Traitement d'automate fini

PROGRAMME A DEVELOPPER

(pour obtenir au moins 10 pour votre projet, vous devez réussir à programmer les points de 1 à 3 !)

Votre programme se déroule en plusieurs étapes :

1. lecture d'un automate que vous coderez préalablement dans un fichier, puis mise en mémoire et affichage sur l'écran ;
2. affichage des informations : l'automate est-il déterministe ? standard ? complet ?
3. si l'automate n'est pas standard, standardisation à la demande de l'utilisateur
4. si l'automate n'est pas déterministe complet, obtention de l'automate déterministe et complet équivalent

bonus :

- calcul de l'automate minimal équivalent ;
- test de reconnaissance de mots ;
- création d'un automate reconnaissant le langage complémentaire

Lecture d'un automate dans un fichier

Le 15 mars nous vous donnerons des automates de test. Ceux-ci devront être transcrits par vos soins dans des fichiers « .txt ». Ces fichiers devront être nommés de façon suivante :

<numéro d'équipe>-#.txt

où le # sera remplacé par le numéro de l'automate de test. Par exemple, si vous êtes l'équipe B10 et qu'il s'agisse de l'automate de test numéro 8, le fichier doit être nommé B10-8.txt.

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

Le choix de l'automate sera effectué par l'utilisateur pendant l'exécution de votre programme, et ce choix doit pouvoir se faire de façon très simple : par exemple, pour choisir l'automate de test n° 8, il doit suffire de taper « 8 » en réponse à la question « quel automate voulez-vous utiliser ?».

Tant que vous n'avez pas encore reçu les automates de test (qui vous seront fournis sous forme graphique !), vous êtes libres de travailler sur n'importe quel automate de votre invention, en le codant dans un fichier .txt selon le mode que vous verrez plus loin dans ce document.

Affichage de l'automate

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

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

Par exemple :

		a	b	c
E	0	0, 1	0, 4	0
	1	--	--	2
	2	--	3	--
S	3	3	--	--
S	4	--	--	2

(Attention, la lisibilité de la table est capitale : les colonnes doivent être bien alignées, indépendamment de la longueur du contenu de la cellule ; les lignes verticales et horizontales séparant les cellules sont optionnelles)

Les étapes de lecture et d'affichage peuvent être schématisées par le pseudo-code suivant :

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

Standardisation si demandé

```
SI non_standard(AF)
Alors
     SI l'utilisateur veut le standardiser
          SFA ← standardisation(AF)
```

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_deterministe(AF)
ALORS
     SI est_un_automate_complet(AF)
     ALORS
          AFDC←AF
     SINON
          AFDC←completion(AF)
     FINSI
SINON
     AFDC←determinisation_et_completion_automate(AF)
FINSI
```

```
FINSI
afficher_automate_deterministe_complet(AFDC)
```

où :

`est_un_automate_deterministe(AF)`

Vérifier si l'automate AF 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 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 non complet AF.

`determinisation_et_completion_automate(AF)`

Construction de l'automate déterministe et complet à partir de l'automate non déterministe AF.

Il s'agit ici de mettre en œuvre le processus de déterminisation vu en cours et TD.

`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).
Les états de l'automate déterminisé doivent être notés de telle façon que leur composition en états de l'automate d'origine soit claire : par exemple, « 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 doit être lancée que si l'automate a été explicitement identifié comme n'étant pas déjà déterministe. La déterminisation d'un automate qui l'est déjà sera considéré comme une erreur.

Option Bonus : 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)
```

où :

`minimisation ( AFDC )`

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

Votre programme doit afficher les partitions successives (numérotées !), 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.

Il se peut qu'il ne soit pas pratique de nommer les états de l'automate minimal en évoquant les états de l'automate que vous minimisez : par exemple, si « 2.3 » et « 0.4.5 » sont les noms d'états d'un AFDC, il ne serait pas facile de lire le résultat si, par exemple, un des états est nommé « 2.3, 0.4.5, 1 ». Donc vous avez le droit de renommer les états de l'automate minimal. Mais alors, vous devez impérativement afficher une table de correspondance des états de l'automate que vous avez minimisé et l'automate minimal.

Option Bonus : 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
```

où :

`lire_mot ( mot )`

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

Attention :

-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 déterministe complet A pour vérifier si un mot appartient ou non au langage.

En résultat : « oui » ou « non ».

Option bonus : Langage complémentaire

En bonus, votre programme peut construire l'automate reconnaissant le langage complémentaire.

Pseudo-code :

```
AComp ← automate_complementaire ( A )
afficher_automate ( AComp )
```

où :

`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. Votre programme doit indiquer à partir de quel automate le complémentaire est obtenu.

Boucle de traitement de plusieurs automates

Il est impératif de mettre tous les traitements identifiés ci-dessus dans une boucle générale permettant de traiter plusieurs automates AF sans relancer votre programme.

ENVIRONNEMENT DE PROGRAMMATION

Vous pouvez utiliser les langages C, C++, Python ou Java.

AUTOMATES A PRENDRE EN COMPTE

Les tests seront effectués sur des automates :

- ayant pour alphabet les premières lettres de l'alphabet latin au nombre indiqué dans le fichier .txt de l'automate, sans rupture de séquence:

  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 des automates de test.

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.

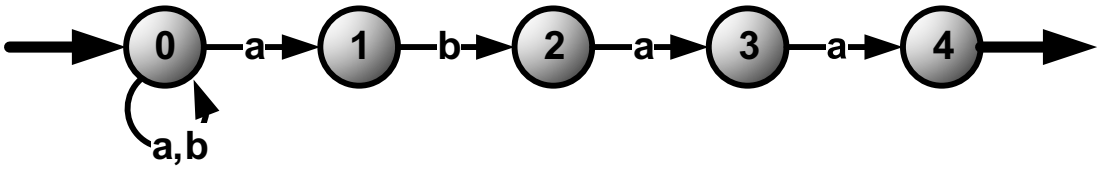
Ligne 5 : nombre de transitions.

Lignes 6 et suivantes : transitions sous la forme

<état de départ><symbole><état d'arrivée>

Par exemple :

Automate :



2 symboles dans l'alphabet
A={a,b}
5 états
Q={0,1,2,3,4}
1 état initial
I={0}
1 état terminal
T={4}
6 transitions

Fichier (par exemple):

```
2
5
1 0
1 4
6
0a0
0b0
0a1
1b2
2a3
3a4
```

DEROULEMENT DU PROJET

Constitution des équipes

En fonction du nombre total d'élèves dans le groupe, vous formez 7 ou 8 équipes de 5 ou de 4. Par exemple, un groupe de 39 élèves aura 7 équipes de 5 et une équipe de 4. On part de l'hypothèse que 5 est le nombre normal d'élèves dans une équipe, et 4 est utilisé si le nombre d'élèves n'est pas un multiple de 5.
Vous devrez communiquer à votre enseignant la liste des équipes **avant le 10 mars (le délégué du groupe doit s'en occuper)**. *Sinon, l'enseignant pourra affecter des élèves aux équipes de façon arbitraire (par exemple, alphabétique), sans que cela puisse être contesté par la suite.*

Tests (préparation à la soutenance)

Les automates de test qui seront utilisés pour la soutenance vous seront communiqués avant **20 mars**.

Vous devrez impérativement vous assurer que tous les fichiers de données correspondant à tous ces automates de test 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

Toutes les équipes devront remettre leur travail sur Moodle pas plus tard que le **26 mars à minuit**.

Le contenu du rendu :

- Code source : Tout fichier code que vous avez tapé vous-même à l'exclusion de tout autre fichier (**donc aucun fichier produit par le logiciel durant la compilation ou exécution**), bien commenté.
- Tous les fichiers .txt des automates de test (oui, malgré le fait que ce sont vos enseignants qui vous ont fourni les automates de test), **dans le même répertoire que le code**.
- Un ppt ou pdf de la présentation (facultatif : vous pouvez le joindre au rendu de votre travail si vous l'avez déjà, mais vous serez autorisés à le modifier jusqu'à votre soutenance).
- Les traces d'exécution sous forme d'autant de fichiers .txt qu'il y a d'automates. Vous devrez exécuter votre programme sur l'intégralité des automates de test mentionnés ci-dessus et fournir les traces d'exécution correspondantes.

Un automate non testé, ou pour lequel les traces d'exécution ne seront pas fournies, sera considéré comme un graphe sur lequel votre programme ne fonctionne pas correctement.

Tout fichier que vous utilisez (et transmettrez à votre enseignant) doit être préfixé par votre numéro d'équipe : par exemple, si vous avez un fichier « main » et si vous utiliser le langage C++, et que vous êtes dans l'équipe B2, ce fichier doit avoir le nom B2-main.cpp (ou B2_main.cpp). Attention ! Cela concerne **tous les fichiers** !

Soutenance

Calendrier : les séances de soutenance sont les séances « PROJET » indiquées sur votre agenda. Votre enseignant définira en temps utile l'ordre de passage des équipes de chaque groupe.

Durée : 30 minutes par équipe.

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 maximum.

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 (PPT ou PDF), à remettre en tant que fichier 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'objets 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 (un tableau d'entiers ou de strings par exemple ?)… **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 déterministes et déterministes ou ce sont deux structures de données différentes.

Attention :

Il est fortement conseillé de préparer une présentation de type « powerpoint ».

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 l'automate sur lequel vous allez travailler, et les mots 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 travailliez 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 projet. 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.

ELEMENTS 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 et de réponses aux questions de chacun.