TD7 : Automates reconnaisseurs et programmation d'automates à base de conditions

Stéphane Devismes

Université Grenoble Alpes

18 mars 2016

Plan

Plan

Cas particulier des automates de Mealy

Les automates reconnaisseurs ont deux particularités :

- Ils ont **au moins un état final** (dans ce cadre, les états finals sont appelés états accepteurs) et
- **Ils n'ont pas de sortie** (leur fonction de sortie n'est définie sur aucun couple état / entrée).

Plan

Définitions

Définitions

- On appelle alphabet un ensemble de symboles (atomiques) appelés **lettres**.

Définitions

- On appelle alphabet un ensemble de symboles (atomiques) appelés **lettres**.
- Un mot (fini) sur un alphabet A est une suite ordonnée de symboles de A.

Définitions

- On appelle alphabet un ensemble de symboles (atomiques) appelés **lettres**.
- Un mot (fini) sur un alphabet A est une suite ordonnée de symboles de A.
- Un langage sur un alphabet donné est un sous-ensemble de mots sur cet alphabet.

Langages et expressions régulières

Les langages sont souvent définis à l'aide d'expressions régulières (la liste ci-dessous n'est pas exhaustive) :

Langages et expressions régulières

Les langages sont souvent définis à l'aide d'expressions régulières (la liste ci-dessous n'est pas exhaustive) :

- ε représente le mot vide

Langages et expressions régulières

Les langages sont souvent définis à l'aide d'expressions régulières (la liste ci-dessous n'est pas exhaustive) :

- ε représente le mot vide
  (**remarque :** le mot vide est reconnu si l'état initial fait parti des états finals).

Langages et expressions régulières

Les langages sont souvent définis à l'aide d'expressions régulières (la liste ci-dessous n'est pas exhaustive) :

- ε représente le mot vide
  (**remarque :** le mot vide est reconnu si l'état initial fait parti des états finals).
- $(abc)^+$ signifie une à plusieurs répétitions de *abc*.

Langages et expressions régulières

Les langages sont souvent définis à l'aide d'expressions régulières (la liste ci-dessous n'est pas exhaustive) :

- ε représente le mot vide
  (**remarque :** le mot vide est reconnu si l'état initial fait parti des états finals).
- $(abc)^+$ signifie une à plusieurs répétitions de *abc*.
- $(abc)^*$ signifie zéro à plusieurs répétitions de *abc*.

Langages et expressions régulières

Les langages sont souvent définis à l'aide d'expressions régulières (la liste ci-dessous n'est pas exhaustive) :

- ε représente le mot vide
  (**remarque :** le mot vide est reconnu si l'état initial fait parti des états finals).
- $(abc)^+$ signifie une à plusieurs répétitions de *abc*.
- $(abc)^*$ signifie zéro à plusieurs répétitions de *abc*.
- $[a\text{-}d]$ signifie une lettre entre *a* et *d*.

Langages et expressions régulières

Les langages sont souvent définis à l'aide d'expressions régulières (la liste ci-dessous n'est pas exhaustive) :

- ε représente le mot vide
  (**remarque :** le mot vide est reconnu si l'état initial fait parti des états finals).
- $(abc)^+$ signifie une à plusieurs répétitions de *abc*.
- $(abc)^*$ signifie zéro à plusieurs répétitions de *abc*.
- $[a\text{-}d]$ signifie une lettre entre *a* et *d*.
- $(ab|cd)$ signifie les lettres *ab* ou les lettres *cd*.

Exemples

- **Nombre entier naturel :**

Exemples

- **Nombre entier naturel :** $(0|[1\text{-}9][0\text{-}9]^*)$ sur l'alphabet $\{0,\ldots,9\}$.

Exemples

- **Nombre entier naturel :** $(0|[1\text{-}9][0\text{-}9]^*)$ sur l'alphabet $\{0,\ldots,9\}$.
- Soit A^* l'ensemble des mots (finis) de symboles de A.

Exemples

- **Nombre entier naturel :** $(0|[1\text{-}9][0\text{-}9]^*)$ sur l'alphabet $\{0,\ldots,9\}$.
- Soit A^* l'ensemble des mots (finis) de symboles de A.

  Si $A = \{a, b, \ldots, z\}$, alors A^* est **l'ensemble de tous les mots (finis) formés sur les lettres de l'alphabet latin**.

Reconnaissance

Le but ici est de **décider si un mot**, dont chacune des lettres est fournie successivement en entrée de l'automate, **appartient ou non à un langage donné**.

Reconnaissance

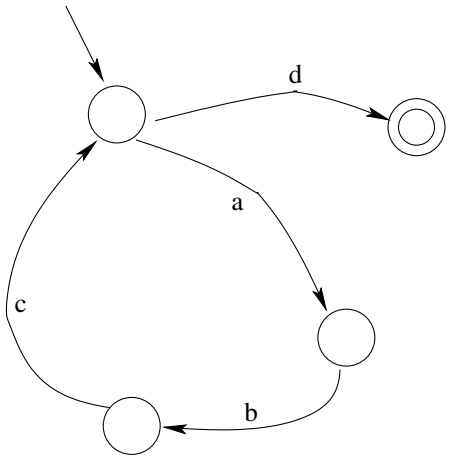
Le but ici est de **décider si un mot**, dont chacune des lettres est fournie successivement en entrée de l'automate, **appartient ou non à un langage donné**.

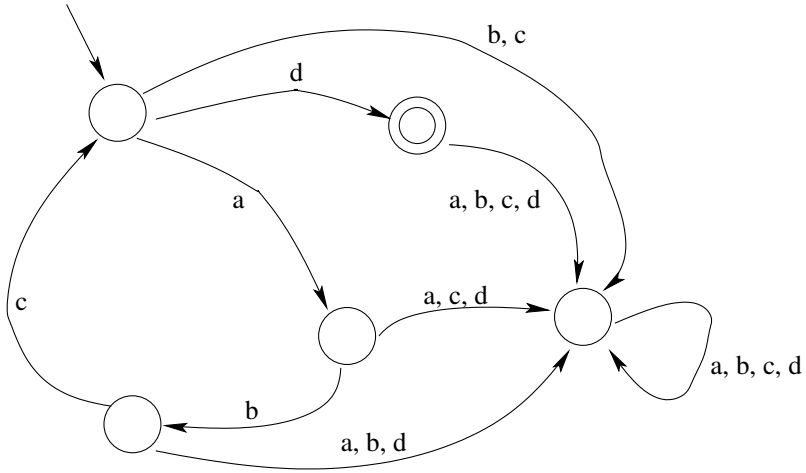
Plus formellement :

- Un chemin $p_0, \ldots, p_n$ est étiqueté par $e_0 \ldots e_{n-1}$ si pour tout $i \in [0..n-1]$, la fonction de transition de l'automate *trans* vérifie $trans(p_i, e_i) = p_{i+1}$.
- Un mot m est accepté (ou reconnu) par l'automate **s'il existe un chemin étiqueté par m démarrant dans l'état initial et terminant dans un état accepteur (final).**

Exemple

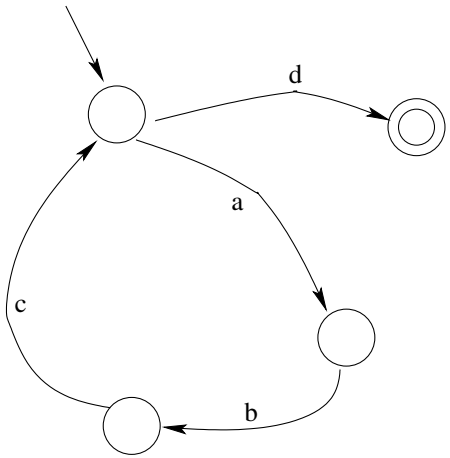
Considérons l'alphabet $A = \{a, b, c, d\}$. Dans la figure ci-dessous, nous donnons deux automates (l'un incomplet, l'autre complet) qui acceptent les mots du langage $(abc)^* d$.

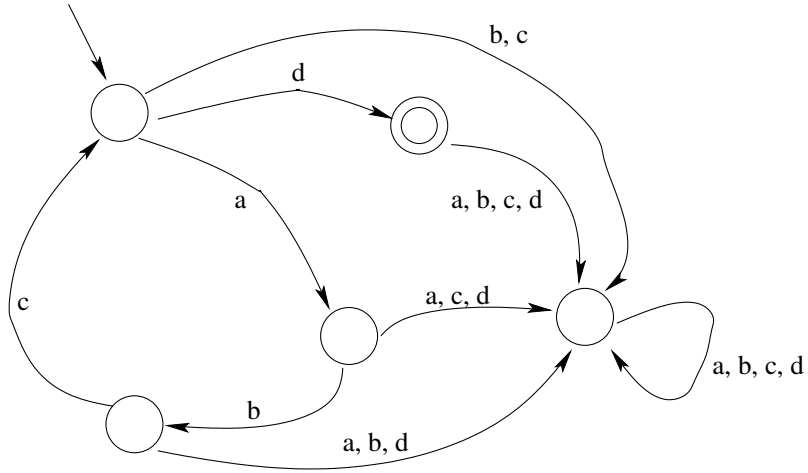




Exemple

Considérons l'alphabet $A = \{a, b, c, d\}$. Dans la figure ci-dessous, nous donnons deux automates (l'un incomplet, l'autre complet) qui acceptent les mots du langage $(abc)^* d$.





Remarque : un état accepteur (donc final) n'est pas forcément un puits, on a le droit d'en sortir !

La règle est bien que le mot est accepté si on est dans un état final lorsqu'il n'y a plus d'entrée.

Applications

Applications

- Au niveau de la compilation, ils sont utilisés lors de l'**analyse lexicale** pour reconnaître les différents mots clés du langage (par exemple, les identificateurs, les constantes).

Applications

- Au niveau de la compilation, ils sont utilisés lors de l'**analyse lexicale** pour reconnaître les différents mots clés du langage (par exemple, les identificateurs, les constantes).
- Ils sont aussi utilisés dans les commandes UNIX qui gèrent les expressions régulières, comme **grep** ou **sed** par exemple.

Plan

Exemple

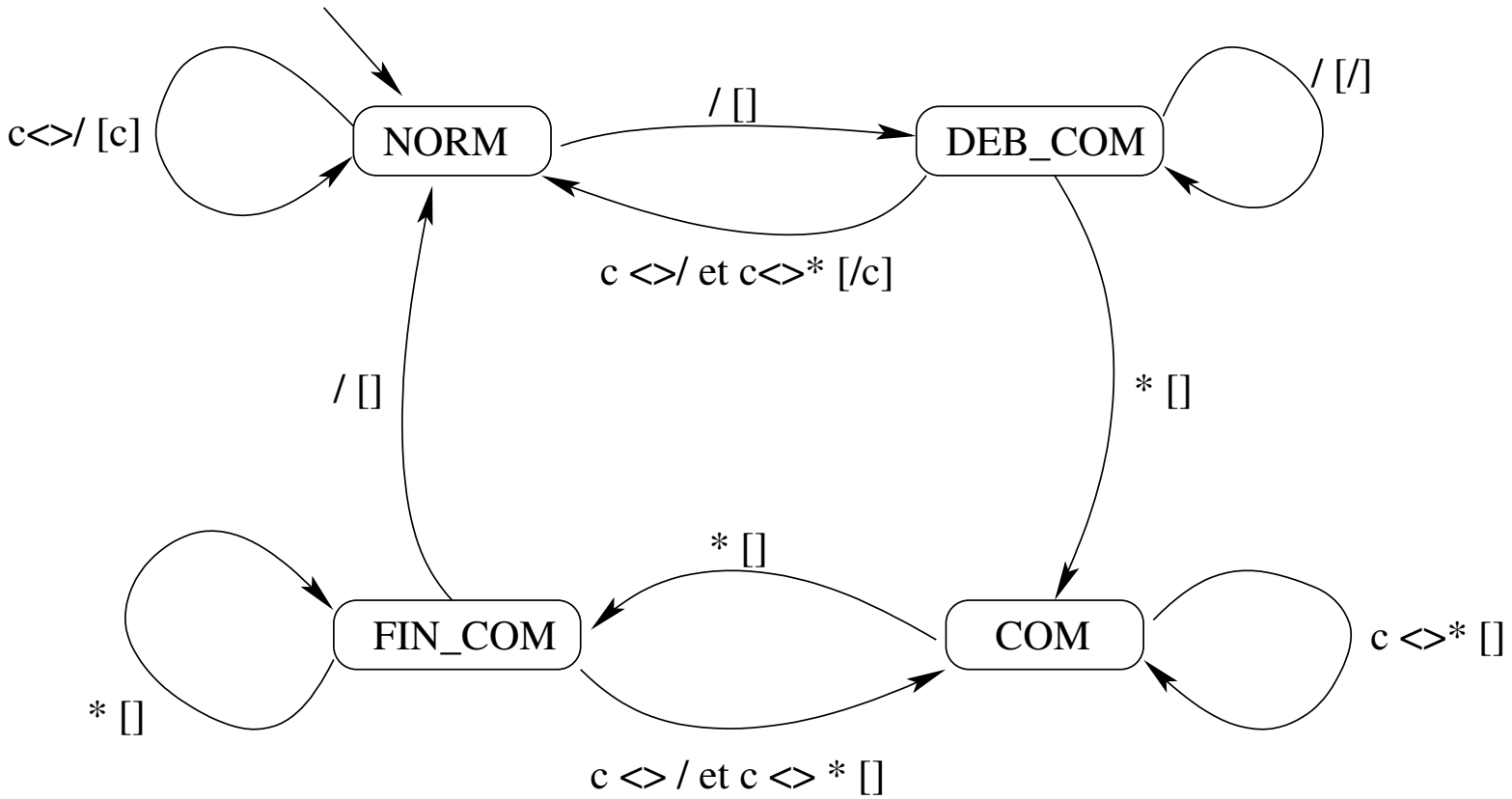
Dessiner l'automate qui **supprime des commentaires en C**.

Exemple

Dessiner l'automate qui **supprime des commentaires en C**.
(Les sorties seront indiquées entre [] pour éviter les confusions) (*c* désignera un caractère quelconque, on pourra aussi mettre des conditions sur *c*)

Exemple

Dessiner l'automate qui **supprime des commentaires en C**.
(Les sorties seront indiquées entre [] pour éviter les confusions) (*c* désignera un caractère quelconque, on pourra aussi mettre des conditions sur *c*)



Rappel : algorithme

```
etat_courant = Init ;

while (! FINI ) {
     entree = lire_entree() ;
     sortie = sortie(etat_courant, entree) ;
     etat_suivant = transition(etat_courant, entree) ;
     traiter_sortie(sortie) ;
     etat_courant = etat_suivant ;
     mise a jour de FINI
    }
```

États

```
#include <stdio.h>
#include <stdlib.h>

#define NORM 1
#define DEB_COM 2
#define COM 3
#define FIN_COM 4
```

Fonction de sortie (1/2)

```
void f_sortie (int etat, char symb, char sortie[])
{
  switch (etat)
  {
     case NORM :
          switch (symb)
          {
            case '/' :          /* pas de sortie */
               sortie[0]='\0' ;
               break ;
            default :           /* texte "normal", sortie=symb */
               sortie[0]=symb ;
               sortie[1]='\0' ;
               break ;
          }
          break ;

          /* suite : slide suivant */
```

Fonction de sortie (2/2)

```
    case DEB_COM  :
         switch (symb)
         {
           case '*' :           /* pas de sortie */
              sortie[0]='\0' ;
              break ;
           case '/' :      /* sortie= '/' */
              sortie[0]='/' ;
              sortie[1]='\0' ;
              break ;
           default :           /* sortie= '/', symb */
              sortie[0]='/' ;
              sortie[1]=symb ;
              sortie[2]='\0' ;
              break ;
         }
         break ;

    case COM :
    case FIN_COM :               /* pas de sortie */
              sortie[0]='\0' ;
         break ;

  }
}
```

Fonction de transition (1/4)

```
int f_transition (int etat, char symb)
```

Fonction de transition (1/4)

```
int f_transition (int etat, char symb)


{
int etat_suiv ;
  switch (etat)
  {
     case NORM :
          switch (symb)
          {
            case '/' :       /* debut de commentaire ??? */
               etat_suiv = DEB_COM ;
               break ;

            default  :      /* texte "normal" */
                  etat_suiv = NORM ;
               break ;
          }
          break ;

          /* suite : slide suivant */
```

Fonction de transition (2/4)

```
case DEB_COM :
     switch (symb)
     {
       case '*' :      /* debut de commentaire */
            etat_suiv = COM ;
          break ;
       case '/' :      /* debut de commentaire ??? */
            etat_suiv = DEB_COM ;
          break ;
       default  :      /* retour au texte "normal" */
            etat_suiv = NORM ;
          break ;
     }
     break ;

     /* suite : slide suivant */
```

Fonction de transition (3/4)

```
case COM :
     switch (symb)
     {
       case '*' :      /* fin de commentaire ??? */
            etat_suiv = FIN_COM ;
          break ;
       default  :      /* texte commente */
            etat_suiv = COM ;
          break ;
     }
     break ;

     /* suite : slide suivant */
```

Fonction de transition (4/4)

```
      case FIN_COM :
           switch (symb)
           {
             case '/' :      /* fin de commentaire */
                  etat_suiv = NORM ;
                break ;
             case '*' :      /* fin de commentaire ??? */
                  etat_suiv = FIN_COM ;
                break ;
             default  :      /* retour au texte commente */
                  etat_suiv = COM ;
                break ;
           }
           break ;
   }
return etat_suiv ;
}
```

Programme principal (1/2)

```
int main (int argc, char *argv[])
{
FILE *fichier_entree, *fichier_sortie ;
char cc ;
char sortie[3] ; /* deux caracteres au maximum, plus le '\0' */
int etat_courant, etat_suivant ;

     if (argc != 3) {
          printf(" il faut 2 arguments !\n") ;
             return 1 ;
             } ;

       fichier_entree = fopen(argv[1], "r") ;
       if (fichier_entree == NULL) {
                /* si le fichier n'a pas pu etre ouvert */
             printf("Le fichier %s n'existe pas\n", argv[1]) ;
             return 2 ;
                } ;

       fichier_sortie = fopen(argv[2], "w") ;

          /* suite : slide suivant */
```

Programme principal (2/2)

```
        etat_courant = NORM ;

        fscanf(fichier_entree, "%c", &cc) ; /* cc est le 1er caractere du texte */

        while (!feof(fichier_entree)) {
           etat_suivant = f_transition (etat_courant, cc) ;

           f_sortie(etat_courant, cc, sortie) ;
           fprintf(fichier_sortie, "%s", sortie) ;

           etat_courant = etat_suivant ;

           fscanf(fichier_entree, "%c", &cc) ;
        }

      fclose(fichier_entree) ;
      fclose(fichier_sortie) ;

      return 0 ;
}
```