Project INFO F 403 : Compilateur Perl

RODRIGUEZ Paul, VACCARI Eric

18 mars 2013

Table des matières

1 Unités lexicales

1.1 Tableau

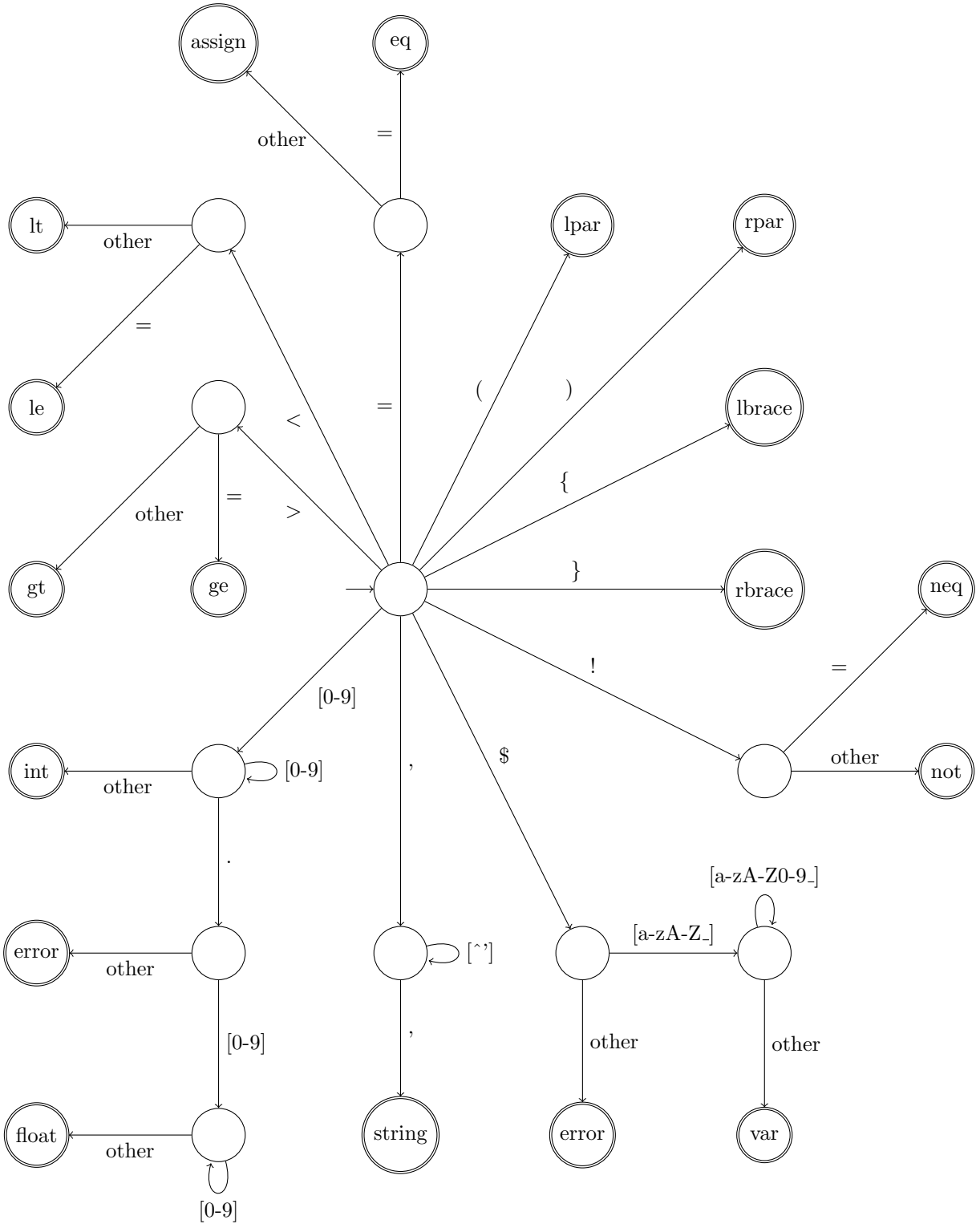
Nom	Regex
var	`$[a-zA-Z_][a-zA-Z0-9_]*`
identifier	`[a-zA-Z_][a-zA-Z0-9_]*`
integer	`[0-9]+`
float	`{integer}\.{integer}`
string	`'[^']*'`
space	`[\t\n ]`
comment	`#.*\n`
lbrace	`\{`
rbrace	`\}`
lpar	`\(`
rpar	`\)`
semicolon	`;`
call_mark	`&`
plus	`\+`
minus	`\-`
times	`\*`
divide	`\/`
not	`!`
notletters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

1.2 Remarques

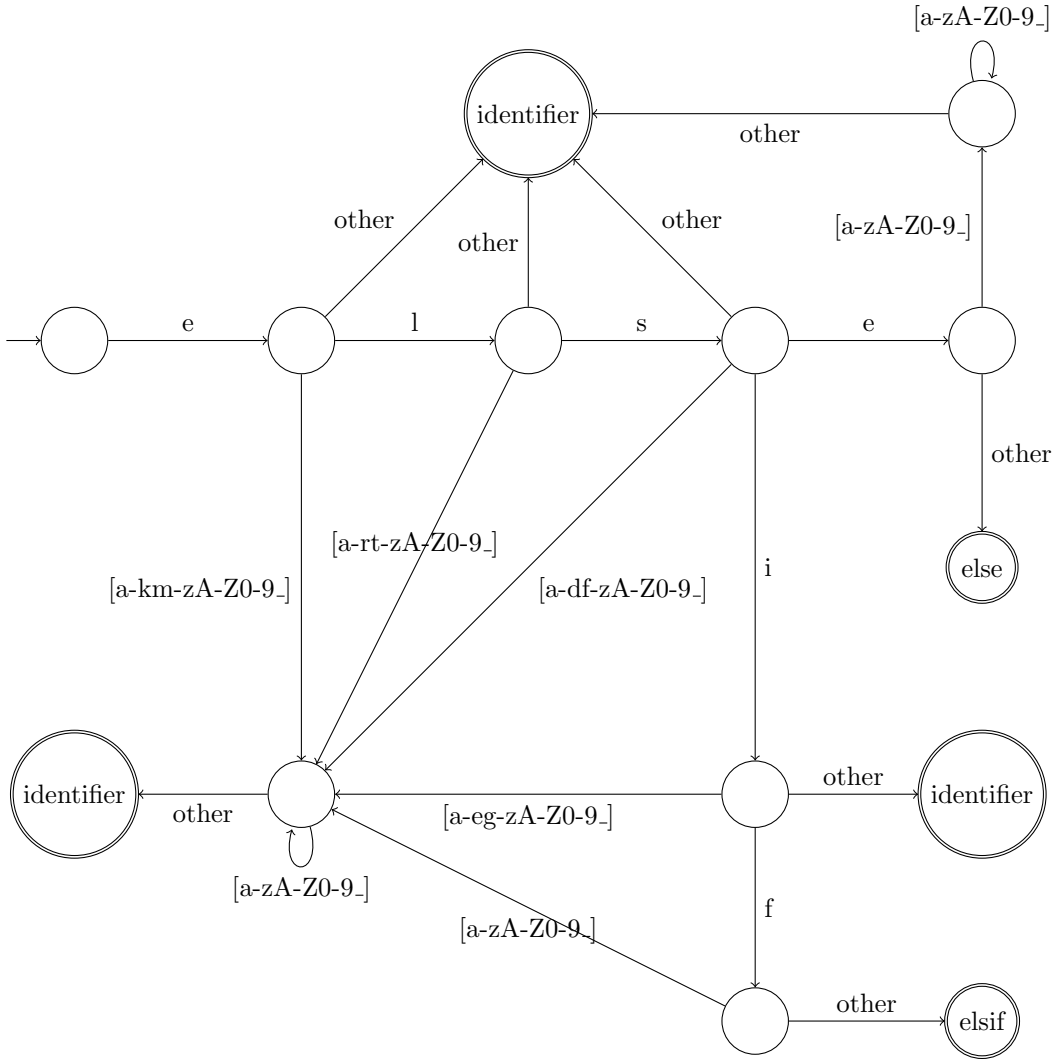
La syntaxe complète de Perl concernant les noms de variables est beaucoup plus compliquée mais concerne des fonctionalités (packages) hors du cadre de ce projet, ce pourquoi nous nous sommes limités aux règles les plus simples.

2 DFA

2.1 Variables, comparateurs, blocs, litéraux

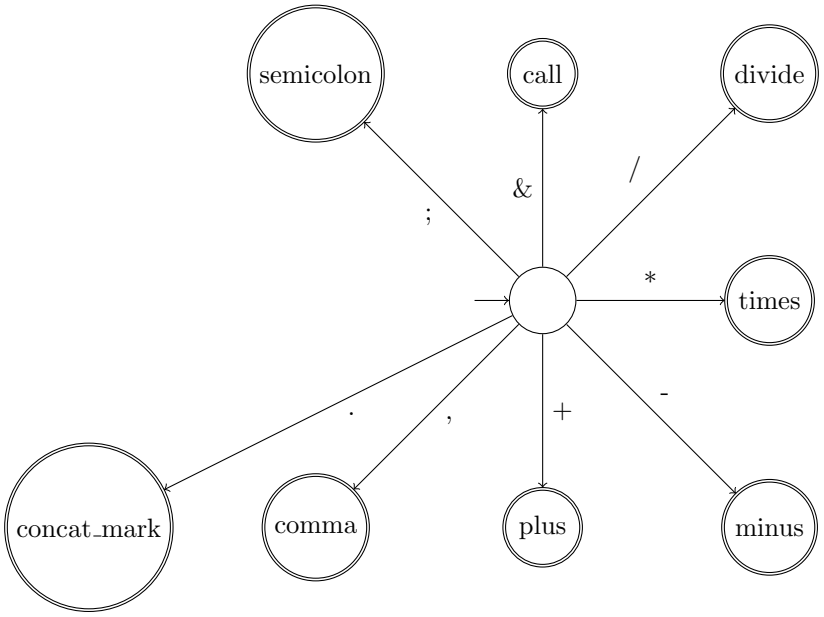


2.2 Else, elsif et identifier



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]	⟨PROGRAM⟩	⟶ ⟨PROGRAM_F⟩ ⟨PROGRAM_V⟩
[2]	⟨PROGRAM_V⟩	⟶ ⟨PROGRAM_F⟩ ⟨PROGRAM_V⟩
[3]		⟶ ϵ
[4]	⟨PROGRAM_F⟩	⟶ ⟨FUNCTION⟩
[5]		⟶ ⟨INSTRUCTION⟩
[6]	⟨FUNCTION⟩	⟶ SUB IDENTIFIER ⟨FUNCTION_ARGUMENT⟩ LBRACE ⟨INSTRUCTION_LIST⟩ RBRACE
[7]	⟨FUNCTION_ARGUMENT⟩	⟶ LPAR ⟨ARGUMENT_LIST⟩ RPAR
[8]		⟶ ϵ
[9]	⟨ARGUMENT_LIST⟩	⟶ VAR ⟨ARGUMENT_LIST_V⟩
[10]		⟶ ϵ
[11]	⟨ARGUMENT_LIST_V⟩	⟶ COMMA VAR ⟨ARGUMENT_LIST_V⟩
[12]		⟶ ϵ
[13]	⟨INSTRUCTION_LIST⟩	⟶ ⟨INSTRUCTION⟩ ⟨INSTRUCTION_LIST⟩
[14]		⟶ ϵ

[15] ⟨INSTRUCTION⟩ ⟶ ⟨EXPRESSION⟩ ⟨INSTRUCTION_F⟩ SEMICOLON
[16] ⟶ RETURN ⟨EXPRESSION⟩ ⟨INSTRUCTION_F⟩ SEMICOLON
[17] ⟶ LBRACE ⟨INSTRUCTION_LIST⟩ RBRACE
[18] ⟶ ⟨CONDITION⟩ ⟨EXPRESSION⟩ LBRACE ⟨INSTRUCTION_LIST⟩ RBRACE ⟨CONDITION_END⟩

[19] ⟨INSTRUCTION_F⟩ ⟶ ⟨CONDITION⟩ ⟨EXPRESSION⟩
[20] ⟶ ϵ

[21] ⟨CONDITION⟩ ⟶ IF
[22] ⟶ UNLESS

[23] ⟨CONDITION_END⟩ ⟶ ELSIF ⟨EXPRESSION⟩ LBRACE ⟨INSTRUCTION_LIST⟩ RBRACE ⟨CONDITION_END⟩
[24] ⟶ ELSE LBRACE ⟨INSTRUCTION_LIST⟩ RBRACE
[25] ⟶ ϵ

[26] ⟨EXPRESSION⟩ ⟶ ⟨EXPRESSION_TWO⟩ ⟨EXPRESSION_V⟩

[27] ⟨EXPRESSION_V⟩ ⟶ ASSIGN_MARK ⟨EXPRESSION_TWO⟩ ⟨EXPRESSION_V⟩
[28] ⟶ ϵ

[29] ⟨EXPRESSION_TWO⟩ ⟶ ⟨EXPRESSION_THREE⟩ ⟨EXPRESSION_TWO_V⟩

[30] ⟨EXPRESSION_TWO_V⟩ ⟶ LAZY_OR ⟨EXPRESSION_THREE⟩ ⟨EXPRESSION_TWO_V⟩
[31] ⟶ ϵ

[32] ⟨EXPRESSION_THREE⟩ ⟶ ⟨EXPRESSION_FOUR⟩ ⟨EXPRESSION_THREE_V⟩

[33] ⟨EXPRESSION_THREE_V⟩ ⟶ LAZY_AND ⟨EXPRESSION_FOUR⟩ ⟨EXPRESSION_THREE_V⟩
[34] ⟶ ϵ

[35] ⟨EXPRESSION_FOUR⟩ ⟶ ⟨EXPRESSION_FIVE⟩ ⟨EXPRESSION_FOUR_V⟩

[36] ⟨EXPRESSION_FOUR_V⟩ ⟶ ⟨EXPRESSION_FOUR_F⟩ ⟨EXPRESSION_FIVE⟩
[37] ⟶ ϵ

[38] ⟨EXPRESSION_FOUR_F⟩ ⟶ DIFFERENT
[39] ⟶ EQ
[40] ⟶ EQUALS
[41] ⟶ NE

[42] ⟨EXPRESSION_FIVE⟩ ⟶ ⟨EXPRESSION_SIX⟩ ⟨EXPRESSION_FIVE_V⟩

[43] ⟨EXPRESSION_FIVE_V⟩ ⟶ ⟨EXPRESSION_FIVE_F⟩ ⟨EXPRESSION_SIX⟩
[44] ⟶ ϵ

[45] ⟨EXPRESSION_FIVE_F⟩ ⟶ GREATER
[46] ⟶ GREATER_EQUALS
[47] ⟶ GT
[48] ⟶ GE
[49] ⟶ LOWER
[50] ⟶ LOWER_EQUALS
[51] ⟶ LT
[52] ⟶ LE

[53] ⟨EXPRESSION_SIX⟩ ⟶ ⟨EXPRESSION_SEVEN⟩ ⟨EXPRESSION_SIX_V⟩

[54] ⟨EXPRESSION_SIX_V⟩ ⟶ ⟨EXPRESSION_SIX_F⟩ ⟨EXPRESSION_SEVEN⟩ ⟨EXPRESSION_SIX_V⟩
[55] ⟶ ϵ

[56] ⟨EXPRESSION_SIX_F⟩ ⟶ PLUS
[57] ⟶ MINUS
[58] ⟶ CONCAT_MARK

[59] ⟨EXPRESSION_SEVEN⟩ ⟶ ⟨EXPRESSION_EIGHT⟩ ⟨EXPRESSION_SEVEN_V⟩

[60] ⟨EXPRESSION_SEVEN_V⟩ ⟶ ⟨EXPRESSION_SEVEN_F⟩ ⟨EXPRESSION_EIGHT⟩ ⟨EXPRESSION_SEVEN_V⟩
[61] ⟶ ϵ

[62] ⟨EXPRESSION_SEVEN_F⟩ ⟶ TIMES
[63] ⟶ DIVIDE

[64] ⟨EXPRESSION_EIGHT⟩ ⟶ ⟨EXPRESSION_NINE⟩
[65] ⟶ ⟨EXPRESSION_EIGHT_F⟩ ⟨EXPRESSION_EIGHT⟩

[66] ⟨EXPRESSION_EIGHT_F⟩ ⟶ NOT
[67] ⟶ PLUS
[68] ⟶ MINUS

[69] ⟨EXPRESSION_NINE⟩ ⟶ LPAR ⟨EXPRESSION⟩ RPAR
[70] ⟶ ⟨SIMPLE_EXPRESSION⟩

[71] ⟨SIMPLE_EXPRESSION⟩ ⟶ ⟨FUNCTION_CALL⟩
[72] ⟶ INTEGER
[73] ⟶ FLOAT
[74] ⟶ STRING
[75] ⟶ VAR

[76] ⟨FUNCTION_CALL⟩ ⟶ CALL_MARK IDENTIFIER LPAR ⟨ARGUMENT_CALL_LIST⟩ RPAR

[77] ⟨ARGUMENT_CALL_LIST⟩ ⟶ ⟨EXPRESSION⟩ ⟨ARGUMENT_CALL_LIST_V⟩
[78] ⟶ ϵ

[79] ⟨ARGUMENT_CALL_LIST_V⟩ ⟶ COMMA ⟨EXPRESSION⟩ ⟨ARGUMENT_CALL_LIST_V⟩
[80] ⟶ ϵ

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

⟨EXPRESSION_SIX⟩	:	LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_SIX_V⟩	:	PLUS, MINUS, CONCAT_MARK, ϵ
⟨EXPRESSION_SIX_F⟩	:	PLUS, MINUS, CONCAT_MARK
⟨EXPRESSION_SEVEN⟩	:	LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_SEVEN_V⟩	:	TIMES, DIVIDE, ϵ
⟨EXPRESSION_SEVEN_F⟩	:	TIMES, DIVIDE
⟨EXPRESSION_EIGHT⟩	:	LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨EXPRESSION_EIGHT_F⟩	:	NOT, PLUS, MINUS
⟨EXPRESSION_NINE⟩	:	LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK
⟨SIMPLE_EXPRESSION⟩	:	INTEGER, FLOAT, STRING, VAR, CALL_MARK

3.3 $Follow_1$

⟨PROGRAM⟩	:	ϕ
⟨PROGRAM_V⟩	:	ϕ
⟨PROGRAM_F⟩	:	SUB, RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS
⟨FUNCTION⟩	:	SUB, RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS
⟨FUNCTION_ARGUMENT⟩	:	LBRACE
⟨ARGUMENT_LIST⟩	:	RPAR
⟨ARGUMENT_LIST_V⟩	:	RPAR
⟨INSTRUCTION_LIST⟩	:	RBRACE
⟨FUNCTION_CALL⟩	:	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
⟨ARGUMENT_CALL_LIST⟩	:	RPAR
⟨ARGUMENT_CALL_LIST_V⟩	:	RPAR
⟨INSTRUCTION⟩	:	RBRACE, SUB, RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS
⟨INSTRUCTION_F⟩	:	SEMICOLON
⟨CONDITION⟩	:	LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, NOT, PLUS, MINUS
⟨CONDITION_END⟩	:	RBRACE, SUB, RETURN, LBRACE, LPAR, INTEGER, FLOAT, STRING, VAR, CALL_MARK, IF, UNLESS, NOT, PLUS, MINUS
⟨EXPRESSION⟩	:	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

⟨EXPRESSION_NINE⟩	:	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
⟨SIMPLE_EXPRESSION⟩	:	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														
⟨PROGRAM_V⟩	2	2			2			2		2	2			2														
⟨PROGRAM_F⟩	5	5			4			5	5	5	5			5														
⟨FUNCTION⟩					6																							
⟨FUNCTION_ARGUMENT⟩								8		7																		
⟨ARGUMENT_LIST⟩											10																	
⟨ARGUMENT_LIST_V⟩											12	11																
⟨INSTRUCTION_LIST⟩	13	13				13		13	14	13				13														
⟨INSTRUCTION⟩	18	18				16		17		15				15														
⟨INSTRUCTION_F⟩	19	19											20															
⟨CONDITION⟩	21	22																										
⟨CONDITION_END⟩			24	23	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															
⟨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
⟨PROGRAM⟩	1	1				1		1	1	1	1
⟨PROGRAM_V⟩	2	2				2		2	2	2	2
⟨PROGRAM_F⟩	5	5				5		5	5	5	5
⟨FUNCTION⟩											
⟨FUNCTION_ARGUMENT⟩											
⟨ARGUMENT_LIST⟩											9
⟨ARGUMENT_LIST_V⟩											
⟨INSTRUCTION_LIST⟩	13	13				13		13	13	13	13
⟨INSTRUCTION⟩	15	15				15		15	15	15	15
⟨INSTRUCTION_F⟩											
⟨CONDITION⟩											
⟨CONDITION_END⟩	25	25				25		25	25	25	25
⟨SIMPLE_EXPRESSION⟩								72	73	74	75
⟨FUNCTION_CALL⟩											
⟨ARGUMENT_CALL_LIST⟩	77	77				77		77	77	77	77
⟨ARGUMENT_CALL_LIST_V⟩											

	+	-	.	*	/	!	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 derniere 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 derriere en une expression de plus haut niveau (et ce afin de respecter sa priorité faible). Mais ceci place une expression de haut niveau a la fin d'une expression de bas niveau, et on obtient des conflits. Nous avons tout simplement supprimé ce symbole.

5.3 Assignation

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 specificités 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 Fonctionnalités

Nous n'avons pas pu finir le compilateur. Les analyses lexicale et syntaxique 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 (et aussi des appels de fonctions dans le code d'autre fonctions), les litéraux (réels, entiers et chaines de caractères), certains opérateurs, les variables globales. Les fonctions print et length on 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).

Il est important de remarquer que le typage dynamique fonctionne. Toutes nos valeurs en Perl sont des pointeurs vers des triplets avec un type, une valeur en virgule flottante (servant aussi a la représentation d'entiers) et une chaine de caractères. Nous avons inféré une bonne partie de la syntaxe de l'assembleur ARM à partir des résultats de compilations de programmes C simples, mais nous avons été limités sur certains aspects (comme l'usage de la GOT pour utiliser des litéraux). Un typecheck systématique est rajouté lorsque nécessaire, par exemple dans la fonction print avant d'appeler le code C. En l'état, les programmes générés par le compilateur sont remplis de fuites de mémoire, car toutes les variables sont déclarées dynamiquement et que nous n'avons pas eu le temps de trouver tous les endroits où une désallocation est possible.

Nous utilisons la pile pour stocker les valeurs intermédiaires lors de l'évaluation d'une expression. Par exemple, si on veut évaluer "3 + 4;", d'abord on push une variable initialisée à 3, puis une autre à 4, puis l'opérateur + retire les deux dernières variables de la pile et les additionne (après un typecheck) et push son résultat (remarquez que du coup nous pouvons avoir un opérateur + associatif à gauche sans que la grammaire aie une récursivité à gauche). Tous les opérateurs et fonctions passent leur résultats de cette façon. De même, nous avions prévu que un "if" fasse des tests sur la dernière case de la pile, celle-ci contenant le résultat de l'expression contenue dans la condition du "if".

7 Mode d'emploi

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.