui vous semblent plus adaptées.

`afficher_automate`

Affichage de l'automate sauvegardé en mémoire, en indiquant explicitement :
- l'état initial, ou les états initiaux ;
- les états terminaux ;

- la table des transitions.

Déterminisation et complétion

Soit l'automate « AF » obtenu (et affiché) à l'étape précédente. Le traitement se déroule selon le pseudo-code suivant :

```
SI est_un_automate_asynchrone(AF)
ALORS
      AFDC←determinisation_et_completion_automate_asynchrone(AF)
SINON
      SI est_un_automate_deterministe(AF)
      ALORS
            SI est_un_automate_complet(AF)
            ALORS
                  AFDC←AF
            SINON
                  AFDC←completion(AF)
            FINSI
      SINON
            AFDC←determinisation_et_completion_automate_synchrone(AF)
      FINSI
FINSI
afficher_automate_deterministe_complet(AFDC)
```

`est_un_automate_asynchrone ( AF )`
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.

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

`est_un_automate_deterministe ( AF )`
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.

`est_un_automate_complet ( AF )`
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.

`completion ( AF )`
Construction de l'automate déterministe et complet à partir de l'automate synchrone et déterministe AF.

`determinisation_et_completion_automate_synchrone ( AF )`
Construction de l'automate déterministe et complet à partir de l'automate synchrone non déterministe (AF).

Il s'agit ici de mettre en œuvre le processus de déterminisation vu en cours et TD. Cette étape ne doit en aucun cas être précédée de la standardisation de l'automate.

`afficher_automate_deterministe_complet(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 AF, du genre « 123 » pour un état composé de 1, de 2 et de 3. Si les numéros d'états dépassent 9, attention de faire la différence entre {1,2,3} et {12,3}, que l'on notera respectivement, par exemple, « 1.2.3 » et « 12.3 » avec un séparateur de votre choix.

Attention :
Les différents tests mentionnés dans le pseudo-code sont impératifs. Par exemple, la déterminisation d'un automate ne peut être lancée que si l'automate a été explicitement identifié comme n'étant pas déjà déterministe.

Minimisation

Il n'y a pas de test à effectuer. On minimise l'automate déterministe complet obtenu précédemment et on affiche le résultat. S'il s'avère que l'automate à minimiser était déjà minimal, votre programme doit afficher un message correspondant.

Pseudo-code :

```
AFDCM← minimisation ( AFDC )
afficher_automate_minimal ( AFDCM )
```

`minimisation ( AFDC )`

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. Faites attention à ce que cet affichage soit facilement lisible.

`afficher_automate_minimal ( 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 état 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).

Reconnaissance de mots

Votre programme procède à l'analyse de mots fournis au clavier par l'utilisateur.
Votre programme doit impérativement permettre de saisir plusieurs mots et lancer la reconnaissance sur chacun d'entre eux.

Pseudo-code

```
lire_mot ( mot )
TANT QUE mot ≠ « fin » FAIRE
      reconnaitre_mot ( mot , A )
      lire_mot ( mot )
REFAIRE
```

`lire_mot ( 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.
-A vous de déterminer le mot correspondant à « fin de lecture des mots ». Le pseudo-code propose la chaîne « fin », mais vous pouvez choisir ce que vous voulez.

`reconnaitre_mot ( mot , A )`

Utilisation de l'automate A pour vérifier si un mot appartient ou non au langage.
En résultat : « oui » ou « non ».
Option : indiquer le premier caractère du mot qui l'empêche d'être reconnu.
Vous êtes libre de mettre en œuvre la vérification avec n'importe quel automate (AF, AFDC, AFDCM). C'est plus facile avec un automate déterministe qu'avec un automate non déterministe.
Si, pour le test de reconnaissance, votre code utilise un automate déterministe complet, il pourra utiliser indifféremment un automate minimal ou non.

Langage complémentaire

Votre programme doit finalement construire l'automate reconnaissant le langage complémentaire et tester la reconnaissance des mots par cet automate.

Pseudo-code :

```
AComp ← automate_complementaire ( A )
afficher_automate ( AComp )
lire_mot ( mot )
TANT QUE mot ≠ « fin » FAIRE
      reconnaitre_mot ( mot , AComp )
      lire_mot ( mot )
REFAIRE
```

`automate_complementaire ( A )`

Construction d'un automate reconnaissant le langage complémentaire.
Le paramètre d'entrée A peut être, à votre choix, l'AFDC ou l'AFDCM obtenu précédemment.

Standardisation

Votre programme doit finalement rendre standard l'automate obtenu à l'étape précedente, puis exécuter une boucle de reconnaissance de mots.

Pseudo-code :

```
ACompStd ← automate_standard ( A )
afficher_automate ( ACompStd )
lire_mot ( mot )
TANT QUE mot ≠ « fin » FAIRE
      reconnaitre_mot ( mot , ACompStd )
      lire_mot ( mot )
REFAIRE
```

`automate_standard ( mot , A )`

L'automate précédent 'A' est standardisé.

Enchaînement des traitements

Si l'intégralité du code présenté ci-dessus est réalisé, on obtient successivement les automates suivant :
AF → AFDC → AFDCM → AComp → ACompStd
Dans le cas où certaines étapes ne seraient pas mises en œuvre, vous pouvez bien entendu avoir des séquences différentes. Par exemple :

Minimisation non effectuée :

AF → AFDC → AComp → ACompStd

Minimisation et passage au langage complémentaire non effectués :

AF → AFDC → AStd

L'automate standard est alors obtenu à partir de l'automate déterministe complet

Vous devrez bien évidemment indiquer clairement ce que votre programme exécute lors de votre soutenance.

Boucle de traitement de plusieurs automates

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.

Environnement de programmation

Vous pouvez utiliser les langages C ou C++.

Votre programme devra pouvoir être compilé et exécuté dans l'environnement de test de votre enseignant. Celui-ci vous indiquera les outils et numéros de versions dont il dispose.

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.

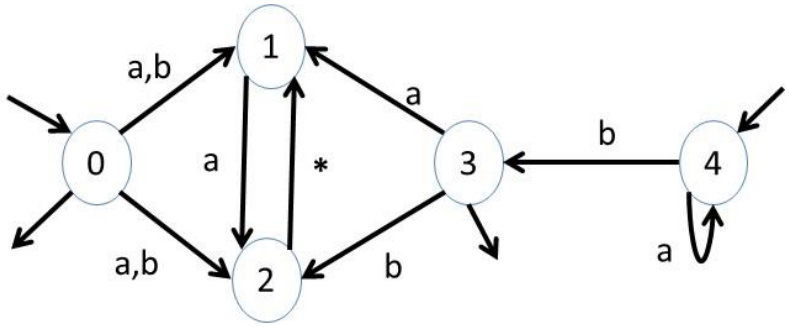
Le fichier de données représentant l'automate lu par votre programme peut avoir la syntaxe suivante (mais vous pouvez opter pour une syntaxe un peu différente) :

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 » peut être représentée par le symbole « * ».

Par exemple :

Automate :



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')

Fichier :

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

Déroulement du TAI

Constitution des équipes

Votre enseignant vous indiquera les règles à suivre pour la constitution des équipes.

Tests (préparation à la soutenance)

Les automates de test qui seront utilisés pour la soutenance vous seront communiqués ultérieurement.
Vous devrez impérativement vous assurer que tous les fichiers de données correspondant à tous ces automates de tests soient disponibles sur l'ordinateur que vous utiliserez lors de votre soutenance, ainsi que celui-ci marche bien et que votre code peut bien être compilé sur cet ordinateur.

Remise de votre travail

Conditions et contenu de l'envoi : votre enseignant vous en fixera ultérieurement les modalités.
Date limite : sera fixée ultérieurement. Elle sera la même pour tous les groupes, quelles que soient les dates de soutenance.

Soutenance

Calendrier : sera fixée ultérieurement.
Durée : 45 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 en vous assurant préalablement que celui-ci marche et compile.

Déroulement : exposé, démonstration, discussion complémentaire éventuelle.

Exposé

Durée de l'exposé : 15 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 connait 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 donnés pour des automates non déterministes et déterministes ou ce sont deux structures de données différentes.*

Attention :
Il est fortement conseillé de préparer une présentation de type « powerpoint ». Les salles sont généralement équipées en vidéoprojecteur, ou en écran télévision. Si ce n'est pas le cas, un ordinateur portable avec un écran lisible sera 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 (qui doit être exactement celui que vous avez envoyé, sans corrections ou ajouts de dernière minute) en présence de l'enseignant, puis vous lancerez son exécution. L'examinateur vous indiquera le ou les fichiers à lire, et les chaînes de caractères 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.

Eléments 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. L'absence des commentaires est habituellement perçue, sauf exceptions rares, comme un code difficilement lisible).

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