Traitement d'automate fini

Reconnaissance de mots

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.

Table des matières

PROGRAMME A DEVELOPPER

Conseil : Au départ, quand vous commencerez à travailler sur le projet, ne vous occupez pas de la question des transitions epsilon et programmez directement les étapes qui viennent ensuite. Puis, dans un deuxième temps, quand la notion sera étudiée en cours, vous pourrez implémenter le fait qu'un automate puisse contenir au moins une transition epsilon.

Votre programme se déroule en plusieurs étapes :

- lecture d'un automate (dans un fichier), mise en mémoire et affichage de l'automate sur l'écran ;
- test sur la présence d'au moins une transition epsilon dans l'automate (automate asynchrone)
- si l'automate n'est pas déterministe complet, obtention de l'automate déterministe et complet équivalent ;
- calcul de l'automate minimal équivalent ;
- test de reconnaissance de mots ;
- création d'un automate reconnaissant le langage complémentaire et test de reconnaissance de mots de ce langage-ci ;

La standardisation de l'automate sans transition epsilon doit être possible à tout moment de l'arborescence, mais ce sera une opération à part : tous les traitements successifs seront effectués sur l'automate non standardisé.

Lecture d'un automate dans un fichier

Quelques temps avant la soutenance, nous vous communiquerons 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 en mémoire l'automate en fonction de votre choix de structures de données.

Le choix de l'automate sera effectué par l'utilisateur pendant l'exécution de votre programme, et ce choix doit pouvoir être fait 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.

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.

Affichage de l'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.

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)
```

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_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_synchrone (AF)
      FINSI
FINSI
afficher_automate_deterministe_complet(AFDC)
```

où :

`est_un_automate_asynchrone(AF)`

Vérifier s'il s'agit d'un automate asynchrone (contenant des transitions epsilon) 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.

`elimination_epsilon (AF)`

Remplacer tout état initial ou cible d'au moins une transition non "epsilon" par sa fermeture "epsilon" dans l'automate et ne garder que les transitions non "epsilon".

`determinisation_et_completion_asynchrone(AF)`

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

Vous avez le choix :

1) soit une déterminisation directe (affichage des fermetures epsilon demandé).

2) soit une élimination des transitions epsilon avec (`elimination_epsilon(AF)`) puis une déterminisation synchrone et une complétion.

`determinisation_et_completion_synchrone (AF)`

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

Il s'agit ici de mettre en œuvre le processus de déterminisation vu en cours et en TD. Cette étape sera effectuée sur l'automate d'origine (à moins qu'il ne soit déjà synchrone, déterministe et complet) et non pas sur l'automate standardisé même si la standardisation a été demandée par l'utilisateur.

`est_un_automate_deterministe(AF)`

Vérifier si l'automate synchrone 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 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.

```
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. La déterminisation d'un automate qui l'est déjà sera considéré comme une erreur.

* La déterminisation de l'automate n'inclut pas sa standardisation. Il est donc ici demandé de déterminiser l'automate tel qu'il est défini dans le fichier importé par votre programme.

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 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 avant de passer au mot suivant.

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 :

-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 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
```

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.

Standardisation

Votre programme doit finalement nous donner la possibilité de rendre standard l'automate à n'importe quel moment de l'enchainement des traitements.

Pseudo-code :

```
AStd ← automate_standard ( A )
```

où :

`automate_standard (A )`

Standardisation de l'automate précédent 'A'.

Si l'automate A est déjà standard, votre programme doit l'indiquer et ne pas le modifier.

Enchaînement des traitements

Si l'intégralité du code présenté ci-dessus est réalisé, on obtient successivement les automates suivant :

AF → AFDC → AFDCM → AComp

avec des flèches latérales pour effectuer la standardisation possibles à tout moment sauf si l'automate est asynchrone.

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

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

Boucle de traitement de plusieurs automates

Il est impératif 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, C++, Python ou Java.

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 A PRENDRE EN COMPTE

Les tests seront effectués sur des automates :

- ayant pour alphabet les premières lettres de l'alphabet latin au nombre indiquée 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 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.

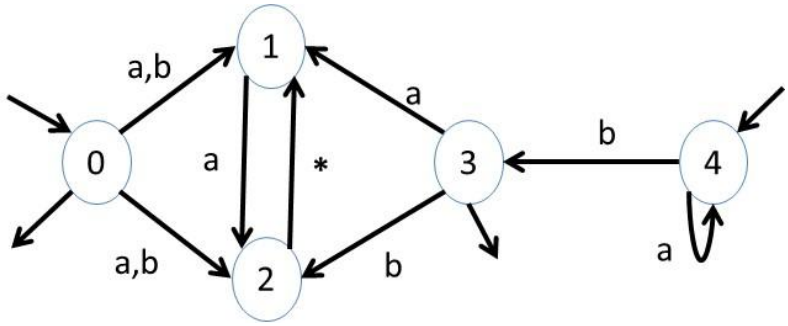
Lignes 6 et suivantes : transitions sous la forme

<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
```

DEROULEMENT DU PROJET

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 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

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 : les séances de soutenance sont les séances « PRJ » (projet) indiquées sur votre agenda. Votre enseignant vous demandera en temps utile de définir l'ordre de passage des équipes de chaque groupe.

Durée : 45 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.

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 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 ».

Dans le cas d'une soutenance en présentiel, 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.

Dans le cas d'une soutenance en visio sur Teams, votre enseignant vous donnera les précisions complémentaires. La soutenance orale restera la même, vous devrez impérativement activer votre caméra, partager votre soutenance et présenter comme en présentiel.

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. En particulier, chacun doit pouvoir expliquer tous les algorithmes. 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 de chacun.