Analyse Lexicale

Chapitre exrait du livre : Compilation Cours et exercices

Auteur : H. DRIAS

Edition : OPU

Dans ce chapitre, nous essayerons de développer une démarche générale pour l'implémentation des analyseurs lexicographiques. Cette approche s'appuie principalement sur les techniques utilisées dans la construction automatique des analyseurs lexicographiques.

L'idée principale consiste à spécifier les entités lexicales à l'aide d'expressions régulières. Ces dernières seront converties en un automate déterministe équivalent ayant un nombre minimum d'états. L'automate ainsi obtenu sera stocké en mémoire dans une structure de données ensuite simulé par un programme.

Avant d'aborder toutes ces techniques, nous exposerons une approche simple non déterministe qui peut être adoptée dans les compilateurs de langages simples. En dernier lieu nous présenterons un langage pour décrire les analyseurs lexicographiques. Nous utiliserons la syntaxe du langage conçu par lesk dans lex 'A Lexical Analyser Generator'.

Rappelons enfin que vu la structure physique du compilateur adoptée dans ce cours, l'analyseur lexicographique se comportera comme une procédure et plus exactement comme une fonction pour l'analyseur syntaxique. Par conséquent, le code de chaque entité lexicale reconnue sera renvoyé à l'analyseur syntaxique.

2.1 ROLE ET FONCTIONS DE L'ANALYSE LEXICOGRAPHIQUE.

Le rôle primordial de l'analyse lexicographique est le découpage du texte du programme source en mots du langage qui représentent les entités lexicographiques ou tokens. Les tokens sont constitués au fur et à mesure de la lecture du texte source caractère par caractère. Les types de tokens sont spécifiques au langage. Toutefois, dans tout langage, il est à distinguer entre les entités intrinsèques au langage et les entités impropres qui sont créées par le programmeur.

Les mots clés, les opérateurs et les séparateurs sont des entités lexicales propres au langage alors que les identificateurs et les constantes sont des entités conçues par le programmeur.

Dès qu'une entité lexicale est reconnue, le scanner ou

l'analyseur lexicographique doit renvoyer un code à l'analyseur syntaxique.

Le code d'un token propre est constitué d'un nombre spécifiant la nature de ce code. Cependant le code d'un token impropre est formé de la paire (type,lexval) où type dénote la nature de l'entité et lexval fournit la valeur ou le nom de cette entité. Le scanner produit des codes pour l'analyseur syntaxique en vue d'alléger le traitement syntaxique.

En plus de la reconnaissance et du codage des entités lexicales, le scanner a d'autres fonctions telles que la mémorisation des tokens impropres dans la table des symboles. Des qu'un token impropre est rencontré, son type ainsi que sa valeur sont enregistrés dans la table et le code (type,lexval) est transmis à l'analyseur syntaxique. Lexval est la position du type et du nom de l'entité lexicographique dans la table des symboles.

Certains scanners procèdent à une analyse préliminaire qui consiste à nettoyer le texte source des symboles superflus tels que les caractères blanc et tab du code ASCII. Pour le cas du langage FORTRAN, tous les caractères blancs sont abolis au préalable de l'analyse lexicale, le blanc n'étant pas considéré comme un séparateur. Par contre pour le cas de PASCAL, à une suite consécutive de blancs ou de tabs est substituée un seul blanc.

Les commentaires sont aussi supprimés du texte au préalable. Et dans certains analyseurs lexicographiques, on procède en parallèle de ces traitements au numérotage des lignes.

2.1.1 Nécéssité De La Bufférisation.

La reconnaissance du type d'un token nécéssite quelquefois, la lecture des caractères qui suivent le token. Par exemple en FORTRAN, pour détecter le type du token IF de la chaine suivante

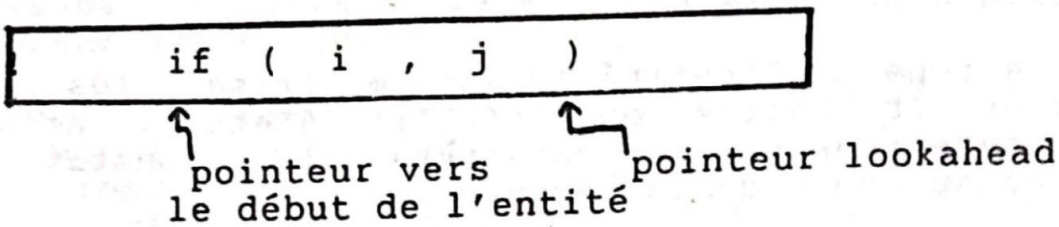
IF(i,j) = 3,

il faut lire tous les caractères qui suivent IF jusqu'au caractère qui suit la parenthèse fermante car IF peut être dans ce cas le mot clé IF ou un identificateur utilisé pour

l'appelation d'un tableau.

Une fois le type de l'entité IF est maitrisé, les entités qui suivent IF telles que '(', 'i','j'etc... ne peuvent plus être détectées car les caractères les constituant se sont égarés au cours de la lecture.

Pour pallier à cette situation, la technique de bufferisation est souvent utilisée. Les caractères, avant d'être traités sont mémorises dans un buffer de taille donnée.



On utilise donc deux pointeurs, un pointeur vers le début de l'entité à reconnaitre et un pointeur lookahead parcourant les caractères qui suivent l'entité jusqu'à un certain caractère caractérisant le type de l'entité.

Après le traitement pertinent de l'entité, le pointeur vers le début de l'entité se déplace vers le caractère qui suit le dernier caractère de l'entité et le processus est répété.

L'analyse préliminaire peut diminuer l'éloignement du pointeur lookahead et de cette façon, le scanner requérra un buffer de taille tolérable.

2.1.2 Mots Réservés.

Certains langages ne tolèrent pas que les mots clés soient utilisés comme des identificateurs. Tel est le cas pour PASCAL. Dans ce cas, les mots clés sont installés initialement dans la table des symboles. Quand un identificateur au sens large c'est à dire un token dénoté par l'expression régulière d'un identificateur est rencontré, il est comparé avec les mots clés de la table des symboles. S'il coincide avec un des mots clés de la table, le scanner génère le code de ce mot clé sinon le code d'un identificateur est transmis. Cette stratégie est souvent adoptée à côté de celle exposée ultérieurement vu sa simplicité.

2.2 UNE APPROCHE SIMPLE POUR L'IMPLEMENTATION D'UN SCANNER.

La procédure de développement d'un analyseur lexicographique consiste dans un premier temps en la description des entités lexicales à l'aide d'expressions régulieres. Ensuite ces expressions régulières sont transformées en des automates équivalents qui seront implémentés sur machine.

exemple:

soit à implémenter un scanner d'un langage possédant les entités lexicales suivantes:

Les mots clés:
- procedure
- function
- for
- while

Les identificateurs
Les constantes

Les operateurs: -
+
*
/

Les expressions régulières qui dénotent ces entités sont respectivement:

```
procedure
function
for
while
lettre(lettre/chiffre)*
chiffre(chiffre)*
+
-
*
/
```

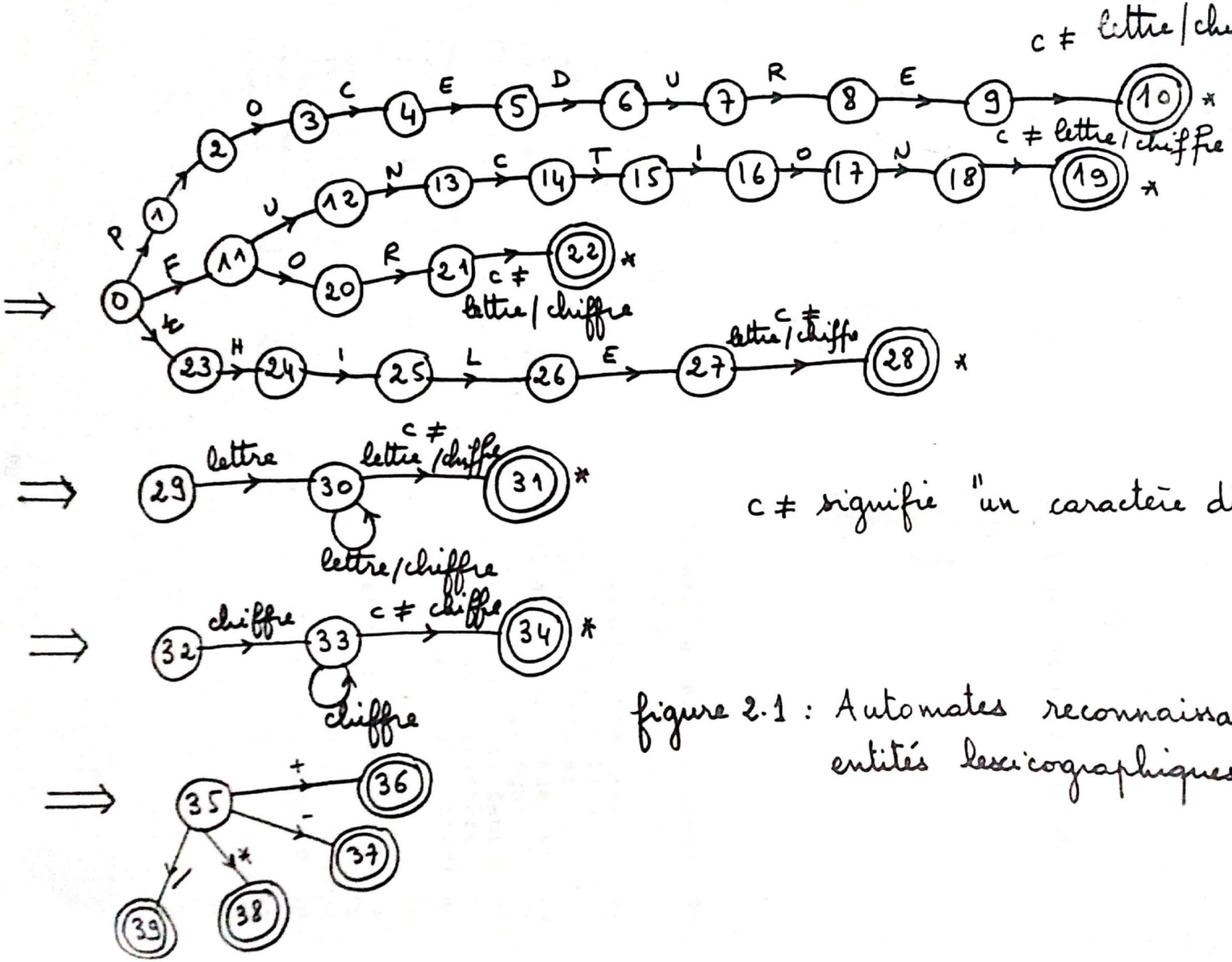
Les automates respectifs équivalents à ses expressions régulieres sont montrés sur la figure 2.1.

Pour reconnaitre le mot clé FOR par exemple, le simulateur passe de l'etat 0 à l'état 11, ensuite de l'état 11 à l'état 20, ensuite de 20 à 21 et enfin de 21 à 22. l'état 22 étant un état final, la transition de 21 a 22 s'éffectue avec un caractère différent d'un caractère alphanumerique. Ce caractère ne fait pas partie de l'entité FOR. Il faut donc réajuster le pointeur début entité d'un caractère en arrière. Tous les états finaux où il faut retirer un caractère sont indiqués par le signe '*'.

Pour reconnaitre l'identificateur PROC par exemple, le simulateur passe dans les états 0,1,2,3 et 4. En s'apercevant que 4 n'est pas un état final, il retire tous les caractères de la chaine "PROC" et recommence la simulation à partir de l'état 29.

2.2.1 Implémentation De L'automate.

Nous implémentons chaque état de l'automate par un programme. Si nous disposons de l'instruction CASE, les programmes correspondant respectivement aux états 0 et 1 peuvent être comme suit:



c ≠ signifie "un caractère différent de"

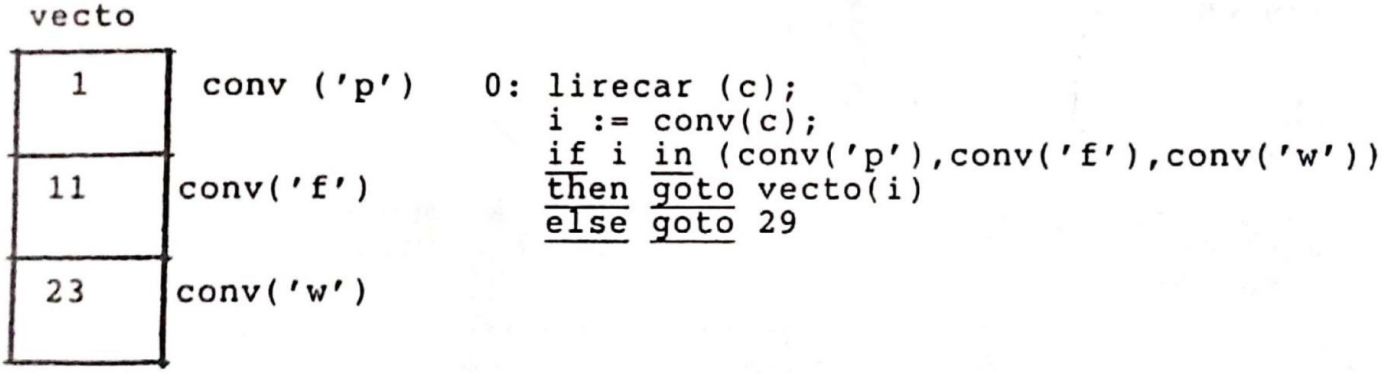
figure 2.1 : Automates reconnaissant des entités lexicographiques.

```
0: lirecar(c);
   if c in ('p','f','w') then
   case c of
   "p":  goto 1
   "f":  goto 11
   "w":  goto 23
  else   goto 29

1: lirecar(c);
  if c = 'r' then goto 2
  else begin retirer(c)
             goto 29
       end
```

La procédure lirecar fournit le caractère courant de la chaine et la procédure retirer fait reculer le pointeur début entité d'un caractère en arrière.

Dans le cas où nous ne disposons pas d'une instruction équivalente à l'instruction case, nous pouvons implémenter cette dernière en utilisant une structure de tableau comme suit:



Conv est une procédure qui convertit c en un entier i. vecto est un vecteur contenant les états de transition à partir de l'état 0.

2.3 IMPLEMENTATION EFFICACE DES ANALYSEURS LEXICOGRAPHIQUES.

Une procédure à suivre pour implémenter un analyseur lexicographique pour un langage donné pourrait se résumer comme suit:

1-Spécifier chaque type d'entité lexicale à l'aide d'une expression régulière.

2-Convertir chaque expression régulière obtenue en 1 en un automate d'états finis.

3-Construire l'automate union de tous les automates de l'étape 2.

4-Rendre l'automate de l'étape 3 déterministe.

5-Réduire au minimum, le nombre des états de l'automate de l'étape 4.

6-Implémenter l'automate obtenu à l'étape 5.

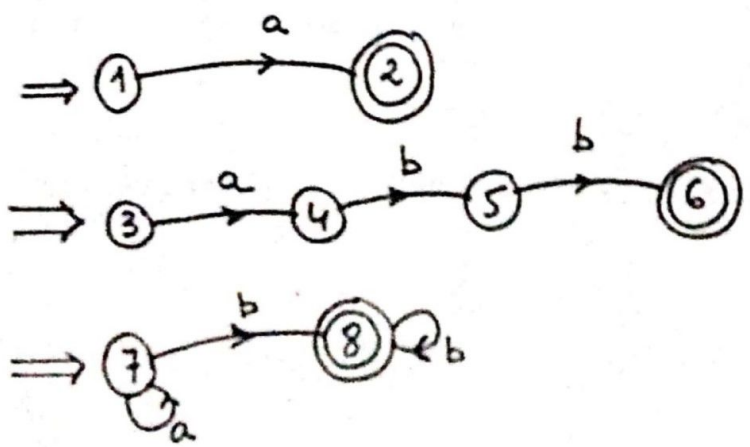
Appliquons cette procédure sur l'exemple suivant:

soit à projeter un analyseur lexicographique qui reconnait les entités

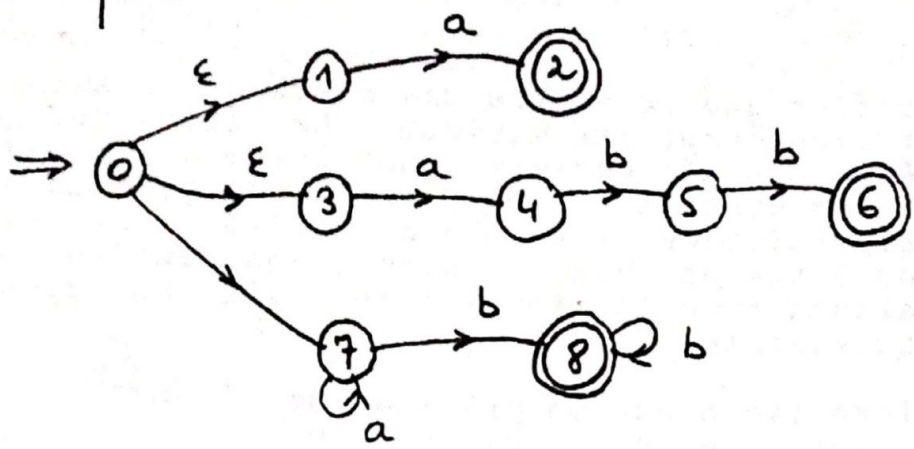
a
abb
a*b+

La première étape de la procédure est à ignorer puisque les entités lexicales sont exprimées à l'aide d'expressions régulières.

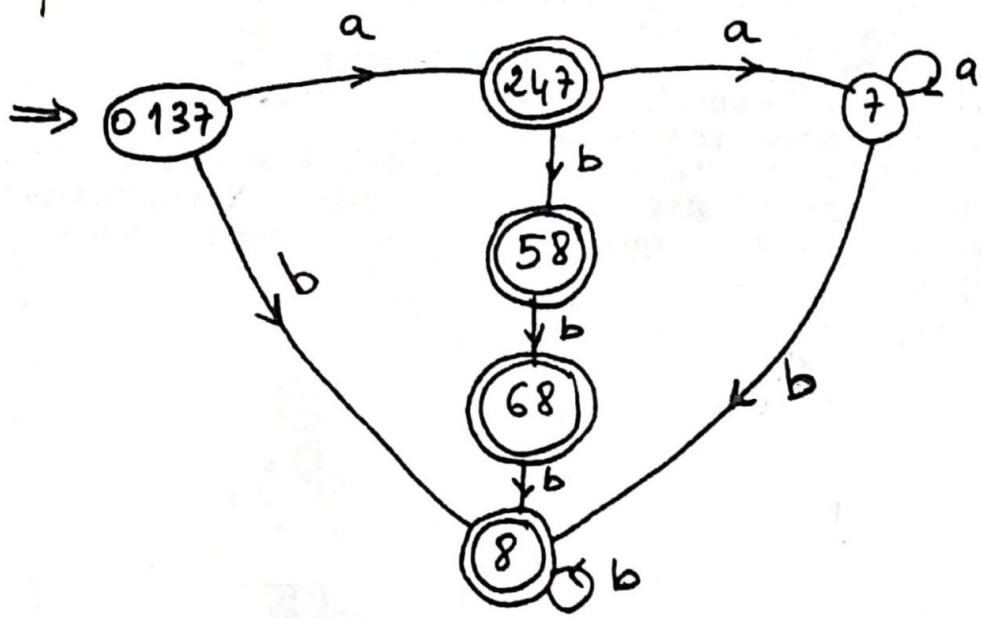
étape 2.



étape 3



étape 4



état	a	b	token détecté
0137	247	8	aucun
247	7	58	a
8	-	8	a* b+
7	7	8	aucun
58	-	68	a* b+
68	-	8	abb

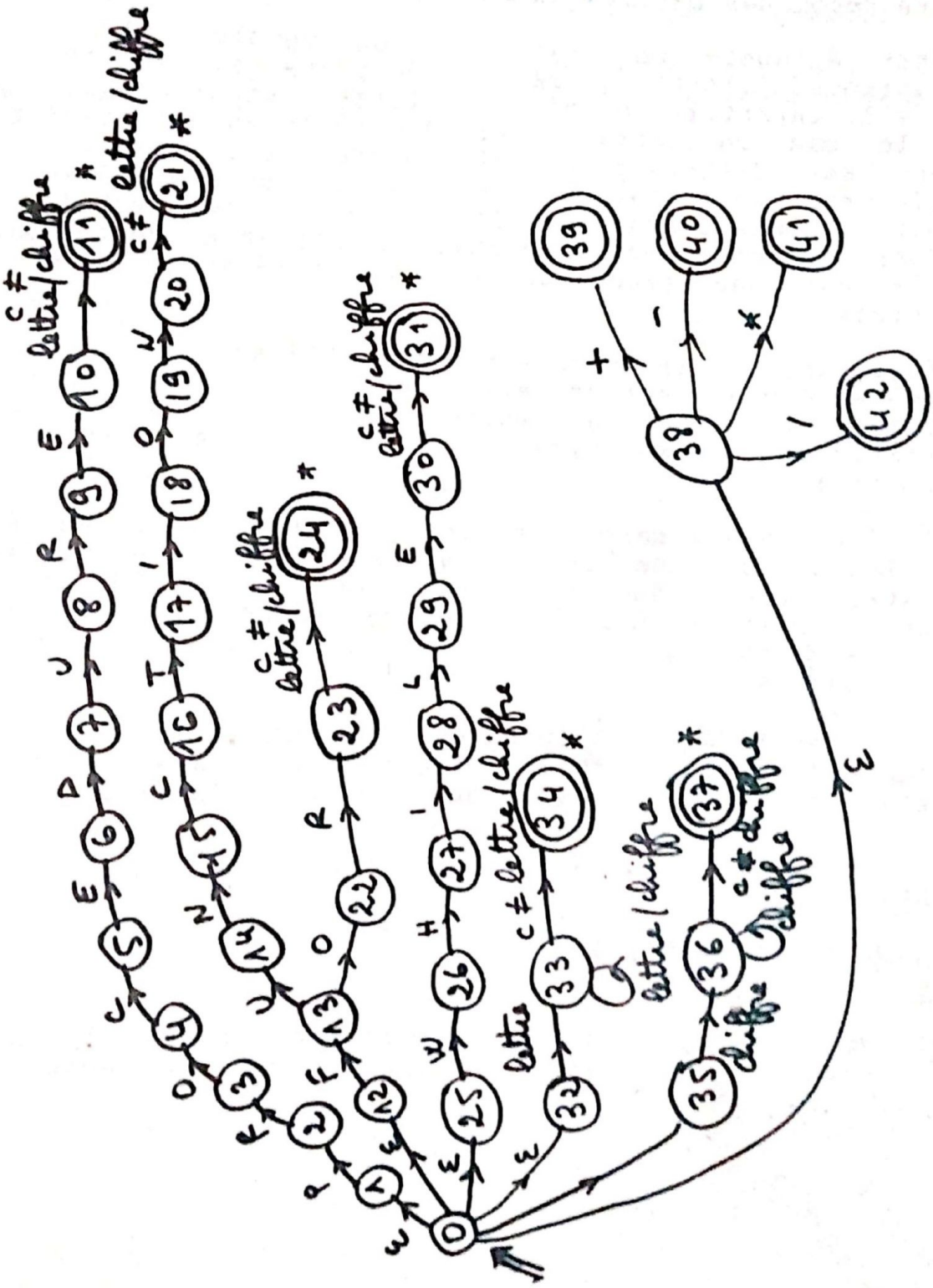
etape.5

Nous pouvons vérifier que le nombre des états de l'automate obtenu à la quatrième étape est minimum. Les états qui sont susceptibles d'être groupés ensemble sont les états (0137 et 7) d'une part et les états(8 et 58) d'une autre part car ils reconnaissent respectivement les mêmes entités lexicales. Mais puisque les états de chaque groupe transitent vers des états non équivalents avec la même lettre, ils ne peuvent donc pas être équivalents.

Remarquons que l'entité a est un préfixe de l'entité abb. L'analyseur lexicographique qui est un simulateur de l'automate ci-dessus pourra reconnaitre soit a soit abb dans le cas où la chaine abb lui est présentée. Cette ambiguité est généralement relevée en faveur de la plus longue chaine.

Un autre problème susceptible de se poser est que la chaine abb pourrait être reconnue soit par l'expression régulière abb soit par l'expression régulière a*b+. Ce conflit est résolu en faveur de l'expression régulière qui apparait en premier. Ou autrement dit, c'est l'ordre d'occurence des expressions régulières qui intervient pour résoudre ce conflit.

Figure 2.2: Union des automates.



Exemple

Considérons l'exemple de la deuxième section, l'automate non déterministe qui reconnait toutes les entités lexicographiques est montré sur la figure 2.2. Un automate déterministe équivalent au précédent est celui de la figure 2.3. Cet automate possède un nombre minimum d'états car chaque état final reconnait une chaine différente des autres chaines reconnues par les autres états finaux.

Un état étiqueté par '#' joue un double rôle. En l'occurrence, l'état 1 permet de transiter vers l'état 2 lorsque le caractère lu est la lettre alphabétique 'r'. Dans le cas contraire c'est à dire lorsque le caractère courant est différent de la lettre 'r', l'état 1 se transformera en l'état 25 ou en d'autres termes l'état 1 se comportera comme l'état 25. L'introduction du symbole '#' et donc de cette double fonction de l'état étiqueté par ce symbole est une façon de relever l'indéterminisme de l'automate.

En effet, lorsque le processus est à l'état 1, et si le caractère courant est un caractère autre que la lettre 'r', l'entité prochaine à former ne peut être qu'un identificateur puisqu'elle commence par une lettre(la lettre 'r').

L'état 9 est aussi marqué par le symbole '#'. Si en étant dans l'état 9, une lettre alphabétique ou un chiffre se présente, l'état 9 jouera le rôle de l'état 25 afin de donner la priorité à la plus longue chaine à être reconnue. En conséquence procedure1 sera considéré comme un identificateur.

Nous pouvons indiquer l'opérateur lookahead noté '/' dans une expression régulière en l'introduisant dans l'expression régulière juste après la fin du token à reconnaitre.

exemple.

IF / ({identificateur,}*).

Au niveau de l'implémentation, nous considérons le symbole '/' comme la chaine vide. Nous poursuivons la

reconnaissance de la suite de l'expression mais avec le pointeur lookahead. Une fois l'expression reconnue, le pointeur début entité est réajusté et est placé après le caractère '/'.