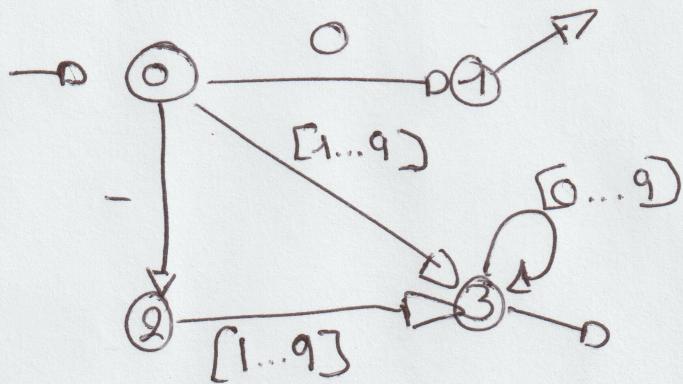


D17

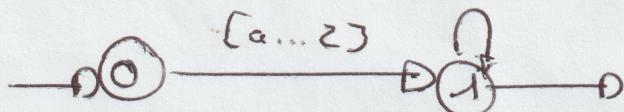
LANGAGES ET AUTOMATES.

Partie 1

1. Automate d'état fini minimal pour $01[1..9]([1..9][0..9])^*$



2. Automate d'état fini minimal pour $[a..z]([a..z][0..9])^*$



Partie 2

2.1.1

La grammaire G n'est pas LL(1) car elle contient des règles récursives gauches:

$\vdash E \rightarrow E \text{ OpArithm } E \mid (E) \mid \text{VarNum}$

$\vdash B \text{Expression} \rightarrow \text{ValBool} \mid \text{not } B \text{Expression} \mid B \text{Expression and } B \text{Expression} \mid B \text{Expression or } B \text{Expression} \mid \text{condition.}$

2.1.2

Pour rendre la grammaire LL(1), on commence par retirer les règles récursives.

$\hookrightarrow \text{Expression} \rightarrow \text{Expression OpArith Expression' } | (\text{Expression}) | \text{VarNum}$

$\text{Expression} \rightarrow (\text{Expression}) \text{ Expression' } | \text{VarNum Expression'}$

$\text{Expression'} \rightarrow \text{OpArith Expression Expression' } | \epsilon$

$\hookrightarrow \text{BExpression} \rightarrow \text{ValBool } | \text{not BExpression } |$

$\quad \quad \quad \text{BExpression and BExpression } |$

$\quad \quad \quad \text{BExpression or BExpression } |$

$\quad \quad \quad \text{Condition}$

$\text{BExpression} \rightarrow \text{ValBool BExpression' } |$

$\quad \quad \quad \text{not BExpression BExpression' } |$

$\quad \quad \quad \text{Condition BExpression' } |$

$\text{BExpression'} \rightarrow \text{or BExpression BExpression' } |$

$\quad \quad \quad \text{and BExpression BExpression' } |$

$\quad \quad \quad \epsilon$

2.1.2 suite

On peut ensuite factoriser les règles obtenus.

Expression

Expression \rightarrow TExpression Expression'

TExpression \rightarrow (Expression) | VarNum

Expression' \rightarrow OpArith Expression Expression' | E

BExpression

BExpression \rightarrow GBExpression BExpression'

GBExpression \rightarrow ValBool | not BExpression | Condition.

BExpression'

BExpression' \rightarrow TBExpression BExpression' | E

TBExpression \rightarrow or BExpression | and BExpression

TBExpression

TBExpression \rightarrow TBExpression' BExpression

TBExpression' \rightarrow or | and

Maintenant, il faut introduire la notion de priorité sur les opérations. Il en existe deux types, les opérateurs logiques (or, and et not), et arithmétique (+, -, ×, ÷).

d'ordre sera :

arithmétique : \times et \div , puis $+$ et $-$

logique : not, puis and, puis enfin or

2.1.2 suite.

Transformation des opérateurs arithmétiques

des règles concernées sont Expression, Expression' et TExpression

$$\text{Expression} \rightarrow \text{TExpression Expression}'$$

$$\text{Expression}' \rightarrow \text{OpPP TExpression Expression}' \mid \epsilon$$

$$\text{TExpression} \rightarrow \text{NExpression TExpression}'$$

$$\text{TExpression}' \rightarrow \text{OpP NExpression TExpression}' \mid \epsilon$$

$$\text{NExpression} \rightarrow \text{VarNum} \mid (\text{Expression})$$

$$\text{OpPP} \rightarrow + \mid -$$

$$\text{OpP} \rightarrow * \mid /$$

Transformation des opérateurs arithmétiques logiques

des règles concernées sont BExpression, BExpression', TBExpression, TBExpression' et GBExpression.

$$\text{BExpression} \rightarrow \text{CombExpression BExpression}'$$

$$\text{BExpression}' \rightarrow \text{or CombExpression BExpression}' \mid \epsilon$$

$$\text{CombExpression} \rightarrow \text{NotExpression AndExpression}$$

$$\text{AndExpression} \rightarrow \text{and NotExpression AndExpression} \mid \epsilon$$

$$\text{NotExpression} \rightarrow \text{not NotExpression} \mid \text{Bool} \mid \text{Not Bool}$$

$$\text{Bool} \rightarrow \text{Val Bool} \mid \text{Condition}$$

2.1.2 suite

Enfin, une grammaire LL(1) doit être factorisée.
IP reste donc à factoriser les règles qui ne peuvent.

↳ $L \rightarrow L \cdot I \rightarrow I; L \cdot I \mid I$

$L \rightarrow L \cdot L'$

$L' \rightarrow \cdot; L' \mid \epsilon$

↳ Affectation \rightarrow ident \leftarrow Expression | ident \leftarrow ValBool

Affectation \rightarrow ident \leftarrow Affectation'

Affectation' \rightarrow ~~Expression~~ Expression | ValBool.

↳ VarNum \rightarrow ident | ident [Expression] | entier

VarNum \rightarrow ident VarNum' | ~~entier~~ entier

VarNum' \rightarrow [Expression] | ϵ

2.1.2 suite

Avec tous ceci, l'on obtient donc une grammaire G' :

$S \rightarrow \text{program ident begin LI end}$

$LI \rightarrow I LI'$

$LI' \rightarrow ; LI | \epsilon$

$I \rightarrow \text{Affectation} | \text{while} | \text{for} | \text{if} | \text{break}$

$\text{Affectation} \rightarrow \text{ident} < \text{Affectation}'$

$\text{Affectation}' \rightarrow \text{Expression} | \text{ValBool}$

$\text{while} \rightarrow \text{while } B\text{Expression do LI end}$

$\text{for} \rightarrow \text{for ident from Valeur to Valeur do LI end}$

$\text{if} \rightarrow \text{if } B\text{Expression then LI else LI end}$

$\text{ValBool} \rightarrow \text{true} | \text{false}$

$B\text{Expression} \rightarrow \text{CombExpression } B\text{Expression}'$

$B\text{Expression}' \rightarrow \text{or } \cancel{\text{Expression}}' \cancel{\text{CombExpression}}' B\text{Expression}^3 | \epsilon$

$\text{CombExpression} \rightarrow \text{NotExpression And Expression}$

~~And~~ $\rightarrow \text{not NotExpression} | \text{Bool Expression}$

$\text{AndExpression} \rightarrow \text{and CombExpression} | \epsilon$

$\text{BoolExpression} \rightarrow \text{ValBool} | \text{Condition}$

$\text{Expression} \rightarrow T\text{Expression } Expression'$

$Expression' \rightarrow \text{OpPP Expression} | \epsilon$

$T\text{Expression} \rightarrow N\text{Expression } T\text{Expression}'$

$T\text{Expression}' \rightarrow \text{OpP } T\text{Expression} | \epsilon$

$N\text{Expression} \rightarrow \text{Var Num} | (Expression)$

$\text{OpP} \rightarrow * | /$

$\text{OpPP} \rightarrow + | -$

$\text{Var Num} \rightarrow \text{ident VarNum}' | \text{entier}$

$\text{VarNum}' \rightarrow [Expression] | \epsilon$

$\text{Valeur} \rightarrow \text{ident} | \cancel{\text{Entier}} \text{ entier}$

$\text{Condition} \rightarrow \text{Expression OpRel Expression}$

$\text{OpRel} \rightarrow \leq | < | \geq | > = | = | !=$

2.1.2 suite

Pour finir, il faut tester que notre nouvelle grammaire G' est bien LL(1) avec l'algorithme premier et suivant

NT	Premier	Suivant
.S	program	\$
LI	ident, while, for, if, break	end, else
LI'	$\epsilon, ;$	end, else
I	ident, ident, while, for, if, break	end, else, ;
Affectation	ident	end, else, ;
Affectation'	ident, entier, (, true, false	end, else, ;
while	while	end, else, ;
for	for	end, else, ;
If	if	end, else, ;
NotBool	true, false	and, or, do, end, then, else, ;
BExpression	ident, entier, (, true, false, not	do, then
BExpression'	ϵ, or	do, then
CombExpression	ident, entier, (, true, false, not	or, do, then
NotExpression	ident, entier, (, true, false, not	and, or, do, then
AndExpression	ϵ, and	or, do, then
OrExpression	ident, entier, (, true, false	and, or, do, then
Expression	ident, entier, (],), <=, <, >, >=, !=, and, or, do, end, then, else, ;
Expression'	$\epsilon, -$],), <=, <, >, >=, !=, and, or, do, end, then, else, ;
TExpression	ident, entier, (],), +, -, <=, <, >, >=, !=, and, or, do, end, then, else, ;
TExpression'	$\epsilon, *, /$],), +, -, <=, <, >, >=, !=, and, or, do, end, then, else, ;
NExpression	ident, entier, (],), *, /, <=, <, >, >=, !=, and, or, else, do, then, ;
OpP	$\epsilon, /$	ident, entier, (
OpPP	$\epsilon, -$	ident, entier, (
VarNum	ident, entier],), *, /, +, -, <=, <, >, >=, !=, and, or, do, end, then, else, ;
VarNum'	ϵ, E],), *, /, +, -, <=, <, >, >=, !=, and, or, do, end, then, else, ;
Valeur	ident, entier	do, to
Condition	ident, entier, (and, or, do, then
OpRel	$<=, <, >, >=, =, !=$	ident, entier, (