TP1 : Implémentation d'automates

L'objectif de ce TP est de réaliser une implémentation d'automates finis : les structures de données ainsi que les fonctions élémentaires de manipulation des automates.

Les structures de données et les fonctions constitueront une librairie composée des deux fichiers `Automaton.h` et `Automaton.c`.

1 Structures de données

L'implémentation repose sur trois types d'objets, les *états*, les *transitions* et les *automates*, modélisés successivement par les structures de données `State`, `Arc` et `Automaton` décrites ci-dessous :

```
typedef struct {int num_arcs; bool is_final; Arc* arcs;} State;
typedef struct {int symbol; int destination;} Arc;
typedef struct {int num_states; int initial_state; State* states;} Automaton;
```

Un état e est composé d'un tableau de transitions `arcs`, contentant les transitions ayant e pour origine, du champ `num_arcs` qui indique la taille du tableau `arcs` et du champ (`is_final`) qui indique s'il est état d'acceptation.

Une transition est composé d'un état destination `destination` et du symbole étiquetant la transition (`symbol`). Nous considèrerons ici que tous les symboles sont des entiers. L'entier zéro sera réservé pour représenter le mot vide.

Un automate est composé du tableau d'états `states`, d'un entier `num_states` qui indique la taille du tableau `states` et du numéro de l'état initial `initial_state`.

On trouvera dans la partie gauche de la figure 1 la représentation graphique d'un automate, et dans sa partie droite un schéma de l'implémentation du même automate.

1.1 Tableaux dynamiques

Les états d'un automate et les transitions d'un état sont stockés dans des tableaux (les tableaux `states` et `arcs`). Ne connaissant pas a priori la taille de ces tableaux, on ne peut leur associer une taille. C'est la raison pour laquelle nous utiliserons des tableaux dynamiques, dont la taille peut augmenter lors de l'exécution. Pour cela nous utiliserons la fonction :

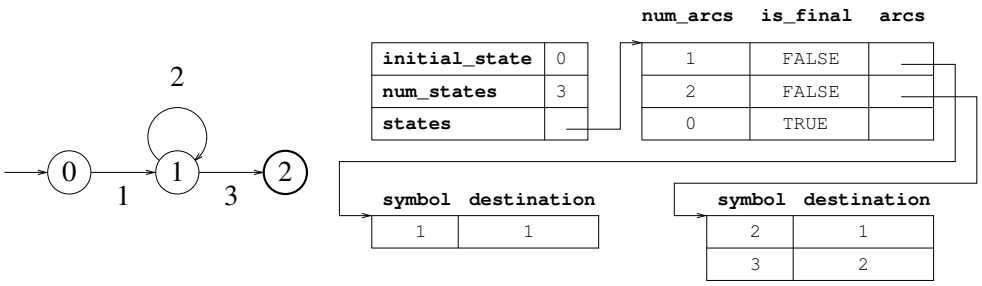

FIGURE 1 – Un automate et son implémentation

```
void *realloc(void *ptr, size_t size);
```

Cette fonction modifie la taille du bloc mémoire commençant à l'adresse `ptr` pour lui donner la taille `size` octets. Le contenu est inchangé dans la partie du bloc allant du début du bloc à son ancienne taille. Si `ptr` vaut `NULL`, alors l'appel est équivalent à `malloc(size)`.

Ainsi, à chaque fois que l'on rajoute un état à un automate ou une transition à un état, la taille du tableau correspondant devra être augmentée par un appel à la fonction `realloc`.

2 Les fonctions

Il est très important que vous respectiez scrupuleusement les prototypes de fonctions que l'on vous demande d'implémenter. Cela vous permettra d'utiliser directement certaines fonctions que l'on vous fournira plus tard.

Création d'un automate `Automaton* create_automaton();`
Alloue de la mémoire pour représenter un automate, initialise ses champs et retourne l'adresse de l'automate.

Affichage d'un automate `int print_automaton(Automaton *a);`
Réalise l'affichage à l'écran de l'automate `a`. Il consiste a afficher chaque transition sur une ligne, en indiquant dans l'ordre le numéro de l'état d'origine, le numéro de l'état destination et le symbole. On indique les états d'acceptation en les affichant seuls sur une ligne. L'état initial est l'état origine de la première transition affichée. Par exemple, l'affichage de l'automate de la figure 1 est le suivant :

```
0 1 1
1 1 2
1 2 3
2
```

Libération d'un automate `void destroy_automaton(Automaton* a);`
Libère la mémoire occupée par l'automate `a`.

Copie d'un automate `Automaton* copy_automaton(Automaton* a);`
Effectue une copie de l'automate `a` et retourne l'adresse de l'automate créé.

Ajout d'un état `int add_state(Automaton* a);`
Ajoute un nouvel état à l'automate `a` et retourne le numéro de l'état créé.

Etats d'acceptation `void set_final(Automaton *a, int state);`
Ajoute l'état `state` à l'ensemble des états d'acceptation de l'automate `a`.

Etat initial `void set_initial(Automaton *a, int state);`
Fait de l'état `state` l'état initial de l'automate `a`.

Ajout d'une transition `void add_arc(Automaton* a, int from, int to, int symbol);`
Ajoute une transition étiquetée `symbol` entre l'état numéro `from` et l'état numéro `to`. La transition est ajoutée s'il n'existe pas déjà une telle transition dans l'automate `a`. Les états `from` et `to` doivent exister au préalable dans `a`, sans quoi, la transition n'est pas créée.

Existence d'un état `bool has_state(Automaton* a, int state);`
Retourne `TRUE` si l'automate `a` possède un état numéroté `state`. Retourne `FALSE` sinon.

Existence d'une transition `bool has_arc(Automaton* a, int from, int to, int symbol);`
Retourne `TRUE` s'il existe une transition de l'état `from` à l'état `to` étiquetée par le symbole `symbol` dans l'automate `a`. Retourne `FALSE` sinon.

Elimination d'une transition `bool rm_arc(Automaton* a, int from, int to, int symbol);`
Elimine la transition dont l'état de départ est `from`, l'état d'arrivé est `to` et l'étiquette est `symbol` dans l'automate `a`. La fonction retourne `TRUE` si cette transition existait et `FALSE` sinon.

3 Test

Vous testerez votre implémentation en compilant le programme `testAutomaton.c` ce dernier construit l'automate de la figure 1 et l'affiche à l'écran.

Pour compiler votre programme, vous utiliserez le `Makefile` qui se trouve sur le site du cours.