TME Solo du 27/11/2017

Durée 1h30

Aucun document autorisé excepté le memento distribué.
Les calculatrices, baladeurs et autres appareils électroniques sont interdits. Les téléphones mobiles doivent être éteints et rangés dans les sacs.

TME solitaire numéro : TS27-11_1

Vous devez récupérer l'archive suivante :
`/Infos/lmd/2017/licence/ue/2I001-2017-oct/fournis/TS27-11_1/fournis.zip`

Pour récupérer les fichiers vous devrez décompresser le fichier `fournis.zip` avec la commande :
`unzip fournis.zip`
Il vous sera demandé un mot de passe. Ce mot de passe est : `automates`

Vous respecterez strictement les prototypes et l'organisation des fichiers.

La soumission d'un code qui ne compile pas donnera lieu à une forte pénalité.

Vous n'avez le droit qu'à un terminal, un éditeur de texte et, pour récupérer les fichiers fournis, un navigateur de fichiers. Le navigateur de fichiers sera fermé dès que les fichiers fournis auront été récupérés. Toute autre application est interdite.

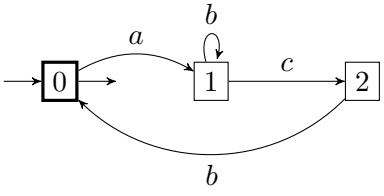
Vous soumettrez votre répertoire avec la commande `remcptcave` habituelle. Compte tenu du nom du TME, ce sera donc de la façon suivante :
`remcptcave 0 TS2711_1 <nom du répertoire>`

Automates finis

Un *automate fini* est un quadruplet $\mathcal{A} = \langle S, e_0, F, T\rangle$ où :

- S est un ensemble fini d'*états* (numérotés à partir de 0),
- e_0 est l'*état initial* (on choisira ici toujours l'état dont le numéro est 0 comme état initial),
- F est le sous-ensemble de S contenant les états appelés *états acceptants* de l'automate,
- et T est un ensemble fini de triplets appelés *transitions* : un triplet $(i, e, j) \in T$ exprime qu'il existe une transition étiquetée par la lettre e de l'état i vers l'état j (on suppose ici que les étiquettes sont des caractères différents du caractère `'\0'`).

Par exemple, pour l'automate $\mathcal{A}_1$ ci-dessous :



on a $S = \{0, 1, 2\}$, $e_0 = 0$ (ici l'état initial, symbolisé par un état sur lequel pointe une flèche sans origine, est toujours 0), $F = \{0\}$ (sur cet exemple il y a un unique état acceptant, symbolisé par un état à partir duquel part une flèche sans destination, qui est 0) et $T = \{(0, a, 1), (1, b, 1), (1, c, 2), (2, b, 0)\}$.

On considère uniquement les *automates déterministes* : à partir de chaque état il n'existe pas plus d'une transition étiquetée par un caractère donné (mais à partir d'un même état il peut exister plusieurs transitions étiquetées par des caractères différents). Toutefois, les automates considérés ne sont pas nécessairement complets : à partir d'un état il peut ne pas exister de transition pour un caractère donné (par exemple, sur l'automate $\mathcal{A}_1$ ci-dessus, à partir de l'état 1 il n'existe pas de transition étiquetée par a).

Un mot m, c-à-d une chaîne de caractères, est *accepté* par un automate $\mathcal{A}$ si en partant de l'état initial e_0 de l'automate et en effectuant les transitions obtenues en parcourant les caractères de ce mot, on arrive dans un état acceptant de l'automate à la fin du parcours des caractères de m. Si on considère l'exemple de l'automate $\mathcal{A}_1$ ci-dessus :

- le mot vide (c-à-d le mot de longueur 0) est accepté par $\mathcal{A}_1$ car l'état initial est aussi un état acceptant
- le mot *abbbcb* est accepté par $\mathcal{A}_1$ car à partir de l'état initial 0, on arrive à l'état 1 avec la lettre a, puis à l'état 1 avec la lettre b, puis à l'état 1 avec la lettre b, puis à l'état 1 avec la lettre b, puis à l'état 2 avec la lettre c, puis à l'état acceptant 0 avec la lettre b
- le mot *abcab* n'est pas accepté par $\mathcal{A}_1$ car à partir de l'état initial 0, on arrive à l'état 1 avec la lettre a, puis à l'état 1 avec la lettre b, puis à l'état 2 avec la lettre c et il n'y a aucune transition étiquetée par la lettre a à partir de l'état 2 (on ne peut donc finir le parcours des caractères du mot en se déplaçant dans l'automate)
- le mot *abcbabcb* est accepté par $\mathcal{A}_1$ car à partir de l'état initial 0, on arrive à l'état 1 avec la lettre a, puis à l'état 1 avec la lettre b, puis à l'état 2 avec la lettre c, puis à l'état 0 avec la lettre b, puis à l'état 1 avec la lettre a, puis à l'état 1 avec la lettre b, puis à l'état 2 avec la lettre c, puis à l'état acceptant 0 avec la lettre b
- le mot *abcbab* n'est pas accepté par $\mathcal{A}_1$ car à partir de l'état initial 0, on arrive à l'état 1 avec la lettre a, puis à l'état 1 avec la lettre b, puis à l'état 2 avec la lettre c, puis à l'état 0 avec la lettre b, puis à l'état 1 avec la lettre a, puis à l'état 1 qui n'est pas un état acceptant avec la lettre b

Représentation d'un automate fini avec des tableaux

Il est possible de représenter un automate fini avec le type `M_Automate` suivant (défini dans le fichier `m_automates.h`) :

```
typedef struct{
  int nb_etats;         /* nombre d'etats de l'automate */
  int nb_etats_accept;  /* nombre d'etats acceptants */
  int *accept;          /* tableau des numeros des etats acceptants */
  char *m_adj;          /* matrice d'adjacence */
}M_Automate;
```

où `nb_etats` désigne le nombre d'états de l'automate, `nb_etats_accept` désigne le nombre d'états acceptants de l'automate, `accept` désigne un tableau contenant les numéros des états acceptants de l'automate (sa taille est donc déterminée par `nb_etats_accept`) et `m_adj` est un tableau de dimension 1 contenant des caractères et représentant la matrice M des transitions de l'automate : il s'agit d'une matrice carrée de dimension `nb_etats` $\times$ `nb_etats` telle que $M[i,j] = e$ si il existe une transition étiquetée par e de l'état i vers l'état j et $M[i,j] = \backslash 0$ sinon. Par exemple, la matrice des transitions de l'automate $\mathcal{A}_1$ ci-dessus est :

$$M = \begin{bmatrix} \backslash 0 & a & \backslash 0 \\ \backslash 0 & b & c \\ b & \backslash 0 & \backslash 0 \end{bmatrix}$$

La matrice des transitions M est représentée par le tableau `m_adj` de dimension 1 contenant `nb_etats` $\times$ `nb_etats` éléments et on rappelle qu'avec une telle représentation $M[i,j] =$ `m_adj`$[(i$ `* nb_etats`$) + j]$.

Représentation d'un automate fini avec des structures chaînées

Il est possible de représenter un automate fini par une structure contenant un tableau de pointeurs sur des états. Chaque état contient un numéro `num`, un entier `acceptant` qui vaut 1 si l'état est acceptant et vaut 0 sinon, et un pointeur `L_trans` sur la liste chaînée des transitions issues de cet état. Une transition contient une étiquette `etiquette`, un pointeur `e_dst` sur l'état destination, et un pointeur `suiv` sur la transition suivante de la liste. Les types suivants (définis dans le fichier `g_automates.h`) permettent une telle représentation :

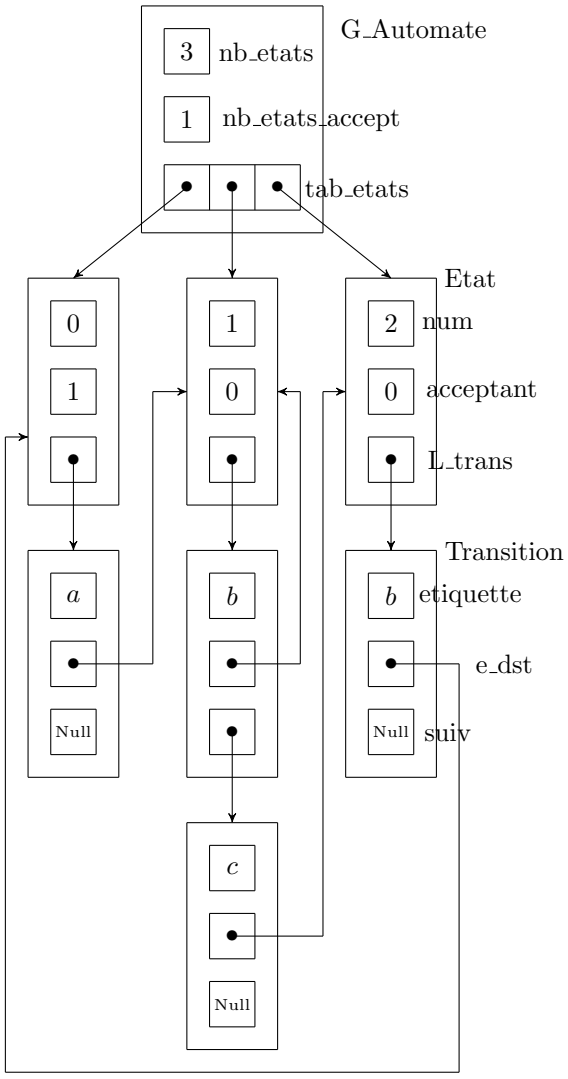
```
typedef struct etat Etat;

typedef struct transition{
  char etiquette;          /* etiquette de la transition */
  Etat *e_dst;             /* etat destination de la transition */
  struct transition *suiv; /* transition suivante a partir de l'etat d'origine */
 } Transition;

struct etat{
  int num;               /* numero de l'etat */
  int acceptant;         /* =1 si etat acceptant, =0 sinon */
  Transition* L_trans;   /* liste des transitions issues de l'etat */
 };

typedef struct {
  int nb_etats;          /* nombre d'etats */
  int nb_etats_accept;   /* nombre d'etats acceptants */
  Etat **tab_etats;      /* tableau des etats */
} G_Automate;
```

La figure ci-dessous illustre cette représentation sur l'automate $\mathcal{A}_1$ donné en exemple plus haut.



Représentation d'un automate fini dans un fichier texte

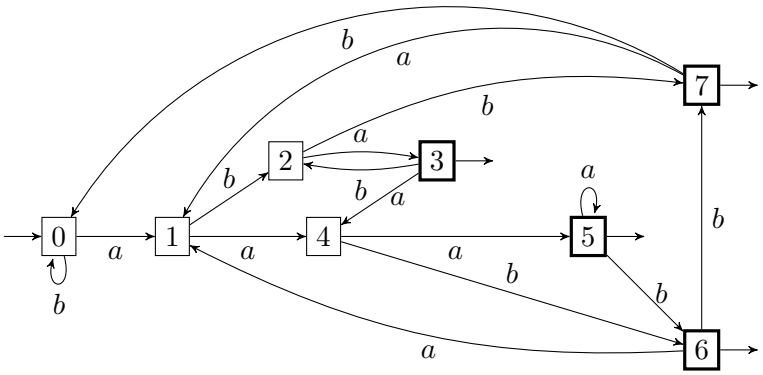
Le codage d'un automate fini dans un fichier texte peut s'effectuer comme suit :

- la première ligne du fichier contient deux entiers désignant respectivement le nombre d'états de l'automate et le nombre d'états acceptants de l'automate
- la deuxième ligne contient les numéros des états acceptants (séparés par un espace)
- les lignes suivantes contiennent les transitions : chaque transition de l'état e_1 à l'état e_2 étiquetée par le caractère c est représentée sur une ligne contenant le numéro de l'état e_1 suivi d'un espace puis du caractère c suivi d'un espace puis du numéro de l'état e_2 (l'ordre d'apparition des transitions est quelconque)

Par exemple, l'automate $\mathcal{A}_1$ défini plus haut peut être codé dans un fichier texte dont le contenu est :

```
3 1
0
0 a 1
1 b 1
1 c 2
2 b 0
```

Ce fichier `ex_auto1_txt` vous est fourni. Deux autres fichiers vous sont également fournis pour réaliser les tests de vos programmes. Le fichier `ex_auto2_txt` contient le codage de l'automate $\mathcal{A}_2$ ci-dessous :



Cet automate accepte les mots de longueur $n \geq 3$ dont la $(n-2)$-ième lettre est le caractère a. Par exemple le mot *abbabb* est accepté par $\mathcal{A}_2$ tandis que le mot *abbab* n'est pas accepté par $\mathcal{A}_2$. Le fichier `ex_auto3_txt` contient le codage d'un automate $\mathcal{A}_3$ qui accepte les mots contenant au moins trois a consécutifs (par exemple le mot *bbaaaabba* est accepté par $\mathcal{A}_3$ tandis que le mot *abbab* n'est pas accepté par $\mathcal{A}_3$).

Fichiers fournis

- `ex_auto1.txt`, `ex_auto2.txt` et `ex_auto3.txt` : fichiers textes contenant le codage des automates $\mathcal{A}_1$, $\mathcal{A}_2$ et $\mathcal{A}_3$ donnés plus haut
- `g_automates.h` et `g_automates.c`
  - contiennent les types et les fonctions permettant de manipuler des automates de type `G_Automate`
  - contiennent le prototype et la définition de la fonction :

    ```
    void affiche_g_automate(G_Automate *a)
    ```

    qui permet d'afficher le contenu d'un automate (et vous permettra de tester vos programmes)
  - dans le fichier `g_automates.c`, 4 fonctions sont à compléter (3 dans l'exercice 1 et 1 dans l'exercice 2)
- `main_g_automates.c` : permet de tester l'ensemble des fonctions définies dans `g_automates.c` (avec les automates $\mathcal{A}_1$, $\mathcal{A}_2$ et $\mathcal{A}_3$ donnés plus haut)

- `m_automates.h` et `m_automates.c`
  - contiennent les types et les fonctions permettant de manipuler des automates de type `M_Automate`
  - contiennent le prototype et la définition de la fonction :

    ```
    void affiche_m_automate(M_Automate *a)
    ```

    qui permet d'afficher le contenu d'un automate (et vous permettra de tester vos programmes)
  - dans le fichier `m_automates.c`, 2 fonctions sont à compléter (1 dans l'exercice 2 et 1 dans l'exercice 4)
- `main_m_automates.c` : permet de tester l'ensemble des fonctions définies dans `m_automates.c` (avec les automates $\mathcal{A}_1$, $\mathcal{A}_2$ et $\mathcal{A}_3$ donnés plus haut)
- `transformations.h` et `transformations.c` : contiennent les prototypes et les définitions des fonctions permettant de construire un automate de type `M_Automate` à partir d'un automate de type `G_Automate` et de construire un automate de type `G_Automate` à partir d'un automate de type `M_Automate` (1 fonction de ce fichier est à compléter dans l'exercice 3)
- `main_transformations.c` : permet de tester les fonctions définies dans `transformations.c` (avec les automates $\mathcal{A}_1$, $\mathcal{A}_2$ et $\mathcal{A}_3$ donnés plus haut)
- `MakeFile` : permet de compiler l'ensemble des fichiers de ce TME Solo

Exercices

Les 4 exercices qui suivent sont indépendants. Le code nécessaire pour les traiter de manière indépendante vous est fourni. Il y a au total 7 fonctions à définir.

Exercice 1 (Construction d'un G-Automate)
Cet exercice porte sur la construction d'automates de type `G_Automate`. Les trois fonctions à définir sont présentes dans le fichier `g_automates.c` et il est demandé de compléter le corps de ces fonctions sans changer leurs prototypes.

1. Définir la fonction :

   ```
   G_Automate *creer_g_automate(int nb_etats,int nb_etats_accept,int *accept)
   ```

   qui permet de créer (avec allocation mémoire) un automate de type `G_Automate` contenant `nb_etats` états, `nb_etats_accept` états acceptants dont les numéros sont contenus dans le tableau `accept` (de taille `nb_etats_accept`) et ne contenant aucune transition. Cette fonction retournera un pointeur sur l'automate construit.

2. Définir la fonction :

   ```
   void ajout_transition(G_Automate *a, int si, int sf, char e)
   ```

   qui permet d'ajouter une transition étiquetée par le caractère `e` de l'état `si` à l'état `sf` dans l'automate `a`. On suppose ici que la transition à ajouter n'est pas déjà présente dans l'automate.

3. Définir la fonction :

   ```
   G_Automate *charger_g_automate(char *nom_fichier)
   ```

   qui permet de construire un automate de type `G_Automate` à partir de son codage dans un fichier texte. Cette fonction pourra utiliser les deux fonctions des questions précédentes et retournera un pointeur sur l'automate construit.

Remarque. Dans les exercices qui suivent, si vous n'avez pas réussi à définir les fonctions de l'exercice 1, et que vous avez besoin de réaliser des tests à partir d'un automate de type `G_Automate`, vous pouvez le construire en effectuant les étapes suivantes :

1. construire un automate de type `M_Automate` à partir du fichier texte contenant le codage de l'automate en utilisant la fonction :

   `M_Automate *charger_m_automate(char *nom_fichier)`

   définie dans le fichier `m_automates.c`

2. transformer l'automate de type `M_Automate` obtenu à l'étape précédente en un automate de type `G_Automate` en utilisant la fonction :

   `G_Automate *m2g_automate(M_Automate *m_a)`

   définie dans le fichier`transformations.c`

La fonction `charger_m_automate` peut également être utilisée pour construire des automates de type `M_Automate` afin de réaliser des tests sur les fonctions manipulant des automates de type `M_Automate`.

Exercice 2 (Mots acceptés)
Dans cet exercice on souhaite définir deux fonctions qui permettent de déterminer si un mot (c-à-d une chaîne de caractères) est accepté par un automate : la première considère un automate de type `G_Automate` et est à compléter dans le fichier `g_automates.c` et la deuxième considère un automate de type `M_Automate` et est à compléter dans le fichier `m_automates.c`. Les fonctions `main` fournies contiennent les jeux de test présentés plus haut.

1. Définir la fonction :

   ```
   int accepte_mot_g_automate(G_Automate *a,char *mot)
   ```

   qui retourne 1 si le mot `mot` est accepté par l'automate `a` et retourne 0 sinon.

2. Définir la fonction :

   ```
   int accepte_mot_m_automate(M_Automate *a,char *mot)
   ```

   qui retourne 1 si le mot `mot` est accepté par l'automate `a` et retourne 0 sinon.

Exercice 3 (Changement de représentation)
Définir une fonction :

```
M_Automate *g2m_automate(G_Automate *g_a)
```

qui permet de construire un automate de type `M_Automate` à partir d'un automate de type `G_Automate`. L'automate construit devra accepter exactement les mêmes mots que l'automate passé en paramètre. Cette fonction est à compléter dans le fichier `transformations.c`.

Exercice 4 (Sauvegarde d'un M-Automate)
Définir une fonction :

```
int sauver_m_automate(char *nom_fichier,M_Automate *a)
```

qui permet d'écrire dans un fichier texte le codage d'un automate de type `M_Automate`. Cette fonction retourne 0 si une erreur est survenue et retourne 1 sinon. Cette fonction est à compléter dans le fichier `m_automates.c`.