INFO-F403 Introduction to Language Theory and Compilation

Chapeaux Thomas
Dagnely Pierre

March 7, 2013

1 Lexèmes

Définition des tokens

Lexical units	regular expressions
INT	([0-9])*
FLOAT	([0-9])*.DOT.([0-9])*
BOOL	(0+1+true+false+")
STRING	'.([A-Za-z]+[0-9])*.'
FAC	!
MUL	*
DIV	/
MINUS	-
ADD	+
LT	<
GT	>
LE	<=
GE	>=
EQUIV	==
DIF	!=
AND	&&
OR	||
NOT	not
LT-S	lt
GT-S	gt
LE-S	le
GE-S	ge
EQ-S	eq
NE-S	ne

Lexical units	regular expressions
EQUAL	=
DOT	.
SEMICOLON	;
COMA	,
OPEN-PAR	(
CLOSE-PAR	)
OPEN-BRAC	{
CLOSE-BRAC	}
OPEN-COND	IF
CLOSE-COND	ELSE
ADD-COND	ELSE IF
NEG-COND	UNLESS
RET	return
FUNCT-DEF	SUB
ID	STRING
FUNCT-NAME	&.STRING
PERL-DEF	defined
PERL-INT	int
PERL-LENG	length
PERL-SCAL	scalar
PERL-SUBS	substr
PERL-PRIN	print
COMM	#.STRING
VARIABLE	$.STRING

coma peut définir l'opérateur coma ou juste un coma entre deux param, mais même lexical unit, c'est le parser qui se charge du reste
COMM ne sera plus utilisé par la suite, on les supprime avant de faire l'analyse lexicale

2 Automates

Définition des automates finis

DFA

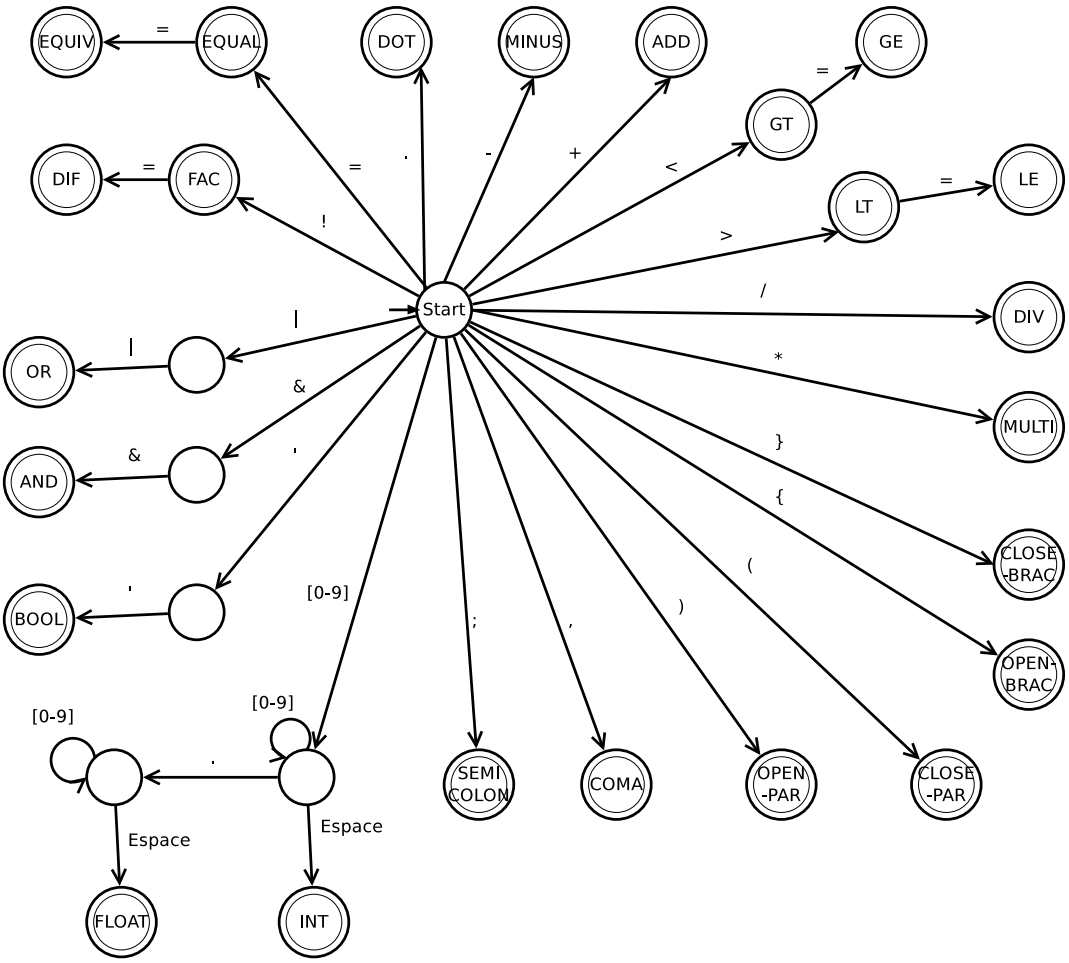

Figure 1: automate "non alphabétique"

La plupart des noeuds pointent vers le token ID, trop lourd a représenter, donc met une petite fleche bleu à la place.

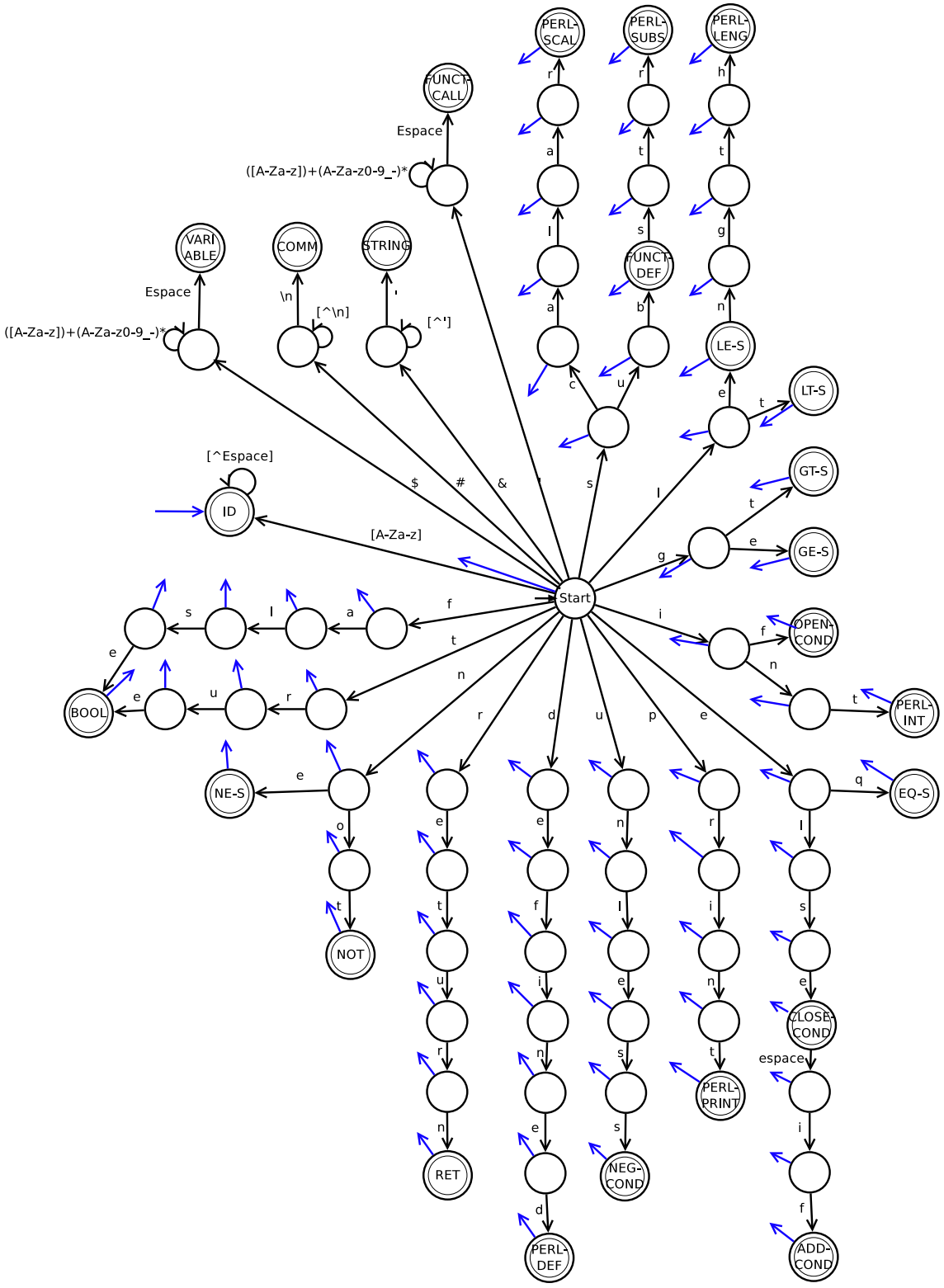

Figure 2: automate "alphabétique"

3 Grammaire

3.1 Grammaire de base

Nous utilisons pour définir la grammaire la version simplifiée de l'assistant :

<PROGRAM> → <FUNCT-LIST> <INSTRUCT-LIST>
→ <FUNCT-LIST>
→ <INSTRUCT-LIST>
<FUNCT-LIST> → <FUNCT>
→ <FUNCT-LIST> <FUNCT>
<FUNCT> → funct-def id open-par <FUNCT-ARG> close-par open-brac <INSTRUCT-LIST> close-brac
<FUNCT-ARG> → open-par <ARG-LIST> close-par
<ARG-LIST> → <ARG-LIST> coma variable
→ variable
→ epsilon
<INSTRUCT-LIST> → <INSTRUCT-LIST> <INSTRUCT> semicolon
→ <INSTRUCT> semicolon
<FUNCT-CALL> → funct-name <FUNCT-CALL-ARG>
<FUNCT-CALL-ARG> → <FUNCT-CALL-ARG> coma <EXP>
→ <EXP>
→ epsilon
<INSTRUCT> → variable equal <EXP>
→ <EXP>
→ ret <EXP>
→ <COND>
<COND> → open-cond <EXP> open-brac <INSTRUCT-LIST> close-brac <COND-END>
<COND-END> → add-cond <EXP> open-brac <INSTRUCT-LIST> close-brac <COND-END>
→ close-cond open-brac <INSTRUCT-LIST> close-brac
→ epsilon
<SIMPLE-EXP> → int
→ <FUNCT-CALL>
→ variable
→ string
<EXP> → <SIMPLE-EXP>
→ open-par <EXP> close-par
→ <EXP> add <EXP>
→ <EXP> minus <EXP>
→ <EXP> multi <EXP>
→ <EXP> div <EXP>
→ <EXP> equiv <EXP>
→ <EXP> gt <EXP>

3.2 Suppression des symboles inutiles

on vire les non-productifs et inaccessibles.
Y en a pas donc ca va vite.

3.3 Gestion des priorités et associativité

Cela ne concerne que la règle EXP, on la transforme donc en :

<EXP>	→ <EXP> equiv <EXP-2>
	→ <EXP> gt <EXP-2>
	→ <EXP-2>
<EXP-2>	→ <EXP-2> add <EXP-3>
	→ <EXP-2> minus <EXP-3>
	→ <EXP-3>
<EXP-3>	→ <EXP-3> mul <SIMPLE-EXP>
	→ <EXP-3> div <SIMPLE-EXP>
	→ <SIMPLE-EXP>

3.4 left factoring

Cela ne concerne que EXP, EXP-2, EXP-3 et PROGRAM :

<PROGRAM>	→ <FUNCT-LIST> <PROG-TAIL>
	→ <INSTRUCT-LIST>
<PROG-TAIL>	→ <INSTRUCT-LIST>
	→ epsilon

<EXP>	→ <EXP-2> <EXP-TAIL>
<EXP-TAIL>	→ equiv <EXP-2> <EXP-TAIL>
	→ gt <EXP-2> <EXP-TAIL>
	→ epsilon
<EXP-2>	→ <EXP-3> <EXP-2-TAIL>
<EXP-2-TAIL>	→ add <EXP-3> <EXP-2-TAIL>
	→ minus <EXP-3> <EXP-2-TAIL>
	→ epsilon
<EXP-3>	→ <SIMPLE-EXP> <EXP-3-TAIL>
<EXP-3-TAIL>	→ mul <SIMPLE-EXP> <EXP-3-TAIL>
	→ div <SIMPLE-EXP> <EXP-3-TAIL>
	→ epsilon

3.5 Left recursion

concerne FUNCT-LIST, ARG-LIST, INSTRUCT-LIST, FUNCT-CALL-ARG :

<FUNCT-LIST> → <FUNCT-LIST-BEG> <FUNCT-LIST-END>
<FUNCT-LIST-BEG> → <FUNCT>
<FUNCT-LIST-END> → <FUNCT> <FUNCT-LIST-END>
→ EPSILON

<ARG-LIST> → <ARG-LIST-BEG> <ARG-LIST-END>
<ARG-LIST-BEG> → variable
→ epsilon
<ARG-LIST-END> → coma variable <ARG-LIST-END>
→ epsilon

<INSTRUCT-LIST> → <INSTRUCT> semicolon <INSTRUCT-LIST>
→ epsilon

<FUNCT-CALL-ARG> → <FUNCT-CALL-ARG-BEG> <FUNCT-CALL-ARG-END>
<FUNCT-CALL-ARG-BEG> → <EXP>
→ epsilon
<FUNCT-CALL-ARG-END> → coma <EXP> <FUNCT-CALL-ARG-END>
→ epsilon

3.6 Suppresion des productions unitaires

règle FUNCT-ARG (<FUNCT-ARG> → open-par <ARG-LIST> close-par) est dans ce cas
On la remplace donc directement par ARG-LIST dans funct.

FUNCT devient donc : (<FUNCT> → funct-def id <ARG-LIST> open-brac <INSTRUCT-LIST> close-brac)
et arg-list devient : (<ARG-LIST> → open-ar <ARG-LIST-BEG> <ARG-LIST-END> close-par)
supprime donc une règle pas intermédiaire peu utile

3.7 Grammaire finale

<PROGRAM> → <FUNCT-LIST> <PROG-TAIL>
→ <INSTRUCT-LIST>
<PROG-TAIL> → <INSTRUCT-LIST>
→ epsilon
<FUNCT-LIST> → <FUNCT-LIST-BEG> <FUNCT-LIST-END>
<FUNCT-LIST-BEG> → <FUNCT>
<FUNCT-LIST-END> → <FUNCT> <FUNCT-LIST-END>
→ epsilon
<FUNCT> → funct-def id <ARG-LIST> open-brac <INSTRUCT-LIST> close-brac
<ARG-LIST> → open-par <ARG-LIST-BEG> <ARG-LIST-END> close-par
<ARG-LIST-BEG> → variable
→ epsilon
<ARG-LIST-END> → coma variable <ARG-LIST-END>
→ epsilon
<INSTRUCT-LIST> → <INSTRUCT> semicolon <INSTRUCT-LIST>
→ epsilon
<FUNCT-CALL> → funct-name open-par <FUNCT-CALL-ARG> close-par
<FUNCT-CALL-ARG> → <FUNCT-CALL-ARG-BEG> <FUNCT-CALL-ARG-END>
<FUNCT-CALL-ARG-BEG> → <EXP>
→ epsilon
<FUNCT-CALL-ARG-END> → coma <EXP> <FUNCT-CALL-ARG-END>
→ epsilon
<INSTRUCT> → variable equal <EXP>
→ ret <EXP>
→ <COND>
<COND> → open-cond <EXP> open-brac <INSTRUCT-LIST> close-brac <COND-END>
<COND-END> → close-cond open-brac <INSTRUCT-LIST> close-brac
→ add-cond <EXP> open-brac <INSTRUCT-LIST> close-brac <COND-END>
→ epsilon
<SIMPLE-EXP> → <FUNCT-CALL>
→ variable
→ int
→ string
→ open-par <EXP> close-par
<EXP> → <EXP-2> <EXP-TAIL>
<EXP-TAIL> → equiv <EXP-2> <EXP-TAIL>
→ gt <EXP-2> <EXP-TAIL>
→ epsilon
<EXP-2> → <EXP-3> <EXP-2-TAIL>
<EXP-2-TAIL> → add <EXP-3> <EXP-2-TAIL>
→ minus <EXP-3> <EXP-2-TAIL>
→ epsilon
<EXP-3> → <SIMPLE-EXP> <EXP-3-TAIL>
<EXP-3-TAIL> → mul <SIMPLE-EXP> <EXP-3-TAIL>
→ div <SIMPLE-EXP> <EXP-3-TAIL>
→ epsilon

4 Scanner

reconnait plus de token que ceux supporté par la grammaire.
because passé de grammaire complète à grammaire simplifié

5 Parser

5.1 Table de parsing

5.2 First and follow

6 Poubelle

6.1 Grammaire complète : bug

On se base sur la BNF donné par l'assistant et on l'adapte à notre version de perl

PROGRAM → PROGRAM FUNCT-LIST
→ PROGRAM INSTRUCT
→ FUNCT-LIST
→ INSTRUCT
→ EPSILON

FUNCT-LIST → FUNCT
→ FUNCT FUNCT-LIST
→ EPSILON

FUNCT → FUNCT-DEF ID OPEN-BRAC INSTRUCT RETURN CLOSE-BRAC
→ FUNCT-DEF ID OPEN-PAR CLOSE-PAR OPEN-BRAC INSTRUCT RETURN CLOSE-BRAC
→ FUNCT-DEF ID OPEN-PAR PARAM CLOSE-PAR OPEN-BRAC INSTRUCT RETURN CLOSE-BRAC

FUNCT-CALL → USER-FUNCT-CALL
→ PERL-FUNCT-CALL

USER-FUNCT-CALL → ID OPEN-PAR CLOSE-PAR
→ ID OPEN-PAR PARAM CLOSE-PAR
→ ID PARAM
→ ID

PERL-FUNCT-CALL → PERL-DEF EXP
→ PERL-INT EXP
→ PERL-LENG EXP
→ PERL-SCAL EXP
→ PERL-SUBS EXP COMA INT COMA INT
→ PERL-SUBS EXP COMA INT
→ PERL-PRIN LIST

LIST → STRING
→ STRING LIST
→ EPSILON

PARAM → VAR
→ VAR PARAM-END
→ EPSILON

PARAM-END → COMA VAR
→ COMA VAR PARAM-END
→ EPSILON

RETURN → RET EXP SEMICOLON
→ RET EXP-COND SEMICOLON
→ RET VAR SEMICOLON
→ EPSILON

INSTRUCT	→ COND SEMICOLON INSTRUCT
	→ EXP SEMICOLON INSTRUCT
	→ FUNCT-CALL SEMICOLON INSTRUCT
	→ ASSIGNATION SEMICOLON INSTRUCT
	→ COND SEMICOLON
	→ EXP SEMICOLON
	→ FUNCT-CALL SEMICOLON
	→ ASSIGNATION SEMICOLON
	→ EPSILON
ASSIGNATION	→ VAR EQUAL VALUE
	→ VAR EQUAL EXP
COND	→ OPEN-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END
	→ NEG-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END
	→ EXP OPEN-COND EXP-COND
	→ EXP NEG-COND EXP-COND
COND-END	→ ADD-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC
	→ ADD-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END
	→ CLOSE-COND OPEN-BRAC INSTRUCT CLOSE-BRAC
	→ EPSILON
EXP	→ VAR
	→ EXP OPERATOR EXP
	→ EXP-COMP
	→ NOT EXP
	→ FAC EXP
	→ ADD EXP
	→ MINUS EXP
EXP-COMP	→ EXP OPERATOR-COMP EXP
OPERATOR	→ MUL
	→ DIV
	→ MINUS
	→ CONC
	→ ADD
	→ COMA
	→ DOT
OPERATOR-COMP	→ LT
	→ GT
	→ LE
	→ GE
	→ EQUIV
	→ DIF
	→ AND-LOGIC
	→ OR
	→ LT-S
	→ GT-S
	→ LE-S
	→ GE-S
	→ COMA-LOGIC
	→ EQ-S
	→ NE-S
VALUE	→ INT
	→ FLOAT
	→ BOOL
	→ STRING
VAR	→ VALUE
	→ VARIABLE
	→ MINUS VAR
	→ ADD VAR
	→ OPEN-PAR EXP CLOSE-PAR

6.2 Gestion des priorités

On doit adapter la grammaire pour respecter les priorité et les associativité gauche/-droite.
Cela ne modifie que les règles EXP, EXP-COND, OPERATOR et OPERATOR-COND qui deviennent :

<EXP> → <EXP> LT <EXP2>
→ <EXP> LT-S <EXP2>
→ <EXP> GT <EXP2>
→ <EXP> GT-S <EXP2>
→ <EXP> LE <EXP2>
→ <EXP> LE-S <EXP2>
→ <EXP> GE <EXP2>
→ <EXP> GE-S <EXP2>
→ <EXP> EQUIV <EXP2>
→ <EXP> EQ-S <EXP2>
→ <EXP> NE-S <EXP2>
→ <EXP> DIF <EXP2>
→ <EXP> DOT <EXP2>
→ <EXP> COMA <EXP2>
→ <EXP2> EQUAL <EXP>
→ NOT <EXP>
→ FAC <EXP>
→ <EXP2>
<EXP2> → <EXP> ADD <EXP3>
→ <EXP> MINUS <EXP3>
→ <EXP> OR <EXP3>
→ <EXP3>
<EXP3> → <EXP> MUL <VAR>
→ <EXP> DIV <VAR>
→ <EXP> AND <VAR>
→ <VAR>

6.3 Suppression des symboles inutiles

On doit virer non-productifs et inaccessibles.
Ici ok, rien à faire

6.4 left-factoring

On vire ce qui commence pareillement

On obtient la gram suivante :

PROGRAM	→ PROGRAM PROG-END
	→ FUNCT-LIST
	→ INSTRUCT
	→ EPSILON
PROG-END	→ FUNCT-LIST
	→ INSTRUCT
FUNCT-LIST	→ FUNCT FUNCT-LIST-END
	→ EPSILON
FUNCT-LIST-END	→ FUNCT-LIST
	→ EPSILON
FUNCT	→ FUNCT-DEF ID FUNCT-END
FUNCT-END	→ OPEN-BRAC INSTRUCT RETURN CLOSE-BRAC
	→ OPEN-PAR FUNCT-END2
FUNCT-END2	→ CLOSE-PAR OPEN-BRAC INSTRUCT RETURN CLOSE-BRAC
	→ PARAM CLOSE-PAR OPEN-BRAC INSTRUCT RETURN CLOSE-BRAC
FUNCT-CALL	→ USER-FUNCT-CALL
	→ PERL-FUNCT-CALL
USER-FUNCT-CALL	→ ID USER-FUNCT-CALL-END
USER-FUNCT-CALL-END	→ OPEN-PAR USER-FUNCT-CALL-END2
	→ PARAM
	→ EPSILON
USER-FUNCT-CALL-END2	→ CLOSE-PAR
	→ PARAM CLOSE-PAR
PERL-FUNCT-CALL	→ PERL-DEF EXP
	→ PERL-INT EXP
	→ PERL-LENG EXP
	→ PERL-SCAL EXP
	→ PERL-SUBS PERL-SUBS-END
	→ PERL-PRIN LIST
PERL-SUBS-END	→ EXP COMA INT COMA INT
	→ EXP COMA INT
LIST	→ STRING LIST-END
LIST-END	→ LIST
	→ EPSILON
PARAM	→ PARAM2
	→ EPSILON
PARAM2	→ PARAM-END
	→ EPSILON
PARAM-END	→ COMA VAR PARAM-END2
	→ EPSILON
PARAM-END2	→ PARAM-END
	→ EPSILON
RETURN	→ RET RETURN-END
	→ EPSILON
RETURN-END	→ EXP SEMICOLON
	→ EXP-COND SEMICOLON
	→ VAR SEMICOLON

INSTRUCT	→ COND INSTRUCT-END
	→ EXP INSTRUCT-END
	→ FUNCT-CALL INSTRUCT-END
	→ ASSIGNATION INSTRUCT-END
	→ EPSILON
INSTRUCT-END	→ SEMICOLON INSTRUCT-END2
INSTRUCT-END2	→ INSTRUCT
	→ EPSILON
ASSIGNATION	→ VAR EQUAL ASSIGNATION-END
ASSIGNATION-END	→ VALUE
	→ EXP
COND	→ OPEN-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END
	→ NEG-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END
	→ EXP COND-END2
COND-END2	→ OPEN-COND EXP-COND
	→ NEG-COND EXP-COND
COND-END	→ ADD-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END3
	→ CLOSE-COND OPEN-BRAC INSTRUCT CLOSE-BRAC
	→ EPSILON
COND-END3	→ COND-END
	→ EPSILON
<EXP>	→ <EXP> <EXP-END>
	→ <EXP2> EQUAL <EXP>
	→ NOT <EXP>
	→ FAC <EXP>
	→ <EXP2>
<EXP-END>	→ LT <EXP2>
	→ LT-S <EXP2>
	→ GT <EXP2>
	→ GT-S <EXP2>
	→ LE <EXP2>
	→ LE-S <EXP2>
	→ GE <EXP2>
	→ GE-S <EXP2>
	→ EQUIV <EXP2>
	→ EQ-S <EXP2>
	→ NE-S <EXP2>
	→ DIF <EXP2>
	→ DOT <EXP2>
	→ COMA <EXP2>
<EXP2>	→ <EXP> <EXP2-END>
	→ <EXP3>
<EXP2-END>	→ ADD <EXP3>
	→ MINUS <EXP3>
	→ OR <EXP3>
<EXP3>	→ <EXP> <EXP3-END>
	→ <VAR>
<EXP3-END>	→ MUL <VAR>
	→ DIV <VAR>
	→ AND <VAR>
VALUE	→ INT
	→ FLOAT
	→ BOOL
	→ STRING
VAR	→ VALUE
	→ VARIABLE
	→ MINUS VAR
	→ ADD VAR
	→ OPEN-PAR EXP CLOSE-PAR

6.5 récursion gauche

On doit juste changer les règles PROGRAM et EXP qui deviennent:

PROGRAM	→ PROG PROG-TAIL
PROG	→ FUNCT-LIST
	→ INSTRUCT
	→ EPSILON
PROG-TAIL	→ PROG-END PROG-TAIL
	→ EPSILON
<EXP>	→ <E> <EXP-TAIL>
E	→ <EXP2> EQUAL <EXP>
	→ NOT <EXP>
	→ FAC <EXP>
	→ <EXP2>
EXP-TAIL	→ EXP-END EXP-TAIL
	→ EPSILON

6.6 Grammaire finale

PROGRAM	→ PROG PROG-TAIL
PROG	→ FUNCT-LIST
	→ INSTRUCT
	→ EPSILON
PROG-TAIL	→ PROG-END PROG-TAIL
	→ EPSILON
PROG-END	→ FUNCT-LIST
	→ INSTRUCT
FUNCT-LIST	→ FUNCT FUNCT-LIST-END
	→ EPSILON
FUNCT-LIST-END	→ FUNCT-LIST
	→ EPSILON
FUNCT	→ FUNCT-DEF ID FUNCT-END
FUNCT-END	→ OPEN-BRAC INSTRUCT RETURN CLOSE-BRAC
	→ OPEN-PAR FUNCT-END2
FUNCT-END2	→ CLOSE-PAR OPEN-BRAC INSTRUCT RETURN CLOSE-BRAC
	→ PARAM CLOSE-PAR OPEN-BRAC INSTRUCT RETURN CLOSE-BRAC
FUNCT-CALL	→ USER-FUNCT-CALL
	→ PERL-FUNCT-CALL
USER-FUNCT-CALL	→ ID USER-FUNCT-CALL-END
USER-FUNCT-CALL-END	→ OPEN-PAR USER-FUNCT-CALL-END2
	→ PARAM
	→ EPSILON
USER-FUNCT-CALL-END2	→ CLOSE-PAR
	→ PARAM CLOSE-PAR
PERL-FUNCT-CALL	→ PERL-DEF EXP
	→ PERL-INT EXP
	→ PERL-LENG EXP
	→ PERL-SCAL EXP
	→ PERL-SUBS PERL-SUBS-END
	→ PERL-PRIN LIST
PERL-SUBS-END	→ EXP COMA INT COMA INT
	→ EXP COMA INT
LIST	→ STRING LIST-END
LIST-END	→ LIST
	→ EPSILON
PARAM	→ PARAM2
	→ EPSILON
PARAM2	→ PARAM-END
	→ EPSILON
PARAM-END	→ COMA VAR PARAM-END2
	→ EPSILON
PARAM-END2	→ PARAM-END
	→ EPSILON
RETURN	→ RET RETURN-END
	→ EPSILON
RETURN-END	→ EXP SEMICOLON
	→ EXP-COND SEMICOLON
	→ VAR SEMICOLON

INSTRUCT → COND INSTRUCT-END
→ EXP INSTRUCT-END
→ FUNCT-CALL INSTRUCT-END
→ ASSIGNATION INSTRUCT-END
→ EPSILON
INSTRUCT-END → SEMICOLON INSTRUCT-END2
INSTRUCT-END2 → INSTRUCT
→ EPSILON
ASSIGNATION → VAR EQUAL ASSIGNATION-END
ASSIGNATION-END → VALUE
→ EXP
COND → OPEN-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END
→ NEG-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END
→ EXP COND-END2
COND-END2 → OPEN-COND EXP-COND
→ NEG-COND EXP-COND
COND-END → ADD-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END3
→ CLOSE-COND OPEN-BRAC INSTRUCT CLOSE-BRAC
→ EPSILON
COND-END3 → COND-END
→ EPSILON
<EXP> → <E> <EXP-TAIL>
E → <EXP2> EQUAL <EXP>
→ NOT <EXP>
→ FAC <EXP>
→ <EXP2>
EXP-TAIL → EXP-END EXP-TAIL
→ EPSILON
<EXP-END> → LT <EXP2>
→ LT-S <EXP2>
→ GT <EXP2>
→ GT-S <EXP2>
→ LE <EXP2>
→ LE-S <EXP2>
→ GE <EXP2>
→ GE-S <EXP2>
→ EQUIV <EXP2>
→ EQ-S <EXP2>
→ NE-S <EXP2>
→ DIF <EXP2>
→ DOT <EXP2>
→ COMA <EXP2>
<EXP2> → <EXP> <EXP2-END>
→ <EXP3>
<EXP2-END> → ADD <EXP3>
→ MINUS <EXP3>
→ OR <EXP3>
<EXP3> → <EXP> <EXP3-END>
→ <VAR>
<EXP3-END> → MUL <VAR>
→ DIV <VAR>
→ AND <VAR>

VALUE → INT
→ FLOAT
→ BOOL
→ STRING
VAR → VALUE
→ VARIABLE
→ MINUS VAR
→ ADD VAR
→ OPEN-PAR EXP CLOSE-PAR

EXPRESSION (?)	VARIABLE OPERATOR VARIABLE EXPRESSION OPERATOR VARIABLE
XPRESSION-COND (?)	VARIABLE OPERATOR-COMP VARIABLE EXPRESSION OPERATOR-COMP VARIABLE
ASSIGNATION	VARIABLE EQUAL VALUE
CONDITION (?)	((OPEN-COND+NEG-COND)EXPRESSION-COND OPEN-BRAC INSTRUCTIONS* CLOSE-BRAC (ADD-COND EXPRESSION-COND OPEN-BRAC INSTRUCTIONS* CLOSE-BRAC)* (CLOSE-COND EXPRESSION-COND OPEN-BRAC INSTRUCTIONS* CLOSE-BRAC)) + EXPRESSION (OPEN-COND + NEG-COND) EXPRESSION-COND
INSTRUCTIONS	((CONDITION SEMICOLON)* + (EXPRESSION SEMICOLON)* + (FUNCTION-CALL SEMICOLON)* + (ASSIGNATION SEMICOLON)*)*
PARAM	DOLLAR VARIABLE (COMA DOLLAR VARIABLE)*
USER-FUNCT-CALL	AND FUNCTION-NAME (OPEN-PAR CLOSE-PAR + OPEN-PAR PARAM CLOSE-PAR + PARAM) SEMICOLON
PERL-FUNCT-CALL	defined EXPRESSION + int EXPRESSION + length EXPRESSION scalar EXPRESSION + substr EXPRESSION COMA INT COMA INT scalar EXPRESSION + substr EXPRESSION COMA INT + print (?liste de string)
FUNCTION-CALL	USER-FUNCT-CALL + PERL-FUNCT-CALL
FUNCTION	FUNCTION-ID FUNCTION-NAME (OPEN-PAR CLOSE PAR + OPEN-PAR PARAM CLOSE-PAR) OPEN-BRAC INSTRUCTIONS (RETURN EXPRESSION + RETURN EXPRESSION-COND + RETURN VARIABLE) SEMICOLON CLOSE-BRAC
FUNCTION-LIST	FUNCTION*
PROGRAM	(FUNCTION-LIST + INSTRUCTIONS)*

Grammaire après gestion des priorité et associativité :

PROGRAM	→ PROGRAM FUNCT-LIST
	→ PROGRAM INSTRUCT
	→ FUNCT-LIST
	→ INSTRUCT
	→ EPSILON
FUNCT-LIST	→ FUNCT
	→ FUNCT FUNCT-LIST
	→ EPSILON
FUNCT	→ FUNCT-ID FUNCT-NAME OPEN-BRAC INSTRUCT RETURN CLOSE-BRAC
	→ FUNCT-ID FUNCT-NAME OPEN-PAR CLOSE PAR OPEN-BRAC INSTRUCT RETURN CLOSE-BRA
	→ FUNCT-ID FUNCT-NAME OPEN-PAR PARAM CLOSE-PAR OPEN-BRAC INSTRUCT RETURN CL
FUNCT-CALL	→ USER-FUNCT-CALL
	→ PERL-FUNCT-CALL
USER-FUNCT-CALL	→ FUNCT-NAME OPEN-PAR CLOSE-PAR
	→ FUNCT-NAME OPEN-PAR PARAM CLOSE-PAR
	→ FUNCT-NAME PARAM
	→ FUNCT-NAME
PERL-FUNCT-CALL	→ PERL-DEF EXP
	→ PERL-INT EXP
	→ PERL-LENG EXP
	→ PERL-SCAL EXP
	→ PERL-SUBS EXP COMA INT COMA INT
	→ PERL-SUBS EXP COMA INT
	→ PERL-PRIN LIST
LIST	→ STRING
	→ STRING LIST
	→ EPSILON
PARAM	→ VAR
	→ VAR PARAM-END
	→ EPSILON
PARAM-END	→ COMA VAR
	→ COMA VAR PARAM-END
	→ EPSILON
RETURN	→ RET EXP SEMICOLON
	→ RET EXP-COND SEMICOLON
	→ RET VAR SEMICOLON
	→ EPSILON

INSTRUCT → COND SEMICOLON INSTRUCT
→ EXP SEMICOLON INSTRUCT
→ FUNCT-CALL SEMICOLON INSTRUCT
→ ASSIGNATION SEMICOLON INSTRUCT
→ COND SEMICOLON
→ EXP SEMICOLON
→ FUNCT-CALL SEMICOLON
→ ASSIGNATION SEMICOLON
→ EPSILON

ASSIGNATION → VAR EQUAL VALUE
→ VAR EQUAL EXP

COND → OPEN-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END
→ NEG-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END
→ EXP OPEN-COND EXP-COND
→ EXP NEG-COND EXP-COND

COND-END → ADD-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC
→ ADD-COND EXP-COND OPEN-BRAC INSTRUCT CLOSE-BRAC COND-END
→ CLOSE-COND OPEN-BRAC INSTRUCT CLOSE-BRAC
→ EPSILON

<EXP> → <EXP> LT <EXP2>
→ <EXP> LT-S <EXP2>
→ <EXP> GT <EXP2>
→ <EXP> GT-S <EXP2>
→ <EXP> LE <EXP2>
→ <EXP> LE-S <EXP2>
→ <EXP> GE <EXP2>
→ <EXP> GE-S <EXP2>
→ <EXP> EQUIV <EXP2>
→ <EXP> DIF <EXP2>
→ <EXP2> EQUAL <EXP>
→ NOT <EXP>
→ FAC <EXP>
→ <EXP2>

<EXP2> → <EXP> ADD <EXP3>
→ <EXP> MINUS <EXP3>
→ <EXP> OR <EXP3>
→ <EXP3>

<EXP3> → <EXP> MUL <VAR>
→ <EXP> DIV <VAR>
→ <EXP> AND <VAR>
→ <VAR>

VALUE → INT
→ FLOAT
→ BOOL
→ STRING

VAR → VALUE
→ MINUS VAR
→ ADD VAR
→ OPEN-PAR EXP CLOSE-PAR