Project INFO F 403 : Compilateur Perl

RODRIGUEZ Paul, VACCARI Eric

2 mars 2013

Table des matières

1 Unités lexicales

1.1 Tableau

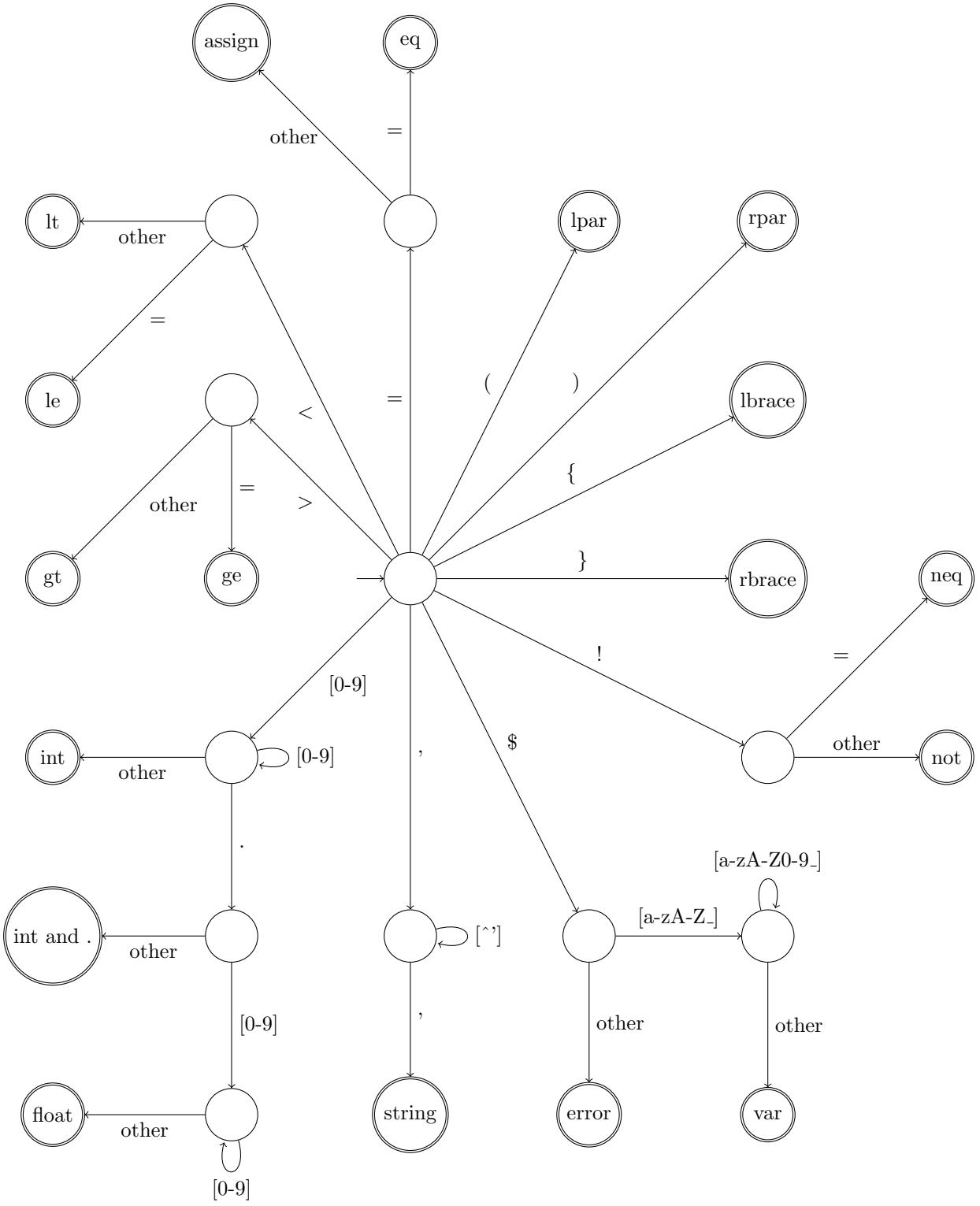
Nom	Regex
var	`$[a-zA-Z_][a-zA-Z0-9_]*`
identifier	`[a-zA-Z_][a-zA-Z0-9_]*`
integer	`[0-9]+`
float	`{integer}\.{integer}`
string	`'[^']*'`
space	`[\t\n ]`
comment	`#.*\n`
lbrace	`\{`
rbrace	`\}`
lpar	`\(`
rpar	`\)`
semicolon	`;`
call_mark	`&`
plus	`\+`
minus	`\-`
times	`\*`
divide	`\/`
not	`!`
notletters	`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`
defined	`defined`
int	`int`
length	`length`
print	`print`
scalar	`scalar`
substr	`substr`

1.2 Remarques

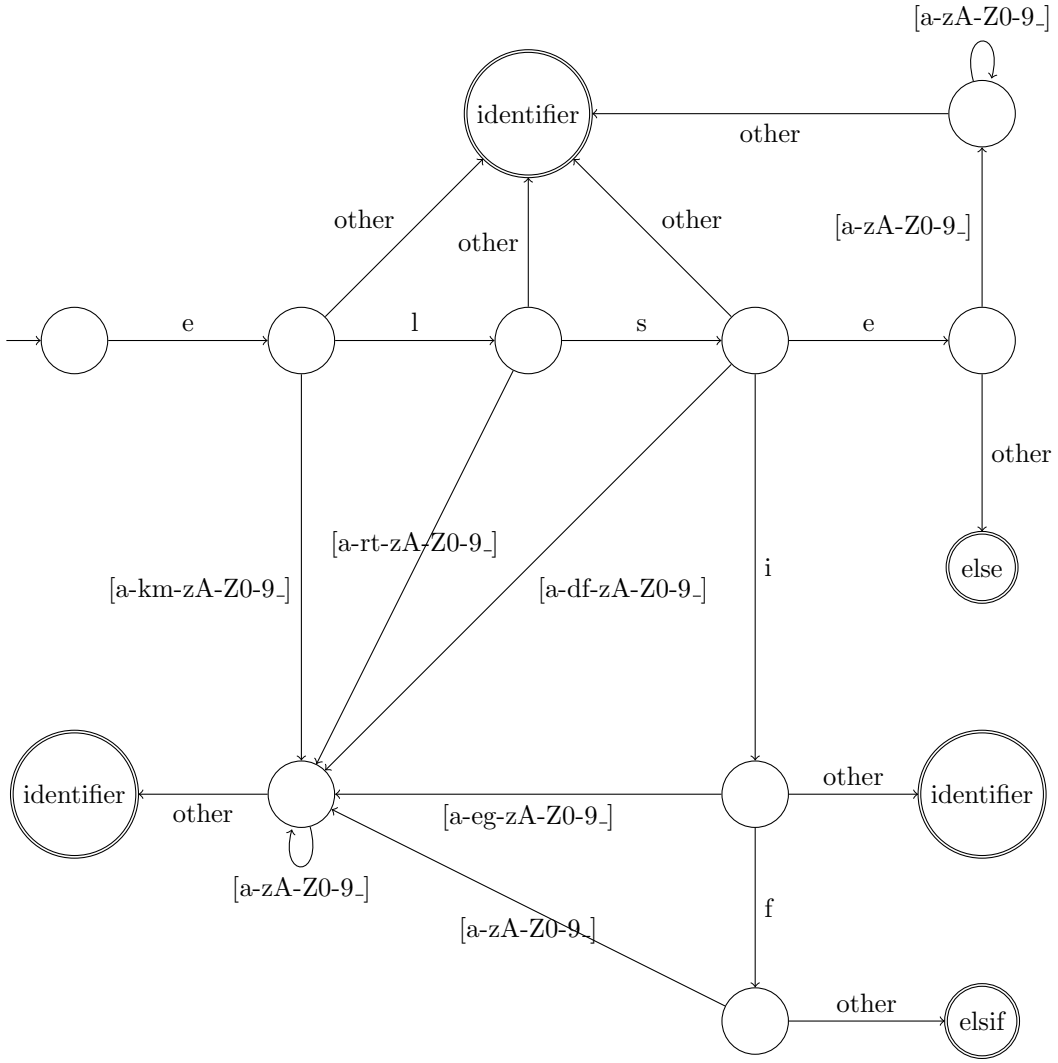
La syntaxe complète de Perl concernant les noms de variables est beaucoup plus compliquée mais concerne des fonctionalités (packages) hors du cadre de ce projet, ce pourquoi nous nous sommes limités aux règles les plus simples.

2 DFA

2.1 Variables, comparateurs, blocs, litéraux

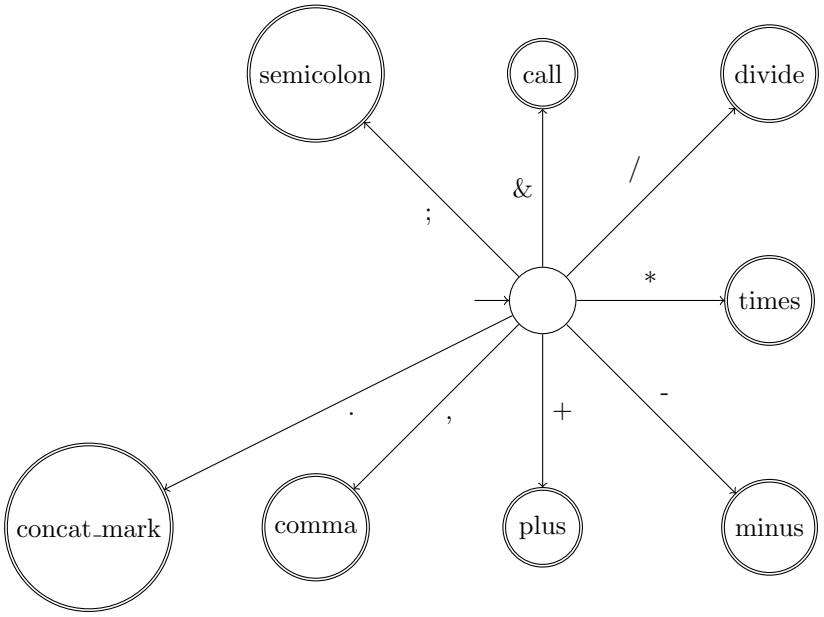


2.2 Else, elsif et identifier



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)

[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⟩ ⟶ NOTLETTERS ⟨EXPRESSION⟩
[27] ⟶ ⟨EXPRESSION_TWO⟩

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

[29] ⟨EXPRESSION_TWO_V⟩ ⟶ ASSIGN_MARK ⟨EXPRESSION_THREE⟩ ⟨EXPRESSION_TWO_V⟩
[30] ⟶ ϵ

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

[32] ⟨EXPRESSION_THREE_V⟩ ⟶ LAZY_OR ⟨EXPRESSION_FOUR⟩ ⟨EXPRESSION_THREE_V⟩
[33] ⟶ ϵ

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

[35] ⟨EXPRESSION_FOUR_V⟩ ⟶ LAZY_AND ⟨EXPRESSION_FIVE⟩ ⟨EXPRESSION_FOUR_V⟩
[36] ⟶ ϵ

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

[38] ⟨EXPRESSION_FIVE_V⟩ ⟶ ⟨EXPRESSION_FIVE_F⟩ ⟨EXPRESSION_SIX⟩
[39] ⟶ ϵ

[40] ⟨EXPRESSION_FIVE_F⟩ ⟶ DIFFERENT
[41] ⟶ EQ
[42] ⟶ EQUALS
[43] ⟶ NE

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

[45] ⟨EXPRESSION_SIX_V⟩ ⟶ ⟨EXPRESSION_SIX_F⟩ ⟨EXPRESSION_SEVEN⟩
[46] ⟶ ϵ

[47] ⟨EXPRESSION_SIX_F⟩ ⟶ GE
[48] ⟶ GREATER
[49] ⟶ GREATER_EQUALS
[50] ⟶ GT
[51] ⟶ LE
[52] ⟶ LOWER
[53] ⟶ LOWER_EQUALS
[54] ⟶ LT

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

[56] ⟨EXPRESSION_SEVEN_V⟩ ⟶ ⟨EXPRESSION_SEVEN_F⟩ ⟨EXPRESSION_EIGHT⟩ ⟨EXPRESSION_SEVEN_V⟩
[57] ⟶ ϵ

[58] ⟨EXPRESSION_SEVEN_F⟩ ⟶ PLUS
[59] ⟶ MINUS
[60] ⟶ CONCAT_MARK

[61] ⟨EXPRESSION_EIGHT⟩ ⟶ ⟨EXPRESSION_NINE⟩ ⟨EXPRESSION_EIGHT_V⟩

[62] ⟨EXPRESSION_EIGHT_V⟩ ⟶ ⟨EXPRESSION_EIGHT_F⟩ ⟨EXPRESSION_NINE⟩ ⟨EXPRESSION_EIGHT_V⟩
[63] ⟶ ϵ

[64] ⟨EXPRESSION_EIGHT_F⟩ ⟶ TIMES
[65] ⟶ DIVIDE

[66] ⟨EXPRESSION_NINE⟩ ⟶ ⟨EXPRESSION_NINE_V⟩ ⟨EXPRESSION_TEN⟩

[67] ⟨EXPRESSION_NINE_V⟩ ⟶ ⟨EXPRESSION_NINE_F⟩ ⟨EXPRESSION_NINE_V⟩
[68] ⟶ ϵ

[69] ⟨EXPRESSION_NINE_F⟩ ⟶ NOT
[70] ⟶ PLUS
[71] ⟶ MINUS

[72] ⟨EXPRESSION_TEN⟩ ⟶ LPAR ⟨EXPRESSION⟩ RPAR
[73] ⟶ ⟨SIMPLE_EXPRESSION⟩

[74] ⟨SIMPLE_EXPRESSION⟩ ⟶ ⟨FUNCTION_CALL⟩
[75] ⟶ INTEGER
[76] ⟶ FLOAT
[77] ⟶ STRING
[78] ⟶ VAR

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

[80] ⟨ARGUMENT_CALL_LIST⟩ ⟶ ⟨EXPRESSION⟩ ⟨ARGUMENT_CALL_LIST_V⟩
[81] ⟶ ϵ

[82] ⟨ARGUMENT_CALL_LIST_V⟩ ⟶ COMMA ⟨EXPRESSION⟩ ⟨ARGUMENT_CALL_LIST_V⟩
[83] ⟶ ϵ

remarques : 1) on a enlevé la possibilité d'omettre les () autour des listes d'arguments a l'appel d'une fonction, c'est pas LL(1) car le follow de expression prend le follow de toutes les expressions

2) On a supprimé le "not", car sa priorité faible engendre un comportement non-LL(1), le follow de expression prend le follow de toutes les expressions