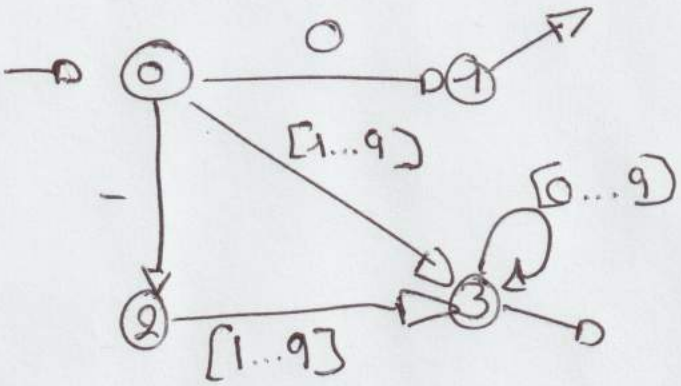
DI7

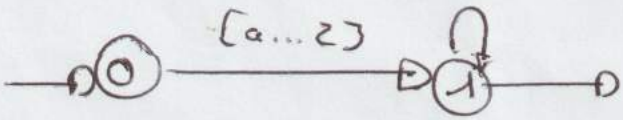
LANGUAGES ET AUTOMATES.

Partie 1

1. Automate d'etat fini minimal pour $0|[-]\varepsilon)([1..9][0..9]^*)$



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



Partie 2

2.1.1

La grammaire 6 n'est pas LL(1) car elle contient des règles recursives gauches :

↳ Expression → Expression opArithm Expression |
(Expression) |
VarNum

↳ BExpression → ValBool |
notBExpression |
BExpression and BExpression |
BExpression or BExpression |
condition.

2.1.2

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

↳ Expression → Expression OpArith Expression | (Expression) |
VarNum

Expression → (Expression) Expression' | VarNum Expression'
Expression' → OpArith Expression Expression' | ε

↳ BExpression → ValBool | not BExpression |
BExpression and BExpression |
BExpression or BExpression |
Condition

BExpression → ValBool BExpression' |
not BExpression BExpression' |
Condition BExpression'
BExpression' → or BExpression BExpression' |
and BExpression BExpression' |
ε

2.1.2 suite

On peut ensuite factoriser les règles obtenus.

Expression	Expression → TExpression Expression'
	TExpression → (Expression) | VarNum
	Expression' → OpArith Expression Expression' | ε
BExpression	BExpression → GBExpression BExpression'
	GBExpression → ValBool | not BExpression | Condition
BExpression'	BExpression' → TBExpression BExpression' | ε
	TBExpression → or BExpression | and BExpression
TBExpression	TBExpression → TBExpression' BExpression
	TBExpression' → 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 (+, -, ×, ÷).

L'ordre sera :

arithmétique : × et ÷, puis + et -

logique : not, puis and, puis enfin or

2.1.2 suite.

Transformation des operateurs arithmetiques

des règles concernées sont Expression, Expression' et TExpression

Expression → TExpression Expression'
Expression' → OpPP TExpression Expression' | ε
TExpression → NExpression TExpression'
TExpression' → OpP NExpression TExpression' | ε
NExpression → VarNum | (Expression)
OpPP → + | -
OpP → * | /

Transformation des operateurs ~~arithmetiq~~ logiques

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

BExpression → CombExpression BExpression'
BExpression' → or CombExpression BExpression' | ε
CombExpression → NotExpression AndExpression
AndExpression → and NotExpression AndExpression | ε
NotExpression → not NotExpression | ~~Bool~~ BoolExpression
Bool~~Expression~~Expression → Val Bool | Condition

2.1.2 suite

Enfin, une grammaire LL(1) doit etre factorisée.
Il reste donc à factoriser les règles qui le peuvent.

↳ LI → I ; LI | I

LI → I LI'
LI' → ; LI | ε

↳ Affectation → ident ← Expression | ident ← ValBool

Affectation → ident ← Affectation'
Affectation' → ~~Affect~~ Expression | ValBool

↳ VarNum → ident | ident [Expression] | entier

VarNum → ident VarNum' | ~~Entier~~ entier
VarNum' → [Expression] | ε

2.1.2 suite

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

S → program ident begin LI end
LI → I LI'
LI' → ; LI | ε
I → Affectation | While | For | If | break
Affectation → ident <- Affectation'
Affectation' → Expression | ValBool
While → while BExpression do LI end
For → for ident from Valeur to Valeur do LI end
If → if BExpression then LI else LI end
ValBool → true | false
BExpression → CombExpression BExpression'
BExpression' → or ~~BExpression' CombExpression~~ BExpression' | ε
CombExpression → NotExpression AndExpression
~~AndEx~~
NotExpression → not NotExpression | BoolExpression
AndExpression → and CombExpression | ε
BoolExpression → ValBool | Condition
Expression → TExpression Expression'
Expression' → OpPP Expression | ε
TExpression → NExpression TExpression'
TExpression' → OpP TExpression | ε
NExpression → VarNum | (Expression)
OpP → * | /
OpPP → + | -
VarNum → ident VarNum' | entier
VarNum' → [Expression] | ε
Valeur → ident | ~~Entier~~ entier
Condition → Expression OpRel Expression
OpRel → <= | < | > | >= | = | !=

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'	ε, ;	end, else
I	~~Affectation~~ 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, ;
VofBool	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
BoolExpression	ident, entier, (, true, false	and, or, do, then
Expression	ident, entier, (	],), <=, <, >, >=, =, !=, and, or, do, end, then, else, ;
Expression'	ε, -	],), <=, <, >, >=, =, !=, and, or, do, end, then, else, ;
TExpression	ident, entier, (	],), +, -, <=, <, >, >=, =, !=, and, or, do, then, else, ;
TExpression'	ε, *, /	],), +, -, <=, <, >, >=, =, !=, and, or, do, then, else, ;
NExpression	ident, entier, (	],), *, /, <=, <, >, >=, =, !=, and, or, else, do, then, ; +, -
OpP	*, /	ident, entier, (
OpPP	+, -	ident, entier, (
VarNum	ident, entier	],), *, /, +, -, <=, <, >, >=, =, !=, and, or, do, end, then, else, ;
VarNum'	ε, [	],), *, /, +, -, <=, <, >, >=, =, !=, and, or, do, end, then, else, ;
Valeur	ident, entier	do, to
Condition	ident, entier, (	and, or, do, then
OpRel	<=, <, >, >=, =, !=	ident, entier, (