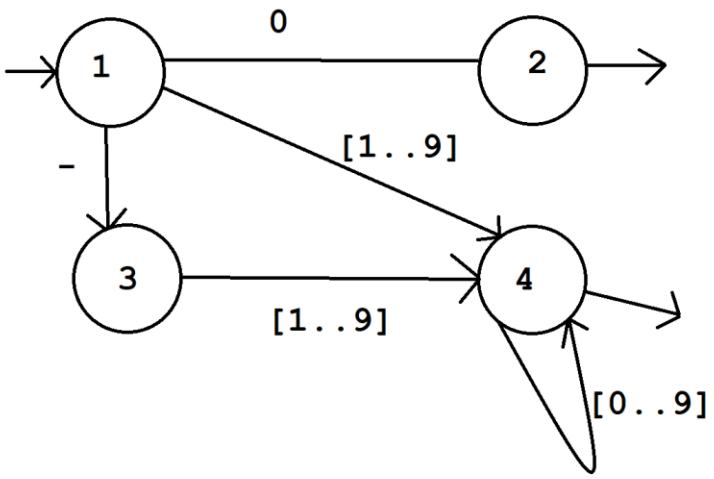
Ionas NEONAKIS L3 Ingé 2170664

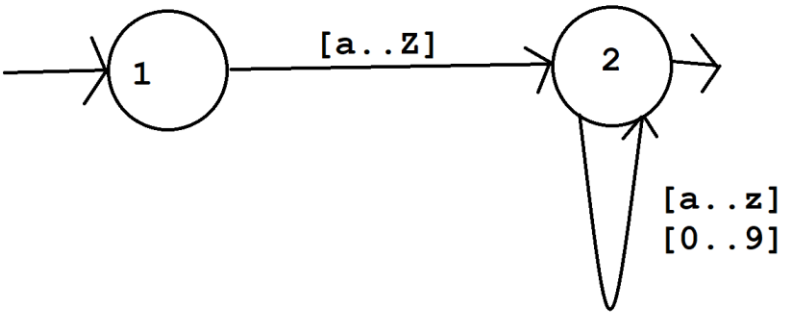
Réponses aux question du projet de langage algébrique

1. Analyse Lexicale

1)Automate pour : $0 \mid (- \mid \epsilon)\ ([1..9]\ [0..9]^*)$.



2) Automate pour $[a..z]\ ([a..z] \mid [0..9])^*$



2. Analyse syntaxique

2.1 Mise sous la fome LL(1)

1) G n'est pas LL(1) entre autres à cause de la récursivité gauche de certaines productions comme par exemple :

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

2)

Pour la rendre LL(1) il suffit de dérécursiver à gauche en tenant compte de la priorité des opérateurs arithmétiques et binaires puis à la fin de factoriser les productions.

On obtient donc à la fin la grammaire G' (Voir pages suivante):

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 -> TBExpression BExpression’

BExpression’ -> or TBExpression BExpression’ | ε

TBExpression -> FBExpression TBExpression’

TBExpression’ -> and FBExpression TBExpression’ | ε

FBExpression -> not FBExpression | GBExpression

GBExpression -> ValBool | Condition

Expression -> TExpression Expression’

Expression’ -> OpPasPrio TExpression Expression’ | ε

TExpression -> FExpression TExpression’

TExpression’ -> OpPrio FExpression TExpression’ | ε

FExpression -> VarNum | (Expression)

OpPrio -> * | /

OpPasPrio -> + | -

VarNum ->ident VarNum’ | entier

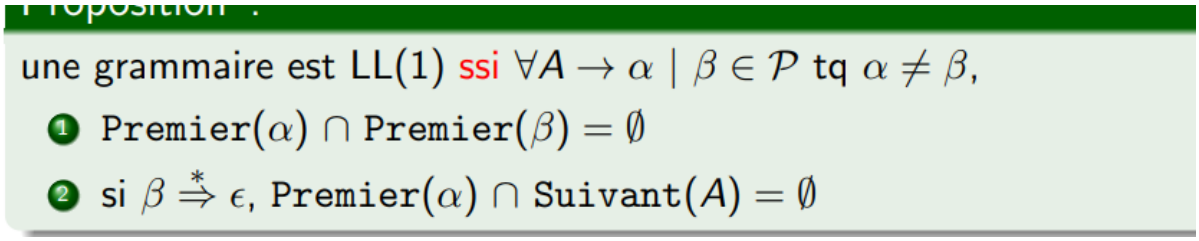
VarNum’ -> [Expression] | ε

Valeur -> ident | entier

Condition -> Expression OpRel Expression

OpRel -> <= | < | > | >= | = | ! =

Pour s'assurer que la grammaire est LL(1) il suffit de faire ces vérifications :



On a donc :

Première regle :

LI' -> ; LI | ε

Premier(-> ; LI) ∩ Premier(ε) = ∅

I -> Affectation | While | For | If | break

Premier(Affectation) ∩ Premier(While) Premier(For I) ∩ Premier(If) Premier break) = ∅

Affectation' -> Expression | ValBool

Premier(Expression) ∩ Premier(ValBool) = ∅

ValBool -> true | false

Premier (true) ∩ Premier(false) = ∅

BExpression' -> or TBExpression BExpression' | ε

Premier (or TBExpression BExpression') ∩ Premier(ε) = ∅

TBExpression' -> and FBExpression TBExpression' | ε

Premier (and FBExpression TBExpression') ∩ Premier(ε) = ∅

FBExpression -> not FBExpression | GBExpression

Premier (not FBExpression) ∩ Premier(GBExpression) = ∅

GBExpression -> ValBool | Condition

Premier (ValBool) ∩ Premier(Condition) = ∅

Expression' -> OpPasPrio TExpression Expression' | ε

Premier (OpPasPrio TExpression Expression') ∩ Premier(ε) = ∅

TExpression' -> OpPrio FExpression TExpression' | ε

Premier (OpPrio FExpression TExpression') ∩ Premier(ε) = ∅

FExpression -> VarNum | (Expression)

Premier (VarNum) ∩ Premier((Expression)) = ∅

OpPrio -> * | /

Premier (*) ∩ Premier(/) = ∅

OpPasPrio -> + | -

Premier (+) ∩ Premier(-) = ∅

VarNum ->ident VarNum' | entier

Premier (ident VarNum') ∩ Premier(entier) = ∅

VarNum' -> [Expression] | ε

Premier ([Expression]) ∩ Premier(ε) = ∅

Valeur -> ident | entier

Premier (ident) ∩ Premier(entier) = ∅

OpRel -> <= | < | > | >= | = | ! =

Premier (<=) ∩ Premier(<) ∩ Premier(>) ∩ Premier(>=) ∩ Premier(=)∩ Premier(!=) = ∅

2e regle

LI' -> ; LI | ε

Premier(; LI) ∩ Suivant(LI') = ∅

BExpression' -> or TBExpression BExpression' | ε

Premier(or TBExpression BExpression') ∩ Suivant(BExpression') = ∅

TBExpression' -> and FBExpression TBExpression' | ε

Premier(and FBExpression TBExpression') ∩ Suivant(TBExpression') = ∅

Expression' -> OpPasPrio TExpression Expression' | ε

Premier(OpPasPrio TExpression Expression') ∩ Suivant(Expression') = ∅

TExpression' -> OpPrio FExpression TExpression' | ε

Premier->(OpPrio FExpression TExpression') ∩ Suivant(TExpression') = ∅

VarNum' -> [Expression] | ε

Premier([Expression]) ∩ Suivant(VarNum') = ∅

Donc la grammaire G' est bien LL(1)