TP 5 : Déterminisation

1 Objectif

L'objectif de ce TP est d'implémenter l'algorithme de déterminisation d'un automate et d'ajouter à notre interpréteur la commande correspondante.

2 Ensembles d'entiers et listes d'ensembles d'entiers

L'algorithme de déterminisation manipule des ensembles d'états. Chaque état étant associé à un entier, nous représenterons les ensembles d'états par des ensembles d'entiers. Pour cela, vous pourrez utiliser la librairie `set.c`. Cette dernière représente un ensemble d'entiers E sous la forme de la structure `set` composée d'un entier `n` indiquant l'indice du plus grand élément que peut contenir E et d'un tableau d'entiers `t` de taille `n+1`.

```
typedef struct{int n; char *t;} set;
```

Pour indiquer que l'entier $i \leq n$ est présent dans E, on affecte la valeur 1 à la case i du tableau `t`. Sinon on lui affecte la valeur 0.

Les fonctions suivantes sont définies :

`set *set_alloc(int n);` alloue la mémoire permettant de représenter un ensemble pouvant contenir les entiers $0 \leq i \leq n$ et initalise le tableau.

`int set_eq(set *e1, set *e2);` retourne 1 si les ensembles `e1` et `e2` sont égaux, 0 sinon.

`set *set_copy(set *e);` retourne une copie de l'ensemble `e`.

`int set_is_empty(set *e);` retourne 1 si l'ensemble `e` est vide, 0 sinon.

`int set_add_elt(set *e, int i);` ajoute l'entier `i` à l'ensemble `e`, augmentant sa taille si besoin. Retourne `1` si l'opération a réussi, `0` sinon.

`int set_rm_elt(set *e, int i);` enlève l'entier `i` de l'ensemble `e`. Retourne `1` si l'opération a réussi, `0` sinon.

`void set_empty(set *e);` enlève tous les éléments de l'ensemble `e`.

`void set_free(set *e);` libère la mémoire occupée par l'ensemble `e`.

L'algorithme manipule aussi des listes de sous-ensembles d'un ensemble (appelé ensemble sous-jacent), implémentées sous la forme de la structure de donnée `subset_list` définie de la façon suivante :

```
typedef struct {int set_size; int n; set** t;} subset_list;
```

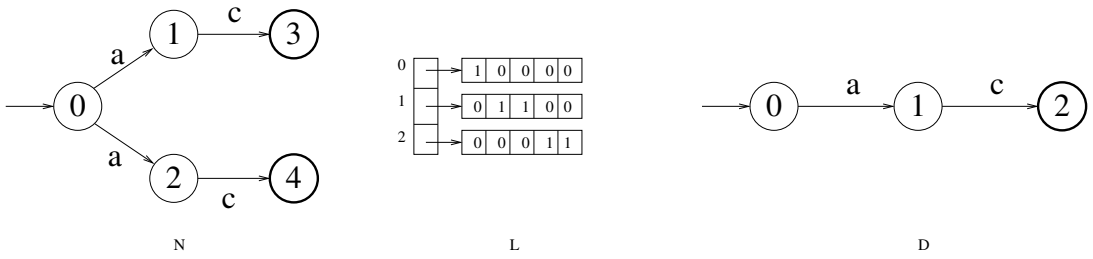

FIGURE 1 – Un automate non déterministe N, la liste L et un automate déterministe D correspondant à N.

où `set_size` représente le cardinal de l'ensemble sous-jacent, `n` représente le nombre de sous-ensembles que comporte la liste et `t` est un tableau d'ensembles : les sous-ensembles qui constituent la liste.

Les fonctions suivantes sont définies dans le fichier `set.c` :

`subset_list* subset_list_alloc (int n);` alloue la mémoire permettant de représenter une liste de sous-ensembles d'un ensemble de cardinal `n`.

`int subset_list_add (subset_list* l, set* s);` ajoute à la liste `l` l'ensemble `s` s'il n'existe pas déjà et renvoie son indice.

`short subset_list_has (subset_list* l, set* s);` renvoie `1` si l'ensemble `s` appartient à la liste `l`, `0` sinon

`void subset_list_free (subset_list* l);` libère l'espace mémoire occupé par la liste `l`.

3 Algorithme

L'algorithme de déterminisation permet de construire, à partir de l'automate $N = \langle Q, \Sigma, \delta, i, F \rangle$, un automate déterministe $D = \langle Q_D, \Sigma, \delta_D, i_D, F_D \rangle$ reconnaissant le même langage que N. L'algorithme consiste principalement à construire une liste de sous-ensembles de l'ensemble Q, chaque sous-ensemble correspondant à un état de D. Dans notre implémentation, chaque sous-ensemble est représenté sous la forme d'une structure `set` et la liste de sous-ensembles par une structure `subset_list`.

L'algorithme maintient une liste de sous-ensembles L contenant tous les sous-ensembles de Q créés jusque là. On initialise L avec l'ensemble composé du seul état i.

La liste L est parcourue depuis son début. Soit $E = \{e_1, \ldots, e_n\}$ l'élément courant de L. Pour chaque symbole s de Σ, les transitions sortantes des états $e_1 \ldots e_n$ sont parcourues et les états destinations de ces transitions sont regroupés au sein de l'ensemble E'.

A l'issue de la construction de E', si ce dernier n'est pas déjà présent dans L, alors il est ajouté **à la fin** de L et un nouvel état lui correspondant est créé dans l'automate D. L'indice de E' dans L et le numéro de l'état lui correspondant dans D sont égaux. Cela permet de savoir quel état de D correspond à quel ensemble dans L et vice-versa, comme l'illustre l'exemple de la figure 1.

L'état initial de D est l'état correspondant au sous-ensemble $\{i\}$ et les états d'acceptation de D sont tous les états correspondant à des sous-ensembles de Q contenant un état d'acceptation de N.