

Project INFO F 403 : Compilateur Perl

RODRIGUEZ Paul, VACCARI Eric

17 mars 2013

Table des matières

1	Unités lexicales	3
1.1	Tableau	3
1.2	Remarques	3
2	DFA	4
2.1	Variables, comparateurs, blocs, littéraux	4
2.2	Else, elsif et identifier	5
2.3	Opérateurs et divers	6
2.4	Remarques	6
3	Grammaire LL_1	6
3.1	Liste des règles	6
3.2	$First_1$	9
3.3	$Follow_1$	10
4	Table d'actions	12
5	Modifications	17
5.1	Appels de fonctions	17
5.2	Not	17
5.3	Assignation	17
5.4	Divers	17
6	Programme et Mode d'emploi	17

1 Unités lexicales

1.1 Tableau

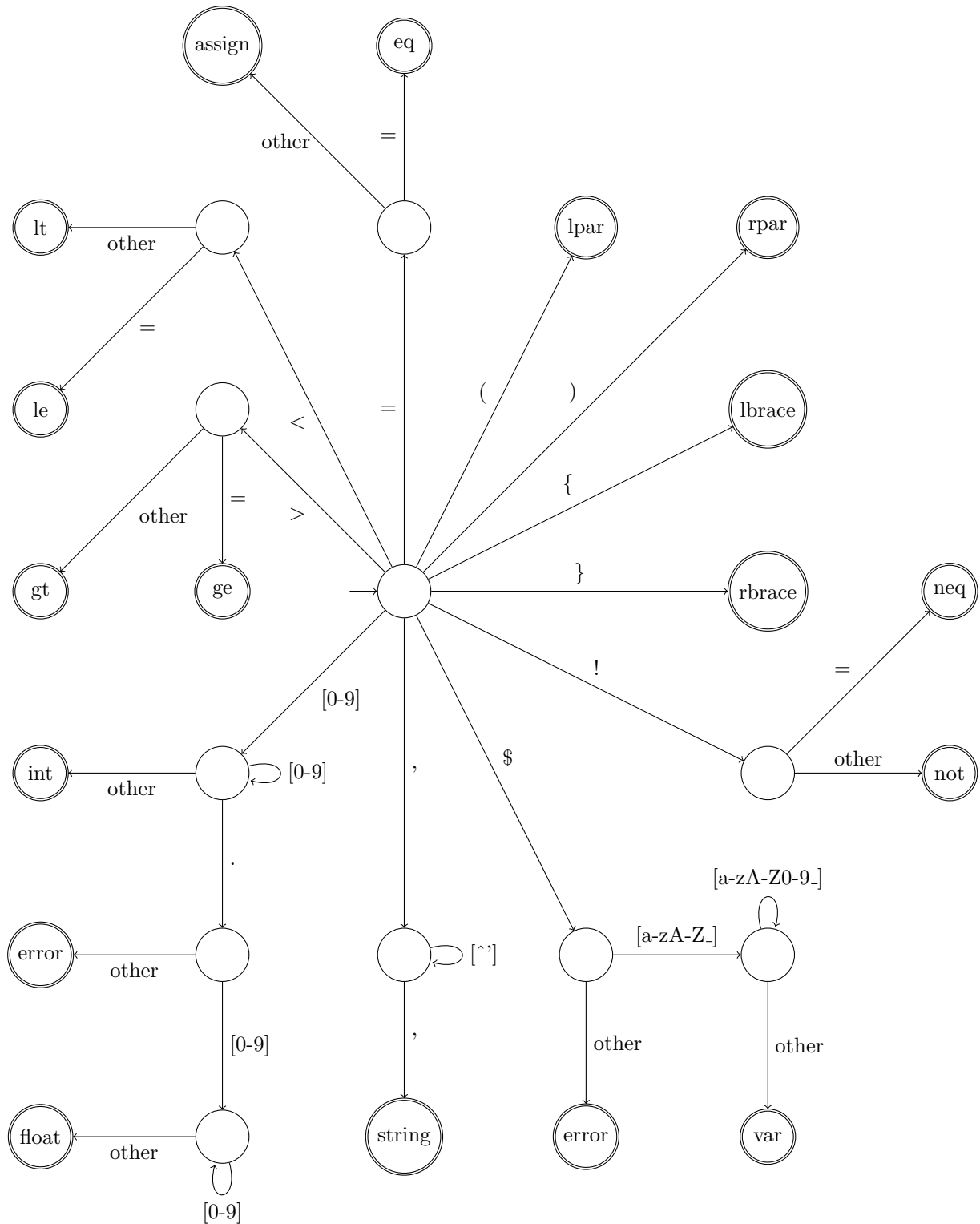
Nom	Regex
var	<code>[\$[a-zA-Z_][a-zA-Z0-9_]*]</code>
identifïer	<code>[a-zA-Z_][a-zA-Z0-9_]*</code>
integer	<code>[0-9]+</code>
float	<code>{integer}\.{integer}</code>
string	<code>'[^']*'</code>
space	<code>[\t\n]</code>
comment	<code>#.*\n</code>
lbrace	<code>\{</code>
rbrace	<code>\}</code>
lpar	<code>\(</code>
rpar	<code>\)</code>
semicolon	<code>;</code>
call_mark	<code>&</code>
plus	<code>\+</code>
minus	<code>\-</code>
times	<code>*</code>
divide	<code>\/</code>
not	<code>!</code>
notletters	<code>not</code>
lazy_and	<code>&&</code>
lazy_or	<code> </code>
equals	<code>==</code>
eq	<code>eq</code>
different	<code>!=</code>
ne	<code>ne</code>
lower	<code><</code>
lt	<code>lt</code>
greater	<code>></code>
gt	<code>gt</code>
lower_equals	<code><=</code>
le	<code>le</code>
greater_equals	<code>>=</code>
ge	<code>ge</code>
comma	<code>,</code>
concat_mark	<code>\.</code>
assign_mark	<code>=</code>
sub	<code>sub</code>
if	<code>if</code>
else	<code>else</code>
elsif	<code>elsif</code>
unless	<code>unless</code>
return	<code>return</code>

1.2 Remarques

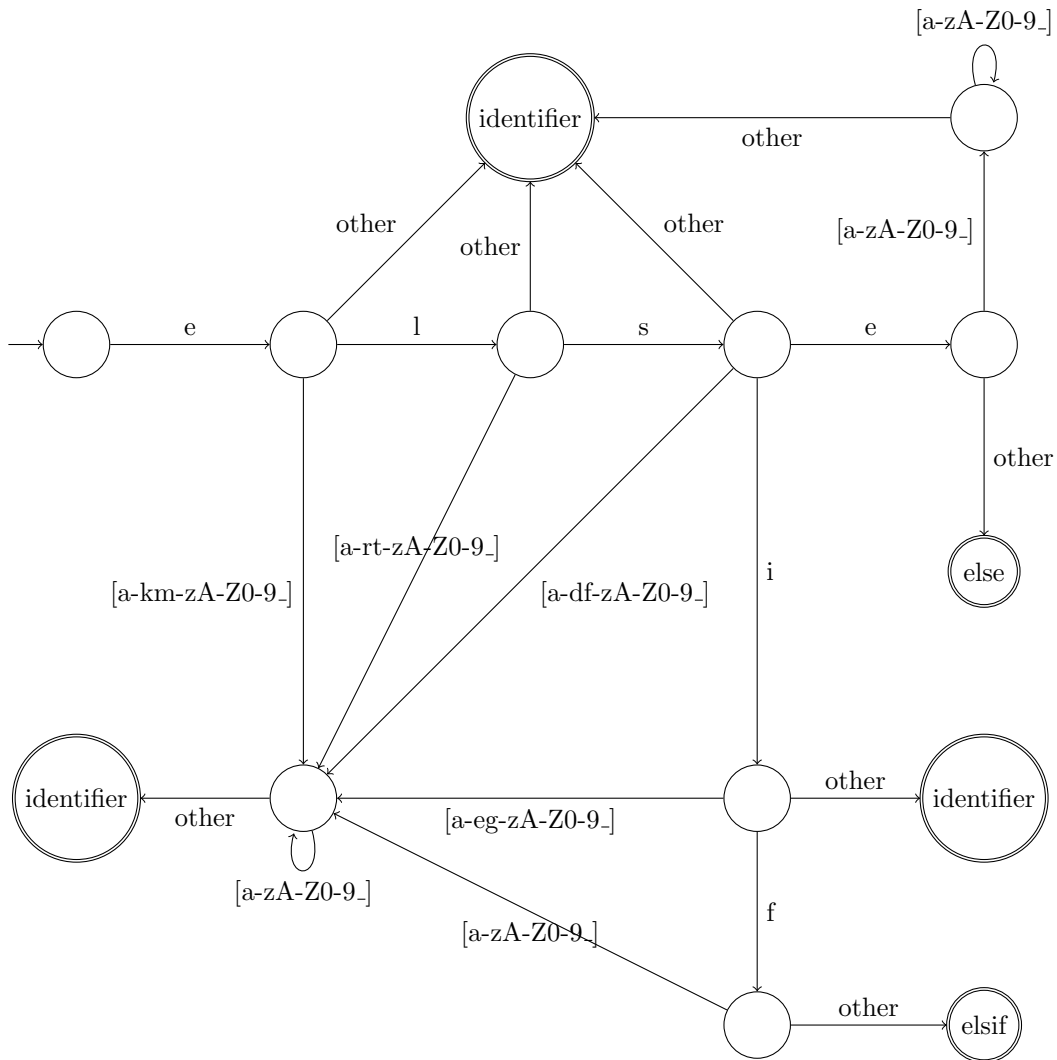
La syntaxe complète de Perl concernant les noms de variables est beaucoup plus compliquée mais concerne des fonctionnalités (packages) hors du cadre de ce projet, ce pourquoi nous nous sommes limités aux règles les plus simples.

2 DFA

2.1 Variables, comparateurs, blocs, littéraux

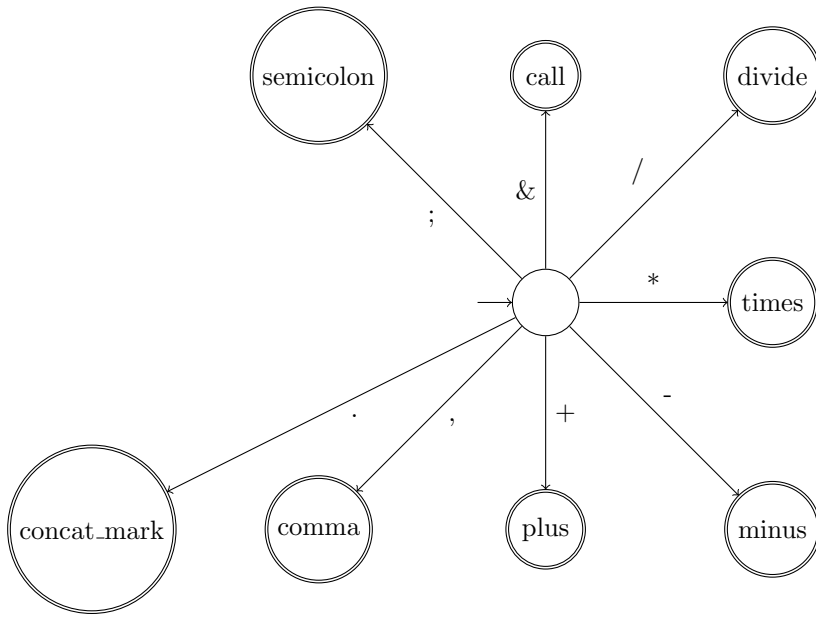


2.2 Else, elsif et identifier



Nous avons décidé de ne représenter que ces deux exemples, tous les mots clés fonctionnent sur le même principe.

2.3 Opérateurs et divers



2.4 Remarques

Certains tokens sont identifiables dès que leur dernier caractère a été lu (par exemple les accolades), d'autres nécessitent la lecture du caractère suivant le dernier (par exemple, pour terminer un entier il faut lire autre chose qu'un chiffre). Dans ce deuxième cas, après avoir identifié le token la lecture du dernier caractère est annulée, il servira comme premier caractère du token suivant.

3 Grammaire LL_1

3.1 Liste des règles

[1]	$\langle \text{PROGRAM} \rangle$	$\rightarrow \langle \text{PROGRAM_F} \rangle \langle \text{PROGRAM_V} \rangle$
[2]	$\langle \text{PROGRAM_V} \rangle$	$\rightarrow \langle \text{PROGRAM_F} \rangle \langle \text{PROGRAM_V} \rangle$
[3]		$\rightarrow \epsilon$
[4]	$\langle \text{PROGRAM_F} \rangle$	$\rightarrow \langle \text{FUNCTION} \rangle$
[5]		$\rightarrow \langle \text{INSTRUCTION} \rangle$
[6]	$\langle \text{FUNCTION} \rangle$	$\rightarrow \text{SUB IDENTIFIER } \langle \text{FUNCTION_ARGUMENT} \rangle$ $\text{LBRACE } \langle \text{INSTRUCTION_LIST} \rangle \text{ RBRACE}$
[7]	$\langle \text{FUNCTION_ARGUMENT} \rangle$	$\rightarrow \text{LPAR } \langle \text{ARGUMENT_LIST} \rangle \text{ RPAR}$
[8]		$\rightarrow \epsilon$
[9]	$\langle \text{ARGUMENT_LIST} \rangle$	$\rightarrow \text{VAR } \langle \text{ARGUMENT_LIST_V} \rangle$
[10]		$\rightarrow \epsilon$
[11]	$\langle \text{ARGUMENT_LIST_V} \rangle$	$\rightarrow \text{COMMA VAR } \langle \text{ARGUMENT_LIST_V} \rangle$
[12]		$\rightarrow \epsilon$
[13]	$\langle \text{INSTRUCTION_LIST} \rangle$	$\rightarrow \langle \text{INSTRUCTION} \rangle \langle \text{INSTRUCTION_LIST} \rangle$
[14]		$\rightarrow \epsilon$

[15]	$\langle \text{INSTRUCTION} \rangle$	$\rightarrow \langle \text{EXPRESSION} \rangle \langle \text{INSTRUCTION_F} \rangle \text{ SEMICOLON}$
[16]		$\rightarrow \text{RETURN } \langle \text{EXPRESSION} \rangle \langle \text{INSTRUCTION_F} \rangle \text{ SEMICOLON}$
[17]		$\rightarrow \text{LBRACE } \langle \text{INSTRUCTION_LIST} \rangle \text{ RBRACE}$
[18]		$\rightarrow \langle \text{CONDITION} \rangle \langle \text{EXPRESSION} \rangle \text{ LBRACE } \langle \text{INSTRUCTION_LIST} \rangle \text{ RBRACE } \langle \text{CONDITION_END} \rangle$
[19]	$\langle \text{INSTRUCTION_F} \rangle$	$\rightarrow \langle \text{CONDITION} \rangle \langle \text{EXPRESSION} \rangle$
[20]		$\rightarrow \epsilon$
[21]	$\langle \text{CONDITION} \rangle$	$\rightarrow \text{IF}$
[22]		$\rightarrow \text{UNLESS}$
[23]	$\langle \text{CONDITION_END} \rangle$	$\rightarrow \text{ELSIF } \langle \text{EXPRESSION} \rangle \text{ LBRACE } \langle \text{INSTRUCTION_LIST} \rangle \text{ RBRACE } \langle \text{CONDITION_END} \rangle$
[24]		$\rightarrow \text{ELSE LBRACE } \langle \text{INSTRUCTION_LIST} \rangle \text{ RBRACE}$
[25]		$\rightarrow \epsilon$
[26]	$\langle \text{EXPRESSION} \rangle$	$\rightarrow \langle \text{EXPRESSION_TWO} \rangle \langle \text{EXPRESSION_V} \rangle$
[27]	$\langle \text{EXPRESSION_V} \rangle$	$\rightarrow \text{ASSIGN_MARK } \langle \text{EXPRESSION_TWO} \rangle \langle \text{EXPRESSION_V} \rangle$
[28]		$\rightarrow \epsilon$
[29]	$\langle \text{EXPRESSION_TWO} \rangle$	$\rightarrow \langle \text{EXPRESSION_THREE} \rangle \langle \text{EXPRESSION_TWO_V} \rangle$
[30]	$\langle \text{EXPRESSION_TWO_V} \rangle$	$\rightarrow \text{LAZY_OR } \langle \text{EXPRESSION_THREE} \rangle \langle \text{EXPRESSION_TWO_V} \rangle$
[31]		$\rightarrow \epsilon$
[32]	$\langle \text{EXPRESSION_THREE} \rangle$	$\rightarrow \langle \text{EXPRESSION_FOUR} \rangle \langle \text{EXPRESSION_THREE_V} \rangle$
[33]	$\langle \text{EXPRESSION_THREE_V} \rangle$	$\rightarrow \text{LAZY_AND } \langle \text{EXPRESSION_FOUR} \rangle \langle \text{EXPRESSION_THREE_V} \rangle$
[34]		$\rightarrow \epsilon$
[35]	$\langle \text{EXPRESSION_FOUR} \rangle$	$\rightarrow \langle \text{EXPRESSION_FIVE} \rangle \langle \text{EXPRESSION_FOUR_V} \rangle$
[36]	$\langle \text{EXPRESSION_FOUR_V} \rangle$	$\rightarrow \langle \text{EXPRESSION_FOUR_F} \rangle \langle \text{EXPRESSION_FIVE} \rangle$
[37]		$\rightarrow \epsilon$
[38]	$\langle \text{EXPRESSION_FOUR_F} \rangle$	$\rightarrow \text{DIFFERENT}$
[39]		$\rightarrow \text{EQ}$
[40]		$\rightarrow \text{EQUALS}$
[41]		$\rightarrow \text{NE}$
[42]	$\langle \text{EXPRESSION_FIVE} \rangle$	$\rightarrow \langle \text{EXPRESSION_SIX} \rangle \langle \text{EXPRESSION_FIVE_V} \rangle$
[43]	$\langle \text{EXPRESSION_FIVE_V} \rangle$	$\rightarrow \langle \text{EXPRESSION_FIVE_F} \rangle \langle \text{EXPRESSION_SIX} \rangle$
[44]		$\rightarrow \epsilon$
[45]	$\langle \text{EXPRESSION_FIVE_F} \rangle$	$\rightarrow \text{GREATER}$
[46]		$\rightarrow \text{GREATER_EQUALS}$
[47]		$\rightarrow \text{GT}$
[48]		$\rightarrow \text{GE}$
[49]		$\rightarrow \text{LOWER}$
[50]		$\rightarrow \text{LOWER_EQUALS}$
[51]		$\rightarrow \text{LT}$
[52]		$\rightarrow \text{LE}$
[53]	$\langle \text{EXPRESSION_SIX} \rangle$	$\rightarrow \langle \text{EXPRESSION_SEVEN} \rangle \langle \text{EXPRESSION_SIX_V} \rangle$

[54]	$\langle \text{EXPRESSION_SIX_V} \rangle$	$\rightarrow \langle \text{EXPRESSION_SIX_F} \rangle \langle \text{EXPRESSION_SEVEN} \rangle$
[55]		$\langle \text{EXPRESSION_SIX_V} \rangle$ $\rightarrow \epsilon$
[56]	$\langle \text{EXPRESSION_SIX_F} \rangle$	$\rightarrow \text{PLUS}$
[57]		$\rightarrow \text{MINUS}$
[58]		$\rightarrow \text{CONCAT_MARK}$
[59]	$\langle \text{EXPRESSION_SEVEN} \rangle$	$\rightarrow \langle \text{EXPRESSION_EIGHT} \rangle \langle \text{EXPRESSION_SEVEN_V} \rangle$
[60]	$\langle \text{EXPRESSION_SEVEN_V} \rangle$	$\rightarrow \langle \text{EXPRESSION_SEVEN_F} \rangle \langle \text{EXPRESSION_EIGHT} \rangle$
[61]		$\langle \text{EXPRESSION_SEVEN_V} \rangle$ $\rightarrow \epsilon$
[62]	$\langle \text{EXPRESSION_SEVEN_F} \rangle$	$\rightarrow \text{TIMES}$
[63]		$\rightarrow \text{DIVIDE}$
[64]	$\langle \text{EXPRESSION_EIGHT} \rangle$	$\rightarrow \langle \text{EXPRESSION_NINE} \rangle$
[65]		$\rightarrow \langle \text{EXPRESSION_EIGHT_F} \rangle \langle \text{EXPRESSION_EIGHT} \rangle$
[66]	$\langle \text{EXPRESSION_EIGHT_F} \rangle$	$\rightarrow \text{NOT}$
[67]		$\rightarrow \text{PLUS}$
[68]		$\rightarrow \text{MINUS}$
[69]	$\langle \text{EXPRESSION_NINE} \rangle$	$\rightarrow \text{LPAR} \langle \text{EXPRESSION} \rangle \text{RPAR}$
[70]		$\rightarrow \langle \text{SIMPLE_EXPRESSION} \rangle$
[71]	$\langle \text{SIMPLE_EXPRESSION} \rangle$	$\rightarrow \langle \text{FUNCTION_CALL} \rangle$
[72]		$\rightarrow \text{INTEGER}$
[73]		$\rightarrow \text{FLOAT}$
[74]		$\rightarrow \text{STRING}$
[75]		$\rightarrow \text{VAR}$
[76]	$\langle \text{FUNCTION_CALL} \rangle$	$\rightarrow \text{CALL_MARK IDENTIFIER LPAR} \langle \text{ARGUMENT_CALL_LIST} \rangle$ RPAR
[77]	$\langle \text{ARGUMENT_CALL_LIST} \rangle$	$\rightarrow \langle \text{EXPRESSION} \rangle \langle \text{ARGUMENT_CALL_LIST_V} \rangle$
[78]		$\rightarrow \epsilon$
[79]	$\langle \text{ARGUMENT_CALL_LIST_V} \rangle$	$\rightarrow \text{COMMA} \langle \text{EXPRESSION} \rangle \langle \text{ARGUMENT_CALL_LIST_V} \rangle$
[80]		$\rightarrow \epsilon$

3.2 *First*₁

⟨PROGRAM⟩	: SUB, RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS
⟨PROGRAM_V⟩	: SUB, RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS, ϵ
⟨PROGRAM_F⟩	: SUB, RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS
⟨FUNCTION⟩	: SUB
⟨FUNCTION_ARGUMENT⟩	: LPAR, ϵ
⟨ARGUMENT_LIST⟩	: VAR, ϵ
⟨ARGUMENT_LIST_V⟩	: COMMA, ϵ
⟨INSTRUCTION_LIST⟩	: RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS, ϵ
⟨FUNCTION_CALL⟩	: CALL_MARK
⟨ARGUMENT_CALL_LIST⟩	: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS, ϵ
⟨ARGUMENT_CALL_LIST_V⟩	: COMMA, ϵ
⟨INSTRUCTION⟩	: RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS
⟨INSTRUCTION_F⟩	: IF, UNLESS, ϵ
⟨CONDITION⟩	: IF, UNLESS
⟨CONDITION_END⟩	: ELSIF, ELSE, ϵ
⟨EXPRESSION⟩	: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_V⟩	: ASSIGN_MARK, ϵ
⟨EXPRESSION_TWO⟩	: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_TWO_V⟩	: LAZY_OR, ϵ
⟨EXPRESSION_THREE⟩	: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_THREE_V⟩	: LAZY_AND, ϵ
⟨EXPRESSION_FOUR⟩	: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_FOUR_V⟩	: DIFFERENT, EQ, EQUALS, NE, ϵ
⟨EXPRESSION_FOUR_F⟩	: DIFFERENT, EQ, EQUALS, NE
⟨EXPRESSION_FIVE⟩	: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_FIVE_V⟩	: GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT, ϵ
⟨EXPRESSION_FIVE_F⟩	: GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT

$\langle \text{EXPRESSION_SIX} \rangle$: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
$\langle \text{EXPRESSION_SIX_V} \rangle$: PLUS, MINUS, CONCAT_MARK, ϵ
$\langle \text{EXPRESSION_SIX_F} \rangle$: PLUS, MINUS, CONCAT_MARK
$\langle \text{EXPRESSION_SEVEN} \rangle$: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
$\langle \text{EXPRESSION_SEVEN_V} \rangle$: TIMES, DIVIDE, ϵ
$\langle \text{EXPRESSION_SEVEN_F} \rangle$: TIMES, DIVIDE
$\langle \text{EXPRESSION_EIGHT} \rangle$: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
$\langle \text{EXPRESSION_EIGHT_F} \rangle$: NOT, PLUS, MINUS
$\langle \text{EXPRESSION_NINE} \rangle$: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK
$\langle \text{SIMPLE_EXPRESSION} \rangle$: INTEGER, FLOAT, STRING, VAR, CALL_MARK

3.3 *Follow₁*

$\langle \text{PROGRAM} \rangle$: ϕ
$\langle \text{PROGRAM_V} \rangle$: ϕ
$\langle \text{PROGRAM_F} \rangle$: SUB, RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS
$\langle \text{FUNCTION} \rangle$: SUB, RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS
$\langle \text{FUNCTION_ARGUMENT} \rangle$: LBRACE
$\langle \text{ARGUMENT_LIST} \rangle$: RPAR
$\langle \text{ARGUMENT_LIST_V} \rangle$: RPAR
$\langle \text{INSTRUCTION_LIST} \rangle$: RBRACE
$\langle \text{FUNCTION_CALL} \rangle$: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE, GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT, PLUS, MINUS, CONCAT_MARK, TIMES, DIVIDE
$\langle \text{ARGUMENT_CALL_LIST} \rangle$: RPAR
$\langle \text{ARGUMENT_CALL_LIST_V} \rangle$: RPAR
$\langle \text{INSTRUCTION} \rangle$: RBRACE, SUB, RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS
$\langle \text{INSTRUCTION_F} \rangle$: SEMICOLON
$\langle \text{CONDITION} \rangle$: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
$\langle \text{CONDITION_END} \rangle$: RBRACE, SUB, RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS
$\langle \text{EXPRESSION} \rangle$: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON

⟨EXPRESSION_V⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON
⟨EXPRESSION_TWO⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK
⟨EXPRESSION_TWO_V⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK
⟨EXPRESSION_THREE⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND
⟨EXPRESSION_THREE_V⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND
⟨EXPRESSION_FOUR⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE
⟨EXPRESSION_FOUR_V⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE
⟨EXPRESSION_FOUR_F⟩	: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_FIVE⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE, GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT
⟨EXPRESSION_FIVE_V⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE, GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT
⟨EXPRESSION_FIVE_F⟩	: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_SIX⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE, GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT, PLUS, MINUS, CONCAT_MARK
⟨EXPRESSION_SIX_V⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE, GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT, PLUS, MINUS, CONCAT_MARK
⟨EXPRESSION_SIX_F⟩	: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_SEVEN⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE, GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT, PLUS, MINUS, CONCAT_MARK, TIMES, DIVIDE
⟨EXPRESSION_SEVEN_V⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE, GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT, PLUS, MINUS, CONCAT_MARK, TIMES, DIVIDE
⟨EXPRESSION_SEVEN_F⟩	: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_EIGHT⟩	: IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE, GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT, PLUS, MINUS, CONCAT_MARK, TIMES, DIVIDE
⟨EXPRESSION_EIGHT_F⟩	: LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS

$\langle \text{EXPRESSION_NINE} \rangle$:	IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE, GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT, PLUS, MINUS, CONCAT_MARK, TIMES, DIVIDE
$\langle \text{SIMPLE_EXPRESSION} \rangle$:	IF, UNLESS, COMMA, LBRACE, RPAR, SEMICOLON, ASSIGN_MARK, LAZY_OR, LAZY_AND, DIFFERENT, EQ, EQUALS, NE, GE, GREATER, GREATER_EQUALS, GT, LE, LOWER, LOWER_EQUALS, LT, PLUS, MINUS, CONCAT_MARK, TIMES, DIVIDE

4 Table d'actions

	if	unless	else	elsif	sub	return	=	{	}	()	,	;	&		&&	!=	eq	==	ne	>	>=	gt	ge	<	<=	lt	le
<PROGRAM>	1	1			1		1	1	1	1	1			1														
<PROGRAM_V>	2	2			2		2	2	2	2	2			2														
<PROGRAM_F>	5	5			4		5	5	5	5	5			5														
<FUNCTION>					6																							
<FUNCTION_ARGUMENT>							8	8	7	7																		
<ARGUMENT_LIST>										10	10																	
<ARGUMENT_LIST_V>										12	11																	
<INSTRUCTION_LIST>	13	13				13	13	13	14	13				13														
<INSTRUCTION>	18	18				16	17	17	15	15				15														
<INSTRUCTION_F>	19	19											20															
<CONDITION>	21	22																										
<CONDITION_END>			24	23	25	25	25	25	25	25				25														
<SIMPLE_EXPRESSION>														71														
<FUNCTION_CALL>														76														
<ARGUMENT_CALL_LIST>										77	78			77														
<ARGUMENT_CALL_LIST_V>										80	79																	

	if	unless	else	elsif	sub	return	=	{	}	()	,	;	&		&&	!=	eq	==	ne	>	>=	gt	ge	<	<=	lt	le
<EXPRESSION>										26				26														
<EXPRESSION_V>	28	28					27	28		28	28	28		26														
<EXPRESSION_TWO>										29				29														
<EXPRESSION_TWO_V>	31	31					31	31		31	31	31			30													
<EXPRESSION_THREE>										32				32														
<EXPRESSION_THREE_V>	34	34					34	34		34	34	34			34	33												
<EXPRESSION_FOUR>										35				35														
<EXPRESSION_FOUR_V>	37	37					37	37		37	37	37			37	37	36	36	36	36								
<EXPRESSION_FOUR_F>																	38	39	40	41								
<EXPRESSION_FIVE>										42				42														
<EXPRESSION_FIVE_V>	44	44					44	44		44	44	44			44	44	44	44	44	44	43	43	43	43	43	43	43	43
<EXPRESSION_FIVE_F>																					45	46	47	48	49	50	51	52
<EXPRESSION_SIX>										53				53														
<EXPRESSION_SIX_V>	55	55					55	55		55	55	55			55	55	55	55	55	55	55	55	55	55	55	55	55	55
<EXPRESSION_SIX_F>																												
<EXPRESSION_SEVEN>										59				59														
<EXPRESSION_SEVEN_V>	61	61					61	61		61	61	61			61	61	61	61	61	61	61	61	61	61	61	61	61	61
<EXPRESSION_SEVEN_F>																												
<EXPRESSION_EIGHT>										64				64														
<EXPRESSION_EIGHT_F>																												
<EXPRESSION_NINE>										69				70														

	+	-	.	*	/	!	id	integer	float	string	var
$\langle \text{PROGRAM} \rangle$	1	1				1		1	1	1	1
$\langle \text{PROGRAM_V} \rangle$	2	2				2		2	2	2	2
$\langle \text{PROGRAM_F} \rangle$	5	5				5		5	5	5	5
$\langle \text{FUNCTION} \rangle$											
$\langle \text{FUNCTION_ARGUMENT} \rangle$											
$\langle \text{ARGUMENT_LIST} \rangle$											9
$\langle \text{ARGUMENT_LIST_V} \rangle$											
$\langle \text{INSTRUCTION_LIST} \rangle$	13	13				13		13	13	13	13
$\langle \text{INSTRUCTION} \rangle$	15	15				15		15	15	15	15
$\langle \text{INSTRUCTION_F} \rangle$											
$\langle \text{CONDITION} \rangle$											
$\langle \text{CONDITION_END} \rangle$	25	25				25		25	25	25	25
$\langle \text{SIMPLE_EXPRESSION} \rangle$								72	73	74	75
$\langle \text{FUNCTION_CALL} \rangle$											
$\langle \text{ARGUMENT_CALL_LIST} \rangle$	77	77				77		77	77	77	77
$\langle \text{ARGUMENT_CALL_LIST_V} \rangle$											

	+	-	.	*	/	!	id	integer	float	string	var
<EXPRESSION>	26	26				26		26	26	26	26
<EXPRESSION_V>											
<EXPRESSION_TWO>	29	29				29		29	29	29	29
<EXPRESSION_TWO_V>											
<EXPRESSION_THREE>	32	32				32		32	32	32	32
<EXPRESSION_THREE_V>											
<EXPRESSION_FOUR>	35	35				35		35	35	35	35
<EXPRESSION_FOUR_V>											
<EXPRESSION_FOUR_F>											
<EXPRESSION_FIVE>	42	42				42		42	42	42	42
<EXPRESSION_FIVE_V>											
<EXPRESSION_FIVE_F>											
<EXPRESSION_SIX>	53	53				53		53	53	53	53
<EXPRESSION_SIX_V>	54	54	54								
<EXPRESSION_SIX_F>	56	57	58								
<EXPRESSION_SEVEN>	59	59				59		59	59	59	59
<EXPRESSION_SEVEN_V>	61	61	61	60	60						
<EXPRESSION_SEVEN_F>				62	63						
<EXPRESSION_EIGHT>	65	65				65		64	64	64	64
<EXPRESSION_EIGHT_F>	68	67				66					
<EXPRESSION_NINE>								70	70	70	70

5 Modifications

Nous sommes partis de la grammaire originale de l'énoncé et l'avons modifiée au fur et à mesure pour la rendre LL_1 (après les transformations automatiques habituelles).

5.1 Appels de fonctions

Toutes les fonctions (y compris les fonctions prédéfinies) doivent être appelés en précédant leur nom par un "&". De plus, nous avons enlevé la possibilité d'omettre les parenthèses autour des listes d'arguments lors de l'appel d'une fonction. En effet ceci ne permettait pas d'obtenir une grammaire LL_1 , car un appel de fonction est une expression du plus bas niveau, mais le dernier argument est une expression du niveau le plus haut, et comme le dernier argument d'une fonction est potentiellement la dernière variable/le dernier token dans cette fonction (quand il n'y a pas de parenthèses pour entourer la liste d'arguments), l'expression de plus haut niveau "hérite" du follow de celle de plus bas niveau, ce qui crée des conflits.

5.2 Not

Le symbole "not" en toutes lettres tel que décrit dans la syntaxe génère le même genre de conflits. Ce symbole "transforme" ce qui se trouve derrière en une expression de plus haut niveau (et ce afin de respecter sa priorité faible). Mais ceci place une expression de haut niveau à la fin d'une expression de bas niveau, et on obtient des conflits. Nous avons tout simplement supprimé ce symbole.

5.3 Assignment

La grammaire autorise l'assignation de n'importe quelle expression à n'importe quelle autre, c'est lors de l'analyse sémantique que la validité de ce genre d'expressions est déterminée.

5.4 Divers

- On ne respecte pas les spécificités de Perl comme par exemple "0 but true" qui est évalué à zéro comme entier mais à vrai comme booléen.
- La fonction scalar ne fait rien dans le cadre de ce projet, car nous n'avons pas de tableaux.

6 Programme et Mode d'emploi

Nous n'avons pas pu finir le compilateur. Les analyses lexicales et syntaxiques fonctionnent correctement et selon les modifications décrites dans la section précédente, mais la génération du code n'est pas finie. Le compilateur est capable de générer du code ARMV5TE correct pour les définitions et appels de fonctions avec un nombre quelconque de paramètres, les littéraux (réels, entiers et chaînes de caractères), certains opérateurs, les variables globales. Les fonctions print et length ont été implémentées (nous avons le code assembleur des autres mais l'intégrer dans le compilateur n'est pas rapide), mais pas les conditions (si un code perl avec des if/elsif/else est lu en input, le comportement du programme est imprévisible).

Le compilateur a été écrit en C++. Pour le lancer, il suffit de lui donner en argument le nom d'un fichier contenant du code perl simplifié. L'output se fera dans le fichier "code.s". Nous avons rédigé un makefile très simple pour le compiler.