POLYTECH
LILLE

Dossier projet langage et traducteur

Par :

Laplace Jordan

et

Sellaï Yanis

Introduction

Afin de nous forger au métier d'ingénieur, des projets sont organisés en marge des cours magistraux et des travaux pratiques ayant pour but de vous préparer à la participation ou la direction d'un projet. Ce projet est organisé autour du cours Langages et Traducteurs afin de mettre en pratique les bases théoriques et pratiques acquises en cours et travaux pratiques.
En informatique, afin d'éviter les difficultés du "langage machine, nous avons recours à des traducteurs afin d'écrire un programme dans un certain langage qui pourra être reconnu lors de la compilation de ce dernier. Chaque langage admet des règles décrivant une structure syntaxique. Le but de ce projet est de réaliser un compilateur d'automates finis appelé scc (Start Chart Compiler) permettant de reconnaître un sous-ensemble de balise SCXML (start chart XML) permettant d'écrire une machine d'état à l'aide de nos compétences acquises en Lex et Accent lors des TP.
Tout d'abord, une première version du compilateur est réalisée afin de vérifier si un fichier texte est un document XML. À partir de cette première version, nous réalisons une seconde version permettant d'afficher son état initial, la liste de ses états en précisant pour chacun d'eux son nom, ses arcs, son nombre d'arcs et si c'est un état final et finalement son nombre total d'états, son nombre total d'états finaux et son nombre d'arcs. Toujours à partir de la première version, une troisième est créée pour vérifier plusieurs règles.

1. Version 1

1.1. Contrainte de la grammaire

Le but de cette première version est de programmer une grammaire vérifiant si un document XML donné définit un automate. Nous allons détailler dans cette partie les contraintes d'écriture d'un automate en XML. Nos choix de contrainte ont été décidé en fonction des consignes données et des contraintes liées au langage xml.

1.1.1. Contrainte du prologue

Un document SCXML dispose d'un prologue toujours égale à **<?xml version="1.0" encoding="UTF-8" ?>**. Nous avons choisit d'appliquer une contrainte supplémentaire au prologue.

1.1.2. Contrainte d'ouverture de balise

Lors d'une ouverture de balise, aucun espace n'est autorisé entre "<" et "transition" par exemple. Ainsi, le chevron ouvrant est contigu au nom de la balise. (voir les terminaux oBALISE_SCXML, oBALISE_STATE, oBALISE_TRANSITION et ENTETE).

1.1.3. Contrainte sur l'url

l'attribut VAL_REF doit toujours être égale à : "http://www.w3.org/2005/07/scxml"

1.1.4. Contrainte sur la valeur d'un identifiant

Le nom d'un état ou de l'étiquette d'un arc est constitué d'une lettre suivie d'une suite pouvant être vide, de chiffre, de lettres et des caractères _ et -. Ainsi, nous avons réalisé cette expression régulière qui résout cette contrainte "[a-zA-Z]([a-zA-Z0-9]|-|_)*". Nous mettons des guillemets pour ne pas avoir d'espace entre les valeurs des attributs.

1.1.5. Contrainte sur la valeur de l'état final

L'attribut final de la balise state peut contenir seulement les valeurs suivantes "true" ou "false". Si l'attribut final n'est pas défini dans la balise nous considérons que final est égale à "false". Dans les autres cas il s'agit d'une erreur de syntaxe.

1.1.6. Contrainte d'espace entre les éléments

Nous avons choisit d'appliquer une contrainte supplémentaire à notre grammaire. Celle ci permet à l'utilisateur d'espacer les différents éléments des balises. Par exemple, notre grammaire reconnaît les prologues écrits de la façon suivante : **<?xml version = "1.0" encoding = "UTF-8" ?>.** Cette contrainte est respectée pour toutes les balises de la grammaire. À noter que ceci nous force donc à définir chaque élément d'une balise séparément. Ce qui engendre un grand nombre de terminaux. Mais c'est le prix à payer pour respecter un maximum les caractéristiques du langage xml.

1.1.7. Contrainte sur l'ordre des attributs

Dans chacune des balises nous permettons à l'utilisateur de définir les attributs dans l'ordre qu'il souhaite. Par exemple l'expression **<?xml encoding = "UTF-8" version = "1.0" ?>** est reconnue par notre grammaire.

1.2. Détail sur les éléments de la grammaire

E : Non terminal reconnaissant le prologue d'un fichier SCXML puis le non terminal F.
F : Non terminal reconnaissant la balise scxml ainsi que le corps du fichier via le non terminal STA.

STA : Non terminal reconnaissant la balise state ainsi que les transitions liées aux états via le non terminal TRANS.
TRANS : Non terminal reconnaissant les transitions entres les états .
INIT : Non terminal reconnaissant l'état initial de l'automate.
ID : Non terminal reconnaissant le nom d'un état initial de l'automate.
EVENT : Non terminal reconnaissant la valeur associée à un arc de l'automate.
TARGET : Non terminal reconnaissant l'état final d'un arc de l'automate.

ENTETE : terminal représentant la balise ouvrante du prologue **<?xml**
ATT_VERSION : terminal représentant l'attribut **version** de la balise xml.
VALEUR_VERSION : terminal représentant la valeur de l'attribut version qui sera toujours **"1.0".**
ATT_ENCODING : Terminal représentant l'attribut **encoding** de la balise xml.
VALEUR_VERSION : Terminal représentant la valeur de l'attribut encoding qui sera toujours **"UTF-8"**.
fENTETE : Terminal représentant le chevron fermant du prologue **?>.**

oBALISE_SCXML : Terminal représentant la balise ouvrante scxml **<scxml**.
ATT_XMLNS : : Terminal représentant l'attribut **xmlns** de la balise scxml.
VALEUR_REF : Terminal représentant la valeur de l'attribut xmlns qui sera toujours **"http://www.w3.org/2005/07/scxml".**
ATT_INIT : Terminal représentan l'attribut **initial** de la balise scxml.
fBALISE_SCXML : Terminal représentant le début de la balise fermante scxml **</scxml;**

oBALISE_STATE : Terminal représentant la balise ouvrante state **<state.**
ATT_ID : Terminal représentant l'attribut **id** de la balise state.
ATT_FINAL : Terminal représentant l'attribut **final** de la balise state.
fBALISE_STATE : Terminal représentant début de la balise fermante state **</state.**

oBALISE_TRANSITION : Terminal représentant la balise ouvrante transition **<transition .**
ATT_EVENT: Terminal représentant l'attribut **event** de la balise transition.
ATT_TARGET: Terminal représentant l'attribut **target** de la balise transition.
fBALISE_TRANSITION : Terminal représentant le chevron fermant de la balise transition **/>.**

TRUE : Terminal représentant la valeur **"true"** de l'attribut final.
FALSE : Terminal représentant la valeur **"false"** de l'attribut final.
IDENTIFIANT : Terminal représentant la valeur **"[a-zA-Z]([a-zA-Z0-9]|-|_)*"** de l'attribut init, event ou target. Cette expression régulière permet que le nom d'un état ou de l'étiquette d'un arc soit constitué d'une lettre suivie d'une suite pouvant être vide de chiffres, de lettres et des caractères - et _.
BALISEf : Terminal représentant le chevron fermant des balises ouvrantes et fermantes state et scxml **>.**

EGALE : Terminal représentant le signe **=** placé après un attribut .

1.3. Grammaire

E → **<?xml version = "1.0" encoding = "UTF-8" ?>** F
| **<?xml encoding = "UTF-8" version = "1.0" ?>** F

F → **<scxml xmlns = "http://www.w3.org/2005/07/scxml"** INIT **>** STA **</scxml >**
| <scxml INIT **xmlns = "http://www.w3.org/2005/07/scxml" >** STA **</scxml >**

INIT → **initial =** IDENTIFIANT

STA → **<state** ID FINAL **>** TRANS **</state >**STA
| **<state** FINAL ID **>** TRANS **</state >** STA
| **ε**

ID → **id =** IDENTIFIANT

FINAL → **final = "true"**
| **final =** "**false"**
| **ε**

TRANS → **<transition** EVENT TARGET **/>** TRANS
| **<transition** TARGET EVENT **/>** TRANS
| **ε**

EVENT → **event =** IDENTIFIANT

TARGET →**target =** IDENTIFIANT

1.4. Tests effectués

1.4.1. Test du prologue

Les fichier “erreurUTF.xml” et “erreurVersion.xml” permettent de générer des erreurs de syntaxe à des mauvaises valeurs dans les attributs encoding et version du prologue.

1.4.2. Test ouverture de balise

Le fichier “erreurMauvaisOuvertureBalise.xml” génère une erreur liée à la mauvaise ouverture de la balise state.

1.4.3. Test url

Le fichier “erreurURL.xml” génère une erreur liée à la mauvaise valeur de l’url de la balise scxml.

1.4.4. Test sur la valeur de l'état final

Le fichier “erreurEtatFinal.xml” génère une erreur liée à la mauvaise valeur de l’attribut final de la balise state.

1.4.5. Test d’espace entre les éléments

Le fichier “valide_Espace_entre_les_éléments.xml” ne génère pas d’erreur liée au espacement entre les éléments d’une balise.

1.4.6. Test sur l’ordre des attributs

Le fichier “valideOrdreAtt.xml” ne génère pas d’erreur liée à l’ordre de déclaration des attributs .

1.4.7. Test sur la valeur d’un identifiant

Le fichier “erreurID.xml” génère une erreur liée à un identifiant invalide.

2. Version 2

2.1. Définitions des attributs utilisés

nous utilisons les attributs suivant :

- STA_eta, attribut synthétisé, associé au non-terminal STA : Entier - nombre d'etats totaux.
- STA_fin, attribut synthétisé, associé au non-terminal STA : Entier - nombre d'états finaux totaux.
- STA_arcs, attribut synthétisé, associé au non-terminal STA : Entier - nombre d'arcs totaux.
- val, attribut synthétisé , associé au terminal IDENTIFIANT : Chaine de caractere - nom d'un etat, d'une étiquette ou d'un arc selon l'emplacement de IDENTIFIANT dans la grammaire.
- bool , attribut synthétisé , associé au non-terminal FINAL : Entier - entier utilisée comme un booléen. Si bool = 1 alors l'état est final si non l'etat n'est pas final.

2.2. Type défini

Dans cette version utilisons un type chaîne de caractère (char * id) que nous avons défini le fichier yystype.h . Son objectif est de contenir les noms des arcs, etats ou étiquettes reconnues. Ce type est donc associé au terminal IDENTIFIANT.

Remarque : dans le fichier v2.lex on peut remarquer les lignes de code suivantes.

```
{
// on alloue à yylval.id la taille mémoire nécessaire pour stocker l'identifiant reconnu
yylval.id = (char * ) malloc (strlen(yytext)*sizeof(char)+1 ) ;

// yytext = "ID" , l'objectif ici est d'enlever les guillemets, donc nous copions yytext sans le
// premier caractère qui est obligatoirement un guillemet
strcpy(yylval.id , yytext + 1 ) ;

// nous enlevons le guillemet final
yylval.id[strlen(yytext)-2] = '\0' ;

 return IDENTIFIANT;
 }
```

2.3. Schéma de traduction dirigée par la syntaxe

grammaire	schéma de traduction dirigés par la syntaxe
E → **<?xml version = "1.0" encoding = "UTF-8" ?>** F	E → **<?xml version = "1.0" encoding = "UTF-8" ?>** F
E → **<?xml encoding = "UTF-8" version = "1.0" ?>** F	E → **<?xml encoding = "UTF-8" version = "1.0" ?>** F
F → **<scxml xmlns="http://www.w3.org/2005/07/scxml"** INIT **>** STA **</scxml >**	F → **<scxml xmlns="http://www.w3.org/2005/07/scxml"** INIT **>** STA **</scxml >** { printf("\n Nombre total d'etats : %d \n Nombre d'etats finaux : %d \n Nombre total d'arc(s) : %d \n",STA.STA_eta, STA.STA_fin, STA.STA_arcs); }
F → INIT **<scxml xmlns="http://www.w3.org/2005/07/scxml" >** STA **</scxml >**	F → **<scxml** INIT **xmlns="http://www.w3.org/2005/07/scxml" >** STA **</scxml >** { printf("\n Nombre total d'etats : %d \n Nombre d'etats finaux : %d \n Nombre total d'arc(s) : %d \n",STA.STA_eta, STA.STA_fin, STA.STA_arcs); }
INIT→ **initial = IDENTIFIANT**	INIT → **initial = IDENTIFIANT** {printf("Etat initial : %s",**IDENTIFIANT**.id);}
STA → **<state** ID FINAL **>** TRANS **</state >**STA	STA → **<state** ID FINAL { if(FINAL.bool) printf(": Etat final"); } **>** TRANS **</state >** {printf("\n\t Nombre d'arc(s) : %d \n ",TRANS.TRA_nbArc);} STA1 {STA.STA_arcs = STA1.STA_arcs + TRANS.TRA_nbArc ; STA.STA_eta = STA1.STA_eta + 1 ; if (bool) STA.STA_fin = STA1.STA_fin +1 ; else STA.STA_fin = STA1.STA_fin ; }
STA → **<state** FINAL ID **>**TRANS **</state >** STA1	STA → **<state** FINAL ID { if(FINAL.bool) printf(": Etat final"); }
	> TRANS **</state >** {printf("\n\t Nombre d'arc(s) : %d \n ",TRANS.TRA_nbArc);} STA1 {STA.STA_arcs = STA1.STA_arcs + TRANS.TRA_nbArc ; STA.STA_eta = STA1.STA_eta + 1 ; if (bool) STA.STA_fin = STA1.STA_fin +1 ; else STA.STA_fin = STA1.STA_fin ; }
STA →**ε**	STA →**ε** {STA.STA_eta = 0 ; STA.STA_fin = 0 ; STA.STA_arcs = 0 ; }
ID → **id = IDENTIFIANT**	ID → **id = IDENTIFIANT** {printf("\n\n\t Etat %s",**IDENTIFIANT**.id);}
FINAL → **final = "true"**	FINAL → **final = "true"** {FINAL.bool = 1 ; }
FINAL →**final = "false"**	FINAL →**final = "false"** {FINAL.bool = 0 ; }
FINAL →**ε**	FINAL →**ε** {FINAL.bool = 0 ; }
TRANS→ **<transition** EVENT TARGET **/>** TRANS1	TRANS→ **<transition** EVENT TARGET **/>**TRANS1 {TRANS.TRA_nbArc = TRANS1.TRA_nbArc +1;}
TRANS→ **<transition** TARGET EVENT **/>** TRANS1	TRANS→ **<transition** TARGET EVENT **/>** TRANS1 {TRANS.TRA_nbArc = TRANS1.TRA_nbArc +1;}
TRANS→**ε**	TRANS→**ε** {TRANS.TRA_nbArc = 0 ;}
EVENT→ **event = IDENTIFIANT**	EVENT→ **event = IDENTIFIANT** {printf("\n\t\tArc : Etiquette %s - Etat ",**IDENTIFIANT**.id);}
TARGET → **target = IDENTIFIANT**	TARGET → **target = IDENTIFIANT** {printf("%s",**IDENTIFIANT**.id);}

2.4. tests effectués

Dans cette partie nous avons effectué les même test que dans la version 1. Cette partie ne nécessite aucune vérification supplémentaire.

3. Version 3

3.1. Définitions des attributs utilisés

Nous utilisons les attributs suivants :

- INIT_nom, attribut synthétisé, associé au non-terminal INIT : ChaineC (char chaineC [50]) - nom de l'état initial
- id_nom, attribut synthétisé, associé au non-terminal ID : ChaineC (char chaineC [50]) - nom de l'état en cours de traitement.
- nomEtat, attribut hérité , associé au non-terminal TRANS : ChaineC (char chaineC [50]) - nom de l'état d'où partent les arcs reconnus.
- EV_val, attribut synthétisé, associé au non-terminal EVENT : ChaineC (char chaineC [50]) - nom de l'événement reconnu.
- TAR_val, attribut synthétisé, associé au non-terminal TARGET : ChaineC (char chaineC [50]) - nom de l'état qui est la destination de l'arc en cours de traitement.
- val, attribut synthétisé , associé au terminal IDENTIFIANT : Chaine de caractere - nom d'un etat, d'une étiquette ou d'un arc selon l'emplacement de IDENTIFIANT dans la grammaire
- bool , attribut synthétisé , associé au non-terminal FINAL : Entier - entier utilisé comme un booléen. Si bool = 1 alors l'état est final sinon l'etat n'est pas final.

3.2. Type défini

Dans cette version utilisons un type chaîne de caractère (char * id) que nous avons défini le fichier yystype.h . Son objectif est de contenir les noms des arcs, etats ou étiquettes reconnues. Ce type est donc associé au terminal IDENTIFIANT.

Remarque : dans le fichier v2.lex on peut remarquer les lignes de code suivantes.

```
{
// on alloue à yylval.id la taille mémoire nécessaire pour stocker l'identifiant reconnu
yylval.id = (char * ) malloc (strlen(yytext)*sizeof(char)+1 ) ;

// yytext = "ID" , l'objectif ici est d'enlever les guillemets, donc nous copions yytext sans le
// premier caractère qui est obligatoirement un guillemet
strcpy(yylval.id , yytext + 1 ) ;

// nous enlevons le guillemet final
yylval.id[strlen(yytext)-2] = '\0' ;

 return IDENTIFIANT;
 }
```

3.3. Schéma de traduction dirigée par la syntaxe

grammaire	schéma de traduction dirigés par la syntaxe
E → **<?xml version = "1.0" encoding = "UTF-8" ?>** F	E → **<?xml version = "1.0" encoding = "UTF-8" ?>** { initTable(&tabE);} F { afficherTable(tabE);}
E → **<?xml encoding = "UTF-8" version = "1.0" encoding = "UTF-8" ?>** F	E → **<?xml encoding = "UTF-8" version = "1.0" ?>** { initTable(&tabE);} F { afficherTable(tabE);}
F → **<scxml xmlns="**http://www.w3.org/2005/07/scxml**"** INIT **>** STA **</scxml >**	F → **<scxml xmlns="**http://www.w3.org/2005/07/scxml**"** INIT **>**STA **</scxml >** {definirEtatInitial(&tabE, INIT.INIT_nom);
F → **<scxml** INIT **xmlns="**http://www.w3.org/2005/07/scxml**" >** STA **</scxml >**	F → **<scxml** INIT **xmlns="**http://www.w3.org/2005/07/scxml**" >**STA **</scxml >** {definirEtatInitial(&tabE, INIT.INIT_nom);
INIT → **initial = IDENTIFIANT**	INIT → **initial = IDENTIFIANT** { strcpy(INIT.INIT_nom, **IDENTIFIANT**.id) ;}
STA → **<state** ID FINAL **>** TRANS **</state >**STA	STA → **<state** ID FINAL **>** { insererEtat(&tabE , ,ID.id_nom) ; strcpy(TRANS.nomEtat ,ID.id_nom) ;if(FINAL.bool) definirEtatFinal(&tabE , ,ID.id_nom) ; } TRANS **</state >** STA
STA → **<state** FINAL ID **>** TRANS **</state>** STA	STA → **<state** FINAL < ID BALISEf { insererEtat(&tabE , ,ID.id_nom) ; strcpy(TRANS.nomEtat, ,ID.id_nom) ; if(FINAL.bool) definirEtatFinal(&tabE , ,ID.id_nom) ;} TRANS **</state >** STA
STA →**ε**	STA →**ε**
ID → **id = IDENTIFIANT**	ID → **id = IDENTIFIANT** { strcpy(ID.id nom , **IDENTIFIANT**.id) ;}
FINAL → **final = "true"**	FINAL → **final = "true"** {FINAL.bool = 1 ; }
FINAL →**final = "false"**	FINAL →**final = "false"** {FINAL.bool = 0 ; }
FINAL → **ε**	FINAL →**ε** {FINAL.bool = 0 ; }
FINAL → **final = IDENTIFIANT**	FINAL → **final = IDENTIFIANT** {yyerror("l'attribut final de la balise state doit avoir pour valeur soit true, soit false");}
TRANS→ **<transition** EVENT TARGET **/>** TRANS1	TRANS→ **<transition** EVENT TARGET **/>** {insererArc(&tabE , TRANS.nomEtat , TARGET.TAR_val ,EVENT.EV_val); strcpy(TRANS1.nomEtat1 ,TRANS.nomEtat) ; } TRANS1
TRANS→ **<transition** TARGET EVENT **/>** TRANS1	TRANS→ **<transition** TARGET EVENT {insererArc(&tabE , TRANS.nomEtat , TARGET.TAR_val ,EVENT.EV_val); strcpy(TRANS1.nomEtat1 ,TRANS.nomEtat) ; } **/>**TRANS1
TRANS→**ε**	TRANS→**ε**
EVENT→ **event = IDENTIFIANT**	EVENT → **event = IDENTIFIANT** { strcpy(EVENT.EV_VAL, **IDENTIFIANT**.id) ;}
TARGET → **target = IDENTIFIANT**	TARGET → **target = IDENTIFIANT** { strcpy(TARGET.TAR_val, **IDENTIFIANT**.id) ;}

3.4. Tests effectuées

3.4.1. Test sur des arcs identiques

Deux arcs ne doivent pas avoir les mêmes états de départ, état d'arrivée et étiquette. Ainsi, l'ajout d'un arc se fait à l'aide de l'action insererArc présente dans etat.c. Nous utilisons cette action dans la balise TRANS pour pouvoir vérifier que, dans un même état, chaque arc est unique. Si deux arcs sont identiques dans un même état, cette action va écrire le message d'erreur suivant : "cet arc existe déjà". De plus, s'il n'y a plus de places (dans le vecteur arc), un message d'erreur est renvoyé. Le fichier "erreurArcIdentique.xml" permet de générer une erreur liée à des arcs de même nom.

3.4.2. Test sur les états finaux

L'attribut final de la balise state doit avoir pour valeur soit true, soit false. Ainsi, dans notre grammaire, nous l'imposons en ajoutant "| ATT_FINAL EGALE IDENTIFIANT {yyerror("l'attribut final de la balise state doit avoir pour valeur soit true, soit false");}". Ainsi, si l'attribut final de la balise state n'a pas les valeurs True ou False, nous renvoyons un message d'erreur. L'erreur peut prendre deux formes. Soit une erreur de syntaxe si la valeur de l'attribut n'est pas sous la forme d'un identifiant. Ou alors une erreur plus précise si la variable ressemble à un identifiant. Le fichier "erreurEtatFinal.xml" permet de générer une erreur liée à une mauvaise valeur pour un état final.

3.4.3. Test sur l'état initial

L'attribut initial de la balise scxml doit référencer un attribut id d'une balise state. Ainsi dans dans notre grammaire, nous utilisons l'action definirEtatInitial qui va vérifier si le nom d'état proposé référencie bien un attribut d'une balise state. Si cet état n'existe pas, le message d'erreur suivant est affiché : "Etat initial non défini". Le fichier "erreurEtatInit.xml" permet de générer une erreur liée à une mauvaise valeur pour un état initial.

3.4.4. Test sur l'unicité des états

Deux états ne doivent pas avoir le même nom. Ainsi, l'ajout d'un arc se fait à l'aide de l'action insererEtat présente dans etat.c. Nous utilisons cette action dans la balise STA pour pouvoir vérifier que, dans un même état, chaque arc est unique. Si deux arcs sont identiques dans un même état, cette action va écrire le message d'erreur suivant : "cet état existe déjà". De plus, s'il n'y a plus de places (dans le vecteur arc), un message d'erreur est renvoyé. Le fichier "mêmeNomEtats.xml" permet de générer une erreur liée à une mauvaise valeur pour des noms d'états.

4. Conclusion

Au terme de ce projet, nous avons réussi à créer un compilateur d'automates en suivant les différentes règles imposées dans les trois versions.

Dans un projet plus ambitieux avec plus d'attributs par balises, notre grammaire aurait été différente. En effet il aurait été intéressant de créer un élément de grammaire juste pour les attributs d'une balise. Mais avec un maximum de deux attributs par balises pour le projet. opter pour ce choix aurait complexifié la grammaire, la rendant moins lisible et pas plus efficace.

Outre l'aspect technique, ce projet nous a permis de mettre en pratique ce que nous avons vu en cours et en travaux pratiques. Ce projet est très pédagogique car il s'est dirigé en binôme, ce qui a créer une coopération même si la maîtrise n'est pas forcément pareille chez chacun de nous. Les tuteurs de ce projet nous ont été très utiles de part leur aide. Cela nous a permis de corriger des erreurs qui pouvaient avoir un effet beaucoup plus important par la suite.

5. Code des versions

5.1. Version 1 lex

```
%{
/* Fichier exAtt.lex */
#include "yygrammar.h"

char err[20]; /* Chaine de caracteres pour les erreurs de syntaxe */
%}

/* Definition de macros */
chiffre         [0-9]
separateur      [ \t]
lettre                  [a-zA-Z]
id                      [a-zA-Z0-9]|-|_

%%
```

```
"<?xml"
                return ENTETE;    /* Indique au parser que l'entête est reconnu */
"version"
                return ATT_VERSION; /* Indique au parser qu'un attribut version est
reconnu */
"\"1.0\""
                return VALEUR_VERSION; /* Indique au parser la version de l'attribut
version est reconnu */
"encoding"
                return ATT_ENCODING; /* Indique au parser qu'un attribut encoding
est reconnu */
"\"UTF-8\""
                return ATT_UTF; /* Indique au parser le type d'encodage de l'attribut
encoding est reconnu */


"<scxml"
                return oBALISE_SCXML; /* Indique au parser qu'une balise ouvrante
scxml  est reconnu  */
"xmlns"
                return ATT_XMLNS; /* Indique au parser qu'un attribut xmlns est
reconnu */
"\"http://www.w3.org/2005/07/scxml\""                                    return
VALEUR_REF; /* Indique au parser que la référence de l'attribut xmlns est reconnu
*/
"initial"
        return ATT_INIT; /* Indique au parser qu'un attribut initial est reconnu */
"</scxml"
                return fBALISE_SCXML; /* Indique au parser qu'une balise fermante
scxml   est reconnu  */

"<state"
                return oBALISE_STATE; /* Indique au parser qu'une balise ouvrante
state   est reconnu  */
"id"
                return ATT_ID; /* Indique au parser qu'un attribut id est reconnu */
"final"
                return ATT_FINAL; /* Indique au parser qu'un attribut final est reconnu
*/
"</state"
                return fBALISE_STATE; /* Indique au parser qu'une balise fermante
state   est reconnu */

"<transition"
        return oBALISE_TRANSITION ;   /* Indique au parser qu'une balise ouvrante
transition est reconnu */
```

```
"event"
                return ATT_EVENT;    /* Indique au parser que l'attribut event est
reconnu */
"target"
        return ATT_TARGET;    /* Indique au parser que l'attribut target est reconnu */

"\"true\""
                return TRUE;  /* Indique au parser qu'un boolean TRUE est reconnu */
"\"false\""
                return FALSE;  /* Indique au parser qu'un boolean  FALSE est
reconnu */
\"{lettre}{id}*\"
return IDENTIFIANT;     /* Indique au parser qu'un identifiant  est reconnu */


"?>"
                return fENTETE; /* Indique au parser qu'une balise fermante  xml est
reconnu */
"/>"
                return fBALISE_TRANSITION; /* Indique au parser qu'une balise
fermante transition est reconnu */
">"
                        return BALISEf; /* Indique au parser qu'une fin de balise
simple  est reconnu   (>)*/
"="
                        return EGALE; /* Indique au parser qu'un signe égale  est
reconnu */

{separateur}+                                                           ;
/* Elimination des espaces */
\n                                                                      yypos++;
/* Compte le nombre de lignes du fichier source */
.                                                                       {
sprintf(err,"Mauvais caractere %c",yytext[0]);

yyerror(err);    /* Generation d'une erreur de syntaxe */
                                                                        }
%%
```

5.2. Version 1 acc

```
/* Fichier exAtt.acc */

%prelude{ /* Code C */
```

```
  /* Inclusion de bibliotheques C */
  #include <stdio.h>
  #include <malloc.h>

  /* Action de fin d analyse */
  void fin_analyse(){
      printf("Syntaxe correcte\n");
  }
}

/* Declaration des tokens */
%token ENTETE,
ATT_VERSION ,
ATT_ENCODING,
VALEUR_VERSION,
ATT_UTF,
VALEUR_REF,
fENTETE,
oBALISE_SCXML ,
ATT_XMLNS,
ATT_INIT,
fBALISE_SCXML ,
oBALISE_STATE,
ATT_ID,
ATT_FINAL,
fBALISE_STATE ,
oBALISE_TRANSITION,
ATT_EVENT,
ATT_TARGET,
fBALISE_TRANSITION,
BALISEf,
BOOLEAN,
EGALE,
IDENTIFIANT,
TRUE,
FALSE;

// Grammaire
E :  ENTETE ATT_VERSION EGALE VALEUR_VERSION ATT_ENCODING EGALE
ATT_UTF fENTETE F { fin_analyse(); }
  | ENTETE  ATT_ENCODING EGALE ATT_UTF ATT_VERSION EGALE
VALEUR_VERSION fENTETE F { fin_analyse(); }
;

F : oBALISE_SCXML ATT_XMLNS EGALE VALEUR_REF INIT BALISEf STA
fBALISE_SCXML BALISEf
```

```
  | oBALISE_SCXML  INIT ATT_XMLNS EGALE VALEUR_REF BALISEf STA
fBALISE_SCXML BALISEf
;

INIT : ATT_INIT EGALE  IDENTIFIANT
;

STA : oBALISE_STATE  ID  FINAL BALISEf TRANS  fBALISE_STATE  BALISEf
STA
    | oBALISE_STATE  FINAL ID BALISEf TRANS  fBALISE_STATE  BALISEf STA
    |
;

ID : ATT_ID EGALE  IDENTIFIANT
;
FINAL : ATT_FINAL EGALE  TRUE
      | ATT_FINAL EGALE  FALSE
      |
;

TRANS : oBALISE_TRANSITION  EVENT TARGET fBALISE_TRANSITION TRANS
  |oBALISE_TRANSITION  TARGET EVENT  fBALISE_TRANSITION TRANS
  |
;

EVENT : ATT_EVENT EGALE  IDENTIFIANT
;

TARGET : ATT_TARGET EGALE  IDENTIFIANT
;
```

5.3. Version 2 lex

```
%{
/* Fichier exAtt.lex */
#include "yystype.h"
#include "yygrammar.h"

char err[20]; /* Chaine de caracteres pour les erreurs de syntaxe */
%}

/* Definition de macros */
chiffre         [0-9]
separateur      [ \t]
lettre                  [a-zA-Z]
```

```
id                      [a-zA-Z0-9]|-|_

%%



"<?xml"
                return ENTETE;    /* Indique au parser que l'entête est reconnu */
"version"
                return ATT_VERSION; /* Indique au parser qu'un attribut version est
reconnu */
"\"1.0\""
                return VALEUR_VERSION; /* Indique au parser la version de l'attribut
version est reconnu */
"encoding"
                return ATT_ENCODING; /* Indique au parser qu'un attribut encoding
est reconnu */
"\"UTF-8\""
                return ATT_UTF; /* Indique au parser le type d'encodage de l'attribut
encoding est reconnu */


"<scxml"
                return oBALISE_SCXML; /* Indique au parser qu'une balise ouvrante
scxml  est reconnu  */
"xmlns"
                return ATT_XMLNS; /* Indique au parser qu'un attribut xmlns est
reconnu */
"\"http://www.w3.org/2005/07/scxml\""                                   return
VALEUR_REF; /* Indique au parser que la référence de l'attribut xmlns est reconnu
*/
"initial"
        return ATT_INIT; /* Indique au parser qu'un attribut initial est reconnu */
"</scxml"
                return fBALISE_SCXML; /* Indique au parser qu'une balise fermante
scxml   est reconnu  */

"<state"
                return oBALISE_STATE; /* Indique au parser qu'une balise ouvrante
state   est reconnu  */
"id"
                return ATT_ID; /* Indique au parser qu'un attribut id est reconnu */
"final"
                return ATT_FINAL; /* Indique au parser qu'un attribut final est reconnu
*/
```

```
"</state"
                return fBALISE_STATE; /* Indique au parser qu'une balise fermante
state   est reconnu */

"<transition"
        return oBALISE_TRANSITION ;   /* Indique au parser qu'une balise ouvrante
transition est reconnu */
"event"
                return ATT_EVENT;    /* Indique au parser que l'attribut event est
reconnu */
"target"
        return ATT_TARGET;    /* Indique au parser que l'attribut target est reconnu */

"\"true\""
                return TRUE;  /* Indique au parser qu un boolean TRUE est reconnu */
"\"false\""
                return FALSE;  /* Indique au parser qu un boolean  FALSE est
reconnu */
\"{lettre}{id}*\"                                                              {
yylval.id = (char * ) malloc (strlen(yytext)*sizeof(char)+1 ) ; strcpy(yylval.id , yytext + 1
) ;yylval.id[strlen(yytext)-2] = '\0' ;   return IDENTIFIANT; }    /* Indique au parser qu
un identifiant  est reconnu */


"?>"
                return fENTETE; /* Indique au parser qu'une balise fermante  xml est
reconnu */
"/>"
                return fBALISE_TRANSITION; /* Indique au parser qu'une balise
fermante transition est reconnu */
">"
                        return BALISEf; /* Indique au parser qu'une fin de balise
simple  est reconnu   (>)*/
"="
                        return EGALE; /* Indique au parser qu'un signe égale  est
reconnu */

{separateur}+                                                                  ;
/* Elimination des espaces */
\n                                                                     yypos++;
/* Compte le nombre de lignes du fichier source */
.                                                                      {
sprintf(err,"Mauvais caractere %c",yytext[0]);

yyerror(err);    /* Generation d'une erreur de syntaxe */
                                                                       }
```

%%

5.4. Version 2 acc

```
/* Fichier exAtt.acc */

%prelude{ /* Code C */
  /* Inclusion de bibliotheques C */
  #include <stdio.h>
  #include <malloc.h>
  #include "yystype.h"

  /* Action de fin d analyse */
  void fin_analyse(int nbEntiers){
      printf("Syntaxe correcte\n");
      printf("Nombre d'entier %d\n",nbEntiers);
  }
}

/* Declaration des tokens */
%token ENTETE,
ATT_VERSION ,
ATT_ENCODING,
VALEUR_VERSION,
ATT_UTF,
VALEUR_REF,
fENTETE,
oBALISE_SCXML ,
ATT_XMLNS,
ATT_INIT,
fBALISE_SCXML ,
oBALISE_STATE,
ATT_ID,
ATT_FINAL,
fBALISE_STATE ,
oBALISE_TRANSITION,
ATT_EVENT,
ATT_TARGET,
fBALISE_TRANSITION,
BALISEf,
BOOLEAN,
EGALE,
IDENTIFIANT,
TRUE,
```

```
FALSE;

// Grammaire
E :  ENTETE ATT_VERSION EGALE VALEUR_VERSION ATT_ENCODING EGALE
ATT_UTF fENTETE F
   |ENTETE ATT_ENCODING EGALE ATT_UTF ATT_VERSION EGALE
VALEUR_VERSION  fENTETE F
;

F : oBALISE_SCXML ATT_XMLNS EGALE VALEUR_REF  INIT BALISEf STA <STA_eta ,
STA_fin , STA_arcs>  fBALISE_SCXML BALISEf {printf("\n Nombre total d'etats : %d  \n
Nombre d'etats finaux : %d \n Nombre total d'arc(s) : %d \n",STA_eta,STA_fin,STA_arcs); }
  | oBALISE_SCXML  INIT ATT_XMLNS EGALE VALEUR_REF BALISEf STA <STA_eta ,
STA_fin , STA_arcs>  fBALISE_SCXML BALISEf {printf("\n Nombre total d'etats : %d  \n
Nombre d'etats finaux : %d \n Nombre total d'arc(s) : %d \n",STA_eta,STA_fin,STA_arcs); }
;

INIT : ATT_INIT EGALE  IDENTIFIANT <val>  {printf("Etat initial : %s",val.id);}
;

STA <%out int STA_eta , int STA_fin , int STA_arcs> : oBALISE_STATE  ID  FINAL<bool>  {
if(bool) printf(": Etat final"); } BALISEf TRANS <TRA_nbArc> fBALISE_STATE BALISEf
{printf("\n\t Nombre d'arc(s) : %d \n ",TRA_nbArc);}
STA <STA1_eta , STA1_fin , STA1_arcs> {*STA_arcs = STA1_arcs + TRA_nbArc ;
*STA_eta = STA1_eta + 1 ; if (bool) *STA_fin = STA1_fin  +1 ; else *STA_fin = STA1_fin  ;   }

   | oBALISE_STATE  FINAL <bool> ID  { if(bool) printf(": Etat final");  } BALISEf TRANS
<TRA_nbArc>  fBALISE_STATE BALISEf {printf("\n\t Nombre d'arc(s) : %d \n
",TRA_nbArc);}  STA <STA1_eta , STA1_fin , STA1_arcs> {*STA_arcs = STA1_arcs +
TRA_nbArc ; *STA_eta = STA1_eta + 1 ;
    if (bool) *STA_fin = STA1_fin + 1   ; else *STA_fin = STA1_fin   ;   }
   | {*STA_eta = 0 ; *STA_fin = 0 ; *STA_arcs = 0 ; }
;

ID : ATT_ID EGALE  IDENTIFIANT  <val>  {printf("\n\n\t Etat %s",val.id);}
;
FINAL <%out int bool> : ATT_FINAL EGALE  TRUE  {*bool = 1 ; }
     | ATT_FINAL EGALE  FALSE  {*bool = 0 ; }
     | {*bool = 0 ; }
;

TRANS < %out int  TRA_nbArc > : oBALISE_TRANSITION  EVENT TARGET
fBALISE_TRANSITION TRANS <TRA1_nbArc > {*TRA_nbArc = TRA1_nbArc +1;}
  |oBALISE_TRANSITION  TARGET EVENT  fBALISE_TRANSITION  TRANS <
TRA1_nbArc > {*TRA_nbArc = TRA1_nbArc +1;}
  | {*TRA_nbArc = 0 ;}
```

```
;

EVENT : ATT_EVENT EGALE  IDENTIFIANT <val>  {printf("\n\t\tArc : Etiquette %s - Etat
",val.id);}
;

TARGET : ATT_TARGET EGALE  IDENTIFIANT <val>  {printf("%s",val.id);}
;
```

5.5. Version 3 lex

```
%{
/* Fichier exAtt.lex */
#include "yystype.h"
#include "yygrammar.h"

char err[20]; /* Chaine de caracteres pour les erreurs de syntaxe */
%}

/* Definition de macros */
chiffre         [0-9]
separateur      [ \t]
lettre                  [a-zA-Z]
id                      [a-zA-Z0-9]|-|_

%%



"<?xml"
                return ENTETE;    /* Indique au parser que l'entête est reconnu */
"version"
                return ATT_VERSION; /* Indique au parser qu'un attribut version est
reconnu */
"\"1.0\""
                return VALEUR_VERSION; /* Indique au parser la version de l'attribut
version est reconnu */
"encoding"
                return ATT_ENCODING; /* Indique au parser qu'un attribut encoding
est reconnu */
"\"UTF-8\""
                return ATT_UTF; /* Indique au parser le type d'encodage de l'attribut
encoding est reconnu */
```

```
"<scxml"
                return oBALISE_SCXML; /* Indique au parser qu'une balise ouvrante
scxml  est reconnu  */
"xmlns"
                return ATT_XMLNS; /* Indique au parser qu'un attribut xmlns est
reconnu */
"\"http://www.w3.org/2005/07/scxml\""                                   return
VALEUR_REF; /* Indique au parser que la référence de l'attribut xmlns est reconnu
*/
"initial"
        return ATT_INIT; /* Indique au parser qu'un attribut initial est reconnu */
"</scxml"
                return fBALISE_SCXML; /* Indique au parser qu'une balise fermante
scxml   est reconnu  */

"<state"
                return oBALISE_STATE; /* Indique au parser qu'une balise ouvrante
state   est reconnu  */
"id"
                return ATT_ID; /* Indique au parser qu'un attribut id est reconnu */
"final"
                return ATT_FINAL; /* Indique au parser qu'un attribut final est reconnu
*/
"</state"
                return fBALISE_STATE; /* Indique au parser qu'une balise fermante
state   est reconnu */

"<transition"
        return oBALISE_TRANSITION ;   /* Indique au parser qu'une balise ouvrante
transition est reconnu */
"event"
                return ATT_EVENT;    /* Indique au parser que l'attribut event est
reconnu */
"target"
        return ATT_TARGET;    /* Indique au parser que l'attribut target est reconnu */


"\"true\""
                return TRUE;  /* Indique au parser qu un boolean TRUE est reconnu */
"\"false\""
                return FALSE;  /* Indique au parser qu un boolean  FALSE est
reconnu */
\"{lettre}{id}*\"                                                       {
yylval.id = (char * ) malloc (strlen(yytext)*sizeof(char)+1 ) ; strcpy(yylval.id , yytext + 1
) ;yylval.id[strlen(yytext)-2] = '\0' ;   return IDENTIFIANT; }    /* Indique au parser qu
un identifiant  est reconnu */
```

```
"?>"
                    return fENTETE; /* Indique au parser qu'une balise fermante  xml est
reconnu */
"/>"
                    return fBALISE_TRANSITION; /* Indique au parser qu'une balise
fermante transition est reconnu */
">"
                              return BALISEf; /* Indique au parser qu'une fin de balise
simple  est reconnu   (>)*/
"="
                              return EGALE; /* Indique au parser qu'un signe égale  est
reconnu */

{separateur}+                                                                    ;
/* Elimination des espaces */
\n                                                                          yypos++;
/* Compte le nombre de lignes du fichier source */
.                                                                           {
sprintf(err,"Mauvais caractere %c",yytext[0]);

yyerror(err);    /* Generation d'une erreur de syntaxe */
                                                                            }
%%
```

5.6. Version 3 acc

```
/* Fichier exAtt.acc */

%prelude{ /* Code C */
  /* Inclusion de bibliotheques C */
  #include <stdio.h>
  #include <stdlib.h>
  #include "yystype.h"
  #include "etats.h"
  #include <string.h>

  tabEtats tabE ;
```

```
}

/* Declaration des tokens */
%token ENTETE,
ATT_VERSION ,
ATT_ENCODING,
VALEUR_VERSION,
ATT_UTF,
VALEUR_REF,
fENTETE,
oBALISE_SCXML ,
ATT_XMLNS,
ATT_INIT,
fBALISE_SCXML ,
oBALISE_STATE,
ATT_ID,
ATT_FINAL,
fBALISE_STATE ,
oBALISE_TRANSITION,
ATT_EVENT,
ATT_TARGET,
fBALISE_TRANSITION,
BALISEf,
EGALE,
IDENTIFIANT,
TRUE,
FALSE;

// Grammaire   // erreur d'affichage des étiquettes
E :  ENTETE ATT_VERSION EGALE VALEUR_VERSION ATT_ENCODING EGALE
ATT_UTF fENTETE  { initTable(&tabE);} F { afficherTable(tabE);}
  | ENTETE  ATT_ENCODING EGALE ATT_UTF  ATT_VERSION EGALE
VALEUR_VERSION fENTETE  { initTable(&tabE);} F { afficherTable(tabE);}

;


F : oBALISE_SCXML ATT_XMLNS EGALE VALEUR_REF  INIT <INIT_nom>
BALISEf STA  fBALISE_SCXML BALISEf {definirEtatInitial(&tabE, INIT_nom);}
  | oBALISE_SCXML   INIT <INIT_nom> ATT_XMLNS EGALE VALEUR_REF
BALISEf STA  fBALISE_SCXML BALISEf {definirEtatInitial(&tabE, INIT_nom);}
;

INIT <%out chaineC INIT_nom>: ATT_INIT EGALE  IDENTIFIANT <val>  { strcpy(
INIT_nom , val.id) ;}
;
```

```
STA : oBALISE_STATE  ID <id_nom> FINAL <bool> BALISEf {  insererEtat( &tabE ,
id_nom) ; strcpy( nomEtat , id_nom) ;if(bool) definirEtatFinal( &tabE , id_nom) ; }
TRANS <nomEtat> fBALISE_STATE BALISEf STA
   | oBALISE_STATE  FINAL <bool> ID <id_nom> BALISEf { insererEtat( &tabE ,
id_nom) ; strcpy( nomEtat , id_nom) ; if(bool) definirEtatFinal( &tabE , id_nom) ;}
TRANS <nomEtat> fBALISE_STATE BALISEf STA
   |
;

ID <%out chaineC id_nom>: ATT_ID EGALE  IDENTIFIANT  <val>  {strcpy( id_nom ,
val.id); }
;

FINAL <%out int bool> : ATT_FINAL EGALE  TRUE {*bool = 1 ; }
     | ATT_FINAL EGALE  FALSE  {*bool = 0 ;    }
     | ATT_FINAL EGALE IDENTIFIANT {yyerror("l'attribut final de la balise state doit
avoir pour valeur soit true, soit false");}
     | {*bool = 0 ; }
;

TRANS <%in chaineC nomEtat> : oBALISE_TRANSITION  EVENT <EV_val>
TARGET <TAR_val>  fBALISE_TRANSITION  {insererArc(&tabE  , nomEtat ,
TAR_val  ,EV_val  ); strcpy( nomEtat1 , nomEtat) ;   } TRANS <nomEtat1>
  |oBALISE_TRANSITION   TARGET <TAR_val> EVENT <EV_val>
{insererArc(&tabE  , nomEtat , TAR_val  ,EV_val ) ; strcpy( nomEtat1 , nomEtat) ;  }
fBALISE_TRANSITION TRANS <nomEtat1>
  |
;

EVENT <%out chaineC EV_val>: ATT_EVENT EGALE  IDENTIFIANT <val>   {
strcpy( EV_val , val.id) ;}
;

TARGET <%out chaineC TAR_val> : ATT_TARGET EGALE  IDENTIFIANT <val>  {
strcpy( TAR_val , val.id) ;}
;
```