Ecole Nationale Supérieure d'Informatique

Rapport TP n° 1 du THL

PASSER D'UN AUTOMATE SIMPLE DÉTERMINISTE À UN AUTOMATE RÉDUIT

Réalisé par : Bouhenni Sarra

Groupe : 08

Section : B

2014 - 2015

Table des matières :

Problématique :

Construire un automate simple déterministe réduit (minimal, qui contient un nombre minimum d'états) à partir d'un automate simple déterministe réduit.

Analyse générale :

Soit A un automate simple déterministe A<X, S0, S, lF, II>

A' est l'automate réduit construit à partir de A avec L (A')= L(A) et A'<X', S0', S', lF', II'>

1- On lit l'automate A<X, S0, S, lF, II>
2- On démarre avec le même état initial : S0'=S0.
3- L'ensemble S' contient les états de S qui sont accessibles et co-accessibles au même temps : S'= $E_{accessible}$ (A) ∩ $E_{Coaccessible}$ (A)
4- On initialise l'ensemble des instructions par le vide : II' = ø
5- Pour tout élément de II (Si, Wi, Sj)
6- Si (Si, Sj appartiennent les deux à S alors : II' = II' ∪ (Si, Wi, Sj)
7- IF' = IF ∩ S'

Les structures utilisées :

Structure = Etat //Cette structure contient le nom de l'état

```
char *nom;
```

Structure = instruction // Une structure contenant l'instruction : <Si, wi, Sj>

```
Etat Si;
char lettre;
Etat Sj;
```

Structure = ListEtat // Structure de la liste des états

```
Etat etat;
ListEtat *suivant;
```

Structure = ListInstruction //Structure de la liste des états

```
instruction ins;
ListInstruction *suivant;
```

Structure = Automate //Structure de l'automate

```
char X[50];
ListEtat *S ;
Etat S0;
ListEtat *F;
ListInstruction *I;
```

Liste des fonctions et procédures utilisées :

Listetat * etatsaccessibles() :

Chercher la liste des états Accessibles de l'automate A<X, S, S0, F, I>

Listetat * etatscoaccessibles() :

Chercher les états coaccessibles de l'automate A<X, S, S0, F, I>

Listetat* insertionetat(listetat *liste,char *nometat) :

La fonction insertion état insère un état donné dans la liste des états dans l'entrée et retourne la tête de la liste mise à jour.

Listinstruction* insertioninst(listinstruction *liste_I,instruction ins) :

Insérer une instruction dans la liste des instructions donnée à l'entrée et retourner la tête de la liste des instructions mise à jour.

Listinstruction* construirei(listetat *S) :

Construction de l'ensemble des instructions I' de l'automate réduit A'.

Int existe(char *nometat, listetat *etats) :

Cette fonction vérifie si un état "Sj" existe déjà dans la liste des états "etats"

Int existe Inst(instruction inst, listinstruction *l) :

Vérifier si une instruction existe déjà dans la liste des instructions donnée à l'entrée ou non.

Int existe_alphabet(char* alphabet,char c) :

Tester si un état donné est final ou non.

Int existe_alphabet(char* alphabet,char c) :

Tester si une lettre 'c' existe déjà dans le tableau de l'alphabet donné à l'entrée.

Void lire_automate() :

Lecture de l'automate déterministe simple non réduit inséré par l'utilisateur

L'alphabet : X

Les états de l'automate : S

L'état initial de l'automate : S0

Les états finaux de l'automate : F

La liste des instructions de l'automate : I

Void affiche_automate_non_red() - void affiche_automate_red();

Les deux fonctions affichent l'automate déterministe simple non réduit et après réduction.

Void affiche_listinstruction(listinstruction *p) :

Affichage de la liste des instructions dans une liste d'instructions donnée.

Void affiche_listetat(listetat * p) :

Afficher la liste des états contenus dans une liste donnée à l'entrée de la fonction.

Void construire_automate_reduit() :

Construction de l'automate réduit A' qui est identique à l'automate A

Quelques algorithmes :

1. Algorithme de recherche des états accessibles :

```
ListEtat * etatsAccessibles()
{
  int trouve=1;
  ListEtat *liste=NULL;
  ListInstruction *q=NULL;

   //on initialise la liste des états accessibles par l'état initial de l'automate qui est accessible par définition
  liste=insertionEtat(liste,automate_non_reduit.S0.nom);

while(trouve)  // tant qu'il y a des états accessibles ayant des successeurs , on fait le test
 {
   q=automate_non_reduit.I;
   trouve=0;
   while(q!=NULL) // on parcourt la liste des instructions
   {
      if(existe(q->ins.Si.nom,liste)&& existe(q->ins.Sj.nom, liste)==0)
           // si dans une instruction, l'état de départ "Si" est accessible, et l'état d'arrivée "Sj"
         // n'appartient pas à la liste des états accessibles, on insère « Sj » dans la liste des états
        // accessibles
      {
            liste=insertionEtat(liste,q->ins.Sj.nom);
            trouve=1;        // un nouveau état accessible trouvé, donc il se peut qu'il ait des successeurs
                             // qui seront des états accessibles
      }

      q=q->suivant;      // passer à la prochaine instruction
   }
 }
   return liste;    // retourne la liste des états accessibles de l'automate non réduit A
}
```

2. Algorithme de recherche des états coAccessibles :

```
ListEtat * etatscoAccessibles()
{
   int trouve=1;
   ListInstruction *q=NULL;
   ListEtat *p=NULL, *liste=NULL;
   p=automate_non_reduit.F;

   while(p!=NULL) {     // les états finaux sont tous des états coAccessibles
                        // donc on les insère dans la liste des états coAccessibles
            liste=insertionEtat(liste,p->etat.nom);
            p=p->suivant;
}

 while(trouve) // tant qu'il y a des états coAccessibles ayant des prédécesseurs , on fait le test
 {
   q=automate_non_reduit.I;
   trouve=0;
   while(q!=NULL)  // on parcourt la liste des instructions
   {
      if(existe(q->ins.Sj.nom,liste)&& existe(q->ins.Si.nom, liste)==0)
          // si l'état coAccessible est l'état d'arrivée d'une instruction
         // alors son état de départ est aussi coAccessible
        // on l'insère dans la liste des états coAccessibles
      {
         liste=insertionEtat(liste,q->ins.Si.nom);
         trouve=1;   // un nouveau état coAccessible trouvé et il se peut qu'il ait des prédécesseurs
                     // qui seront des états coAccessibles
      }

      q=q->suivant; // on passe à la prochaine instruction dans la liste des instructions
   }

 }
 return liste;     // on retourne la liste des états coAccessibles trouvés
}
```

3. Algorithme qui fait l'intersection entre deux listes d'états :

```
ListEtat *intersection(ListEtat *p, ListEtat *q)
{
   ListEtat  *tmp=NULL;
   tmp=q;      // une variable temporaire
   ListEtat *SS=NULL;

   while(p!=NULL)        // On parcourt la première liste
   {
      q=tmp;
      while(q!=NULL)     // on parcourt la deuxième liste
      {
         if(strcmp(p->etat.nom,q->etat.nom)==0)         // si deux états ont le même nom donc
                                                        // on les insère à la nouvelle liste crée
         {
            SS=insertionEtat(SS,p->etat.nom);
         }
         q=q->suivant;
      }
      p=p->suivant;
   }
   return SS;      // on retourne la tête de la nouvelle liste contenant les états en commun
}
```

4. Algorithme de construction de l'automate réduit

```
void construire_automate_reduit()
{
   ListEtat *p=NULL, *q=NULL,*SS=NULL, *FF=NULL;
   ListInstruction *II;
   p=etatsAccessibles();          // créer la liste des états accessibles
   q=etatscoAccessibles();    // créer la liste des états coAccessibles
   SS=intersection(p,q);     // la nouvelle liste d'états S' = l'intersection entre la liste
                             //  des états accessibles et la liste des états coAccessibles
   FF=intersection(automate_non_reduit.F, SS); // La nouvelle liste des etats finaux
                                               // F'= l'intersection entre l'ancien F et le nouvel ensemble S'

   II=construireI(SS);     // On construit la nouvelle liste d'instructions à partir de S'
  int k=0;                 // on reconstruit l'ensemble X = l'Alphabet
  for(k=0;k<strlen(automate_non_reduit.X);k++)
                { automate_reduit.X[k]=automate_non_reduit.X[k]; }
   automate_reduit.S=SS;
   automate_reduit.F=FF;
   automate_reduit.I=II;
   automate_reduit.S0=automate_non_reduit.S0;
}
```

5. Algorithme de construction de l'ensemble des instructions de l'automate réduit

```
ListInstruction* construireI(ListEtat *S)
{
   // on initialise la nouvelle liste des instructions par le vide
   ListInstruction *II=NULL, *i=automate_non_reduit.I;
   instruction j;
   while(i!=NULL) // on parcourt l'ancienne liste des instructions I
   {
      if(existe(i->ins.Si.nom,S)&& existe(i->ins.Sj.nom,S))
           // si l'état de départ et l'état d'arrivée d'une instruction existent
         // dans la nouvelle liste d'instructions, donc on l'insère dans I'
      {
         j.lettre=i->ins.lettre;
         j.Si=i->ins.Si;
         j.Sj=i->ins.Sj;
         II=insertionInst(II,j);
      }
      i=i->suivant;
   }
return II;   // on retourne la nouvelle liste d'instructions I'
}
```

Déroulement d'un exemple :

Soit l'automate A< X, S, S_0, F, I> avec :

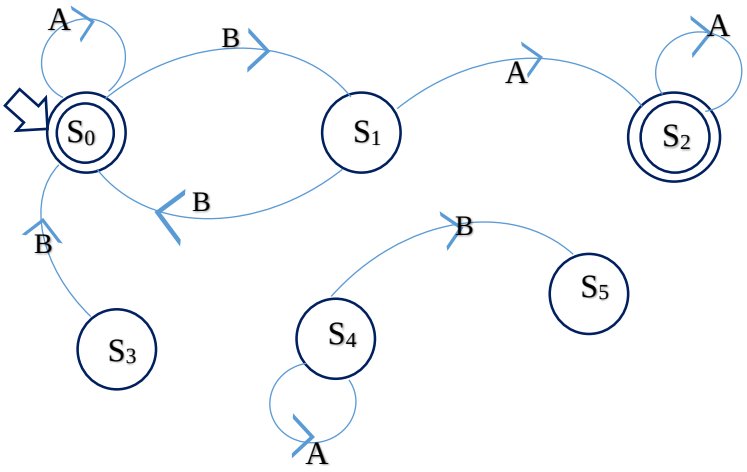
X = {a, b}

S = {S_0, S_1, S_2, S_3, S_4, S_5}

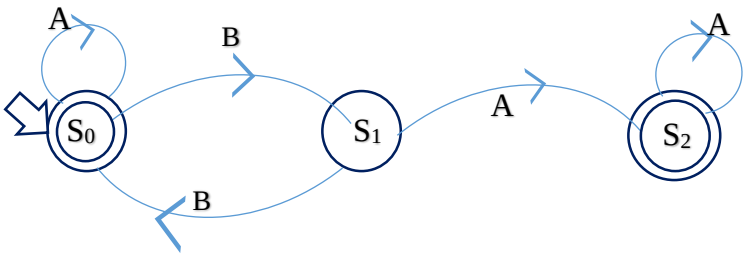
S_0= {S_0}

F = {S_0, S_2, S_4}

I = {(S_0, a, S_0), (S_0, b, S_1), (S_1, b, S_0), (S_1, a, S_2), (S_2, a, S_2), (S_3, b, S_0), (S_4, a, S_4), (S_4, b, S_5)}



L'automate A' réduit avec L(A)= L(A') :



L'automate A' qui est un automate simple déterministe réduit est définit par :

X' = {a, b}

S' = {S_0, S_1, S_2, S_3, S_4, S_5}

S'_0= {S_0}

F'= {S_2, S_4}

I' = {(S_0, a, S_0), (S_0, b, S_1), (S_1, b, S_0), (S_1, a, S_2), (S_2, a, S_2)}

Résultats du l'exécution de l'algorithme de réduction d'un automate sur le même exemple :

Lecture de l'automate A :

```
        +-------------------------------------------------------------------------------+
        +       Construction d'un Automate simple deterministe  A<X, S, S0, F, I>       +
        +-------------------------------------------------------------------------------+


  Entrer le nombre de lettres de l'alphabet  : 2
  Entrer la lettre Num  1   :   A
  Entrer la lettre Num  2   :   B

  Entrer le nombre  d'etats  :    6

         Alphabet :  AB

  La liste des etats :

   - etat : S0
   - etat : S1
   - etat : S2
   - etat : S3
   - etat : S4
   - etat : S5

  Entrer le nom de l'etat initial (Majiscule) :   S0

  Entrer le nombre d'etats finaux  :      3
  Entrer le nom de l'etat final Num  1 :   S0
  Entrer le nom de l'etat final Num  2 :   S2
  Entrer le nom de l'etat final Num  3 :   S4
```

Affichage de l'automate après la lecture :

```
       +-----------------------------------------------------------+
       +       Affichage de l'automate A<X, S, S0, F, I>           +
       +-----------------------------------------------------------+


L'alphabet de l'automate A, X = {  A ,B }


liste des etats de l'automate A  :

 - etat : S0
 - etat : S1
 - etat : S2
 - etat : S3
 - etat : S4
 - etat : S5


l'etat initial est :    < S0 >


les etats finaux l'automate A  :

 - etat : S0
 - etat : S2
 - etat : S4


La liste des instructions de l'automate A

  1 - <S0,B,S1>
  2 - <S0,A,S0>
  3 - <S1,A,S2>
  4 - <S1,B,S0>
  5 - <S2,A,S2>
  6 - <S3,B,S0>
  7 - <S4,B,S5>
  8 - <S4,A,S4>


 Veuillez appuyer sur une touche pour afficher l'automate apres la reduction.
```

Affichage de l'automate après la réduction :