TP 3

Automates en C

MPI/MPI*, lycée Faidherbe

Résumé

Dans ce T.P., on implémente les automates dans le langage C.

Dans la première temps on travaille avec des automates non déterministes (sans transitions spontanées) ; les automates déterministes en sont un cas particulier.

La dernière partie code la déterminisation des automates non déterministes.

Implémentation

Types de données fournies

Un module dicos constitué de `dicos.h` et `dicos.c`. Il définit une structure de liste chaînée ainsi qu'une structure dedictionnaire par tables de hachages qui est utilisé dans la partie III.

Listes

Les listes sont de type `liste`, ce sont des pointeurs vers des enregistrements à 2 champs.

- Si `l` est une liste, `l->val` est la tête de la liste, son premier élément.
- Si `l->suivant` est un pointeur vers le reste.
- La liste vide est le pointeur vide `NULL` ; on teste par `l == NULL`.
- On libère la mémoire occupée par une liste par `void liberer_liste(liste)`.
- On crée une liste par adjonctions successives avec la fonction `liste cons(int, liste)` qui ajoute l'entier en tête de liste.

Dictionnaires

Les dictionnaires sont implémentés par un enregistrement dont la structure principale est un tableau de liste de couples d'entiers mais on ne devrait pas utiliser leur cuisine interne. On les utilise par l'intermédiaire d'un pointeur vers un dictionnaire.
Les fonctions utilisables sont les suivantes

- `dico* creer_dico(void);` qui crée un dictionnaire vide.
- `void ajoute_entree(dico* d, int k, int v)` ajoute une association (clé, valeur) à un dictionnaire.
- `void liberer_dico(dico* d)` libère l'espace mémoire occupé par un dictionnaire.
- `int valeur_associee(dico* d, int k)` renvoie la valeur associée à une clé.
- `bool appartient_cle(dico* d, int k)` teste l'appartenance d'une clé à un dictionnaire.
- `void supprime_entree(dico* d, int k)` supprime une association d'un dictionnaire.
- `liste liste_cles(dico* d)` renvoie la liste des clés d'un dictionnaire.
- `int taille_dico(dico* d)` envoie le nombre d'associations d'un dictionnaire.
- `int obtenir_cle(dico* d)` renvoie une clé quelconque d'un dictionnaire.

Automates

Un automate $Q = (\Sigma, S, \Delta, I, T)$ est représenté à l'aide du type suivant :

```
struct afnd_s {int nAlpha;
               int nEtats;
               liste** Delta;
               bool* initiaux;
               bool* finaux;};

typedef struct afnd_s afnd;
```

Les automates sont aussi utilisés par l'intermédiaire d'un pointeur.
Champs d'une variable `aut` de type `afnd*`

- `aut->nLettres` représente le nombre de lettres de l'alphabet Σ ; ces lettres sont les entiers de 0 à $|\Sigma| - 1$ dans la variable et représentent les minuscules à partir de a
  Par exemple, si $\Sigma = \{a, b\}$, `aut->nLettres` vaut 2, a est représentée par 0 et b par 1.
  La représentation d'une lettre x est obtenue par `(int) x - (int) 'a'`.
- `aut->nEtats` représente $|S|$: les états sont les entiers de 0 à $|S| - 1$.
- `aut.Delta` est une matrice de listes telle que pour tout état $s \in Q$ et tout $x \in \Sigma$,
  si `k = (int) x - (int) 'a'` représente la lettre x, `aut->Delta[s][a]` est une liste (éventuellement vide) contenant tous les états `t` qu'on peut atteindre en lisant `k` depuis `s`.
- `aut->initiaux` est un tableaux de booléens : `aut->initiaux[s]` vaut `true` ssi l'état `s` est initial.
- `aut->finaux` est un tableaux de booléens : `aut->finaux[s]` vaut `true` ssi l'état `s` est acceptant.

On travaillera sur le fichier `TP3.c` fourni.
Pour simplifier la compilation, un fichier `makefile` est fourni qui compile les fichiers modifiés et lie les fichiers obtenus. Taper `make` depuis le répertoire contenant les fichiers génère un exécutable du nom de `TP3` que l'on exécute à l'aide de `./TP3`

I Création d'automates

Question 1 Écrire une fonction `void liberer(afnd* aut)` qui libère toute la mémoire occupée par un automate.
Attention à l'ordre des libérations et à leur exhaustivité.

Question 2 Écrire `afnd* init(int nl, int ne)` comme fonction qui créé un automate contenant `ne` états, sur un alphabet de `ne` lettres, sans aucun état initial ni final et sans aucune transition.

Question 3 Écrire `void ajout_transition(afnd* aut, int s, char x, int t)` comme fonction qui modifie l'automate `A` en lui ajoutant la transition $s \xrightarrow{x} t$.

Les automates, déterministes ou non-déterministes, sont donnés sous forme de fichiers texte où un automate est codé par les lignes suivantes.

- une ligne pour le nombre de lettres de l'automate,
- une ligne pour le nombre d'états,
- une ligne pour le nombre n_i d'états initiaux, pour un automate déterministe, $n_i = 1$,
- n_i lignes pour les numéros des états initiaux, un par ligne
- une ligne pour le nombre n_f d'états finaux,
- n_f lignes pour les numéros des états finaux, un par ligne

- les dernières lignes contiennent les transitions (une par ligne) sous la forme `d l a` avec `d` un entier correspondant au départ de la transition, `l` un caractère correspondant à son étiquette et `a` un entier correspondant à son arrivée.

Question 4 Écrire une fonction `afnd* text2afnd(char* nom)` qui lit un fichier texte formaté comme indiqué ci-dessus et qui renvoie la représentation de l'automate associé au texte.

Le sujet fournit une fonction `void afnd2dot(afnd* aut, char* nom)` qui traduit un objet de type `afnd*` en un fichier au format `.dot`. Ce fichier est exploitable par le logiciel GRAPHVIZ librement téléchargeable qui en fait une représentation graphique. Le logiciel est aussi disponible en ligne à l'adresse http://magjac.com/graphviz-visual-editor/ ; sur cette page on peut coller le texte du fichier, il sera traduit graphiquement.

Question 5 Lire les 3 fichiers, `ab.txt`, `bab.txt` et `mystere.txt` sous forme de variables de type `afnd*`, les exporter en format GRAPHVIZ afin de les visualiser.
Quels sont les langages reconnus par ces automates ?
On note `aut1` (resp. `aut2`) l'automate décrit par `ab.txt` (resp. par `ab.txt`).

Question 6 Créer et visualiser un automate qui reconnaît le langages des mots de longueur paire et qui contiennent au moins un `a`.

II Reconnaissance de mots

on représente un ensemble d'états $S' \subset S$ dans un automate par un tableau de booléens dont la s-ème case vaut `true` ssi $s \in S'$.

Question 7 Écrire une fonction `bool* Delta_partie(afnd* aut, bool* entree, int k)` prenant en entrée un automate représenté par `aut`, un ensemble d'états S' représenté par `entree` et un entier k représentant une lettre $x \in \Sigma$ et renvoyant l'ensemble $\Delta(S', x)$ des états qu'on peut atteindre dans partir d'un état de S' en lisant x : $\Delta(S', a) = \bigcup\_s \in S' \Delta(s, x)$.

Question 8 Écrire une fonction `bool* Delta_etoile(afnd* aut, bool* entree, char* u)` indiquant les extrémités des calculs d'étiquette `u` dans l'automate `aut` à partir d'un des états de l'ensemble défini par `entree`.

Question 9 En déduire une fonction `bool reconnu(afnd* aut, char* u)` renvoyant `true` si et seulement si le mot `u` est reconnu par l'automate représenté par `aut`.

Question 10 Prédire quels mots parmi $u = abbabbabaab$, $v = baababbbbba$ et $w = abaabaabb$ sont reconnus par `aut1` (resp. `aut2`). Vérifier avec la fonction `reconnu`.

III Déterminisation

L'objectif de cette partie est d'implémenter l'algorithme de déterminisation d'un automate qui construit un automate émondé. La déterminisation "naïve" consisterait à calculer l'automate des parties, qui aura systématiquement $2^{|S|}$ éléments, puis à l'émonder. On procéder à l'envers en calculant les parties accessibles depuis l'ensemble des états initiaux puis en définissant l'automate. Pour garder trace des parties accessibles, on va utiliser un dictionnaire mais un tableau n'est pas directement une clé possible. On va convertir une partie, codée sous forme d'un tableau de booléen, en entier en lisant les booléen comme des 0 ou 1 et en convertissant en base 2 en lisant les booléens de la gauche vers la droite (convention petit-boutienne) ; par exemple, on associe 5 à `[true, false, true, false]`.

Question 11 Écrire une fonction `int partie2entier(bool* etats, int taille)` prenant en entrée un ensemble d'états $S' \subset S$ représenté par le tableau de booléens `etats` dont la taille est `taille` et renvoyant dont l'écriture binaire est représentée par le le tableau.

Question 12 Écrire la fonction inverse `bool* entier2partie(int n, int taille)` qui calcule le tableau de booléen de taille `taille` représentant l'entier n en base 2.

Question 13 À l'aide des fonctions sur les dictionnaires, écrire une fonction `dico* accessibles(afnd* aut)` renvoyant un dictionnaire dont les clés sont les numéros associés aux ensembles d'états accessibles dans l'automate des parties et les valeurs sont des entiers consécutifs à partir de 0 permettant de renuméroter correctement ces états.

Question 14 Écrire une fonction `afnd* determinise(afnd* aut)` qui calcule l'automate déterminisé émondé depuis un automate.

Question 15 Définir un automate (par exemple sous la forme d'un fichier texte) qui reconnaît le langage des mots $u = x_1x_2\cdots x_n$ de $\{a,b\}^*$ avec $n \geqslant 4$ tels que $x_{n-3} = a$ et en faire dessiner le déterminisé.