Projet de compilation

Rapport intermédiaire

Loïc Bertrand
Timon Fugier
Tony Zhou
Zineb Ziani El Idrissi

TELECOM Nancy 2ème Année
2019 – 2020

Table des matières

1 Grammaire du langage

```
/*
    ANTLR grammar and AST based on the original Algol60 grammar.
    Developped by  Loïc Bertrand, Timon Fugier, Tony Zhou and
    Zineb Ziani El Idrissi.
*/

grammar Algol60;

options {
    output = AST;
    backtrack=false;
    k=1;
    ASTLabelType=CommonTree;
}

tokens {
    ROOT;               // Program root
    BLOCK;              // Block of code
    VAR_DEC;            // Variable declaration
    PROC_DEC;           // Procedure declaration
    PROC_HEADING;       // Procedure heading
    PARAMS_DEC;         // Parameters
    ID_LIST;            // List of identifiers
    VALUE_PART;         // Procedure's parameters passed by value
    PARAM_PART;         // Procedure's parameters passed by reference
    SPEC_PART;          // Procedure's parameters names
    ARG_TYPE;           // Type of a procedure's argument
    PARAM_LIST;         // List of parameters
    PROC_CALL;          // Procedure call
    IF_STATEMENT;       // If statement
    IF_DEF;             // First part of an if statement
    THEN_DEF;           // Then part of an if statement
    ELSE_DEF;           // Else part of an if statement
    FOR_CLAUSE;         // For loop
    INIT;               // Initialization part of for loop
    STEP;               // Step part of for loop
    UNTIL;              // Condition part of for loop
    DO;                 // Code part of for loop
```

```
    WHILE_CLAUSE;       // While loop
    CONDITION;          // While loop condition
    ASSIGNMENT;         // Assignment
    ARRAY_DEC;          // Array Declaration
    BOUND_DEC;          // Declaration of the boundaries of an array
    BOUND_LIST;         // List of BOUND_DEC
    ARRAY_ASSIGNMENT;   // Assignment of an array
    INDICES;            // Indices of an element of an array
    MULT;               // Multiplication
    DIV;                // Division
    INT_DIV;            // Integer division
    ADD;                // Addition
    MINUS;              // Substraction
    TERM;               // Term in expression
    FACTOR;             // Factor in expression
    LABEL_DEC;          // Label declaration
    GOTO;               // Goto statement
    POW;                // Power
    INT;                // Integer
    POW_10;             // Scientific notation
    ARRAY_CALL;         // Access to a value of an array
    STR;                // String
    REAL;               // Real number
}

@parser::header {
package eu.telecomnancy;
}

@lexer::header {
package eu.telecomnancy;
}

// PARSER RULES

prog:   block EOF -> ^(ROOT block)
    ;

block
    :   'begin' statement (';' statement)* 'end' -> ^(BLOCK statement+)
    ;

// Statement

statement
    :   declaration
    |   goto_statement
    |   if_clause
```

```
    |   for_clause
    |   while_clause
    |   block
    |   identifier! id_statement_end[$identifier.tree]
    ;

id_statement_end[CommonTree id]
    :   procedure_call_end[$id]
    |   assignment_end[$id]
    |   label_dec_end[$id]
    ;

// Label / goto statement

goto_statement
    :   'goto' identifier -> ^(GOTO identifier)
    ;

label_dec_end[CommonTree id]
    :   ':' -> ^(LABEL_DEC {$id})
    ;

// Declaration

declaration
    :   TYPE! type_declaration_end[$TYPE]
    |   procedure_declaration_no_type
    ;

type_declaration_end[Token type]
    :   variable_declaration_end[$type]
    |   procedure_declaration_end[$type]
    ;

variable_declaration
    :   TYPE! identifier_list_head[$TYPE]
    ;

variable_declaration_end[Token type]
    :   identifier_list_head[$type]
    ;

identifier_list_head[Token type]
    :   identifier_list -> ^(VAR_DEC {new CommonTree($type)} identifier_list)
    |   'array' identifier '[' boundaries(',' boundaries)* ']'
        -> ^(ARRAY_DEC {new CommonTree($type)}
                                        identifier ^(BOUND_LIST boundaries+))
    ;
```

```
// Procedure declaration

procedure_declaration
    :   TYPE? procedure_declaration_end[$TYPE]
    ;

procedure_declaration_end[Token type]
    :   'procedure' procedure_heading procedure_body
        -> ^(PROC_DEC {new CommonTree($type)} procedure_heading procedure_body)
    ;

procedure_declaration_no_type
    :   'procedure' procedure_heading procedure_body
        -> ^(PROC_DEC procedure_heading procedure_body)
    ;

procedure_heading
    :   identifier formal_parameter_part ';' value_part specification_part
        -> ^(PROC_HEADING formal_parameter_part? value_part? specification_part?)
    ;

formal_parameter_part
    :   '(' identifier_list ')' -> ^(PARAM_PART identifier_list)
    |
    ;

identifier_list
    :   identifier ( ',' identifier )* -> ^(ID_LIST identifier*)
    ;

value_part
    :   'value' identifier_list ';' -> ^(VALUE_PART identifier_list)
    |
    ;

specification_part
    :   ( TYPE identifier_list ';' )*
        -> ^(SPEC_PART ^(ARG_TYPE TYPE identifier_list)*)
    ;

procedure_body
    :   block
    ;

// Procedure call

procedure_call
```

```
    :   identifier! procedure_call_end[$identifier.tree]
    ;

procedure_call_end[CommonTree id]
    :   '(' actual_parameter_list ')'
        -> ^(PROC_CALL {$id} actual_parameter_list)
    ;

actual_parameter_list
    :   arithmetic_expression ( ',' arithmetic_expression )*
        -> ^(PARAM_LIST arithmetic_expression*)
    |   -> ^(PARAM_LIST)
    ;

// Assignment

assignment
    :   identifier! assignment_end[$identifier.tree]
    ;

assignment_end[CommonTree id]
    :   ':=' arithmetic_expression -> ^(ASSIGNMENT {$id} arithmetic_expression)
    |   '[' bound (',' bound)* ']' ':=' arithmetic_expression
        -> ^(ARRAY_ASSIGNMENT {$id} ^(INDICES bound+) arithmetic_expression)
    ;


boundaries
    :   bound ':' bound -> ^(BOUND_DEC bound bound)
    ;

bound
    :   arithmetic_expression
    ;

array_call_end[CommonTree id]
    :   '[' actual_parameter_list ']'
        -> ^(ARRAY_CALL {$id} actual_parameter_list)
    ;

// Expression

expression
    :   string
    |   integer
    |   identifier! id_expression_end[$identifier.tree]
    |   scientific_expression
    ;
```

```
id_expression_end[CommonTree id]
    :   array_call_end[$id]
    |   procedure_call_end[$id]
    |   -> {$id}
    ;

// Arithmetic

arithmetic_expression
    :   term! arithmetic_expression_end[$term.tree]
    ;

arithmetic_expression_end[CommonTree t2]
    :   '+' arithmetic_expression -> ^(ADD {$t2} arithmetic_expression?)
    |   '-' arithmetic_expression -> ^(MINUS {$t2} arithmetic_expression?)
    |   -> {$t2}
    ;

term
    :   expression! term1[$expression.tree]
    ;

term1[CommonTree t2]
    :   '*' term -> ^(MULT {$t2} term?)
    |   '/' term -> ^(DIV {$t2} term?)
    |   '//' term -> ^(INT_DIV {$t2} term?)
    |   '**' term -> ^(POW {$t2} term?)
    |   -> {$t2}
    ;

// If clause

if_clause
    :   'if' logical_statement 'then' statement
                                  (options{greedy=true;}:'else' statement)?
        -> ^(IF_STATEMENT ^(IF_DEF logical_statement)
                            ^(THEN_DEF statement) ^(ELSE_DEF statement)*)
    ;

// For clause

for_clause
    :   'for' assignment 'step' expression 'until' expression 'do' statement
        -> ^(FOR_CLAUSE ^(INIT assignment) ^(STEP expression)
                                      ^(UNTIL expression) ^(DO statement))
    ;
```

```
// While clause

while_clause
    :   'while'  logical_statement 'do' statement
        -> ^(WHILE_CLAUSE ^(CONDITION logical_statement) ^(DO statement))
    ;

logical_statement
    :   arithmetic_expression! logical_statement_end[$arithmetic_expression.tree]
    |   LOGICAL_VALUE
    ;

logical_statement_end[CommonTree t2]
    :   boolean_operator arithmetic_expression
        -> ^(boolean_operator {$t2} arithmetic_expression)
    ;

boolean_operator
    :   RELATIONAL_OPERATOR
    |   LOGICAL_OPERATOR
    ;

// Intermediate parser rules

integer
    :   '-'? INTEGER -> ^(INT '-'? INTEGER)
    ;

scientific_expression
    :   REAL! scientific_expression_end[$REAL]
    ;

scientific_expression_end[Token real]
    :   '#' INTEGER -> ^(POW_10 {new CommonTree($real)} INTEGER)
    |   -> ^(REAL {new CommonTree($real)})
    ;

string
    :   STRING -> ^(STR STRING)
    ;

identifier
    :   IDENTIFIER //-> ^(ID IDENTIFIER)
    ;

// LEXER RULES

TYPE:   'real'
```

```
    |   'integer'
    |   'boolean'
    |   'string'
    ;

COMMENT
    :   'comment' ~( ';' )* ';'
        // Ignore comments (not in the AST)
        { $channel=HIDDEN; }
    ;

STRING
    :   '"' ~( '"' | '\r' | '\n' )* '"'
        // Strips the string from its quotes in the lexer
        // https://theantlrguy.atlassian.net/wiki/spaces/
                                ANTLR3/pages/2687006/How+do+I+strip+quotes
        { setText(getText().substring(1, getText().length() - 1)); }
    ;

LOGICAL_VALUE
    :   'true'
    |   'false'
    ;

INTEGER
    :   ('1'..'9')('0'..'9')*
    |   '0'
    ;

REAL
    :   ('0'..'9')*'.'('0'..'9')*
    ;

IDENTIFIER
    :   ('a'..'z'|'A'..'Z'|'_')('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
    ;

RELATIONAL_OPERATOR
    :   '<'
    |   '<='
    |   '='
    |   '<>'
    |   '>'
    |   '>='
    ;

LOGICAL_OPERATOR
    :   '<=>'
```

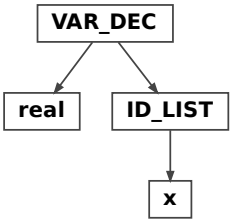
```
    |   '=>'
    |   '\\/'
    |   '/\\'
    |   '~'
    ;

WS  :   (' '|'\t'|'\r'|'\n')+
        // Ignore whitespace (not in the AST)
        { $channel=HIDDEN; }
    ;
```

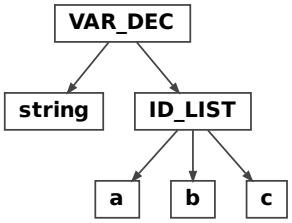
2 Structure de l'arbre abstrait

2.1 Déclarations

2.1.1 Déclaration de variable



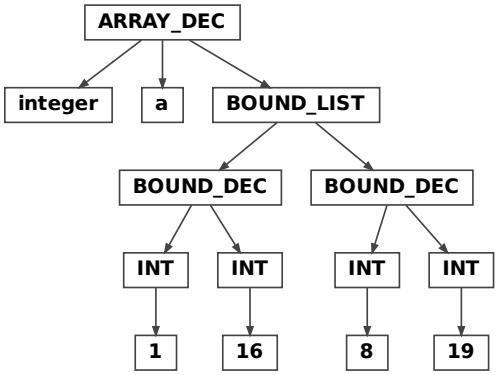

```
real x
```




```
string a, b, c
```

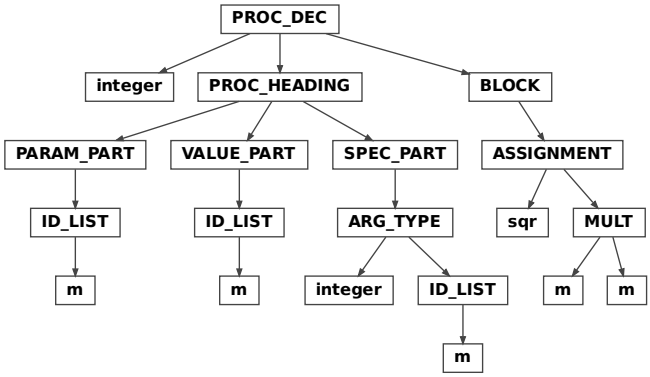
2.1.2 Déclaration de tableau

```
integer array a[1:16, 8:19]
```



2.1.3 Déclaration de procédure

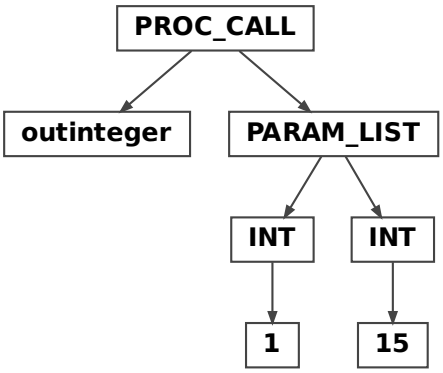
```
integer procedure sqr(m);
  value m; integer m;
begin
  sqr := m * m
end
```



2.2 Appels

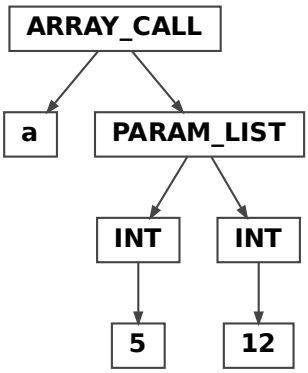
2.2.1 Appel de procédure

```
outinteger(1, 15)
```



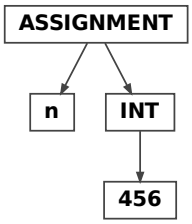
2.2.2 Appel de tableau

```
a[5, 12]
```

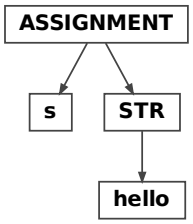


2.3 Affectations

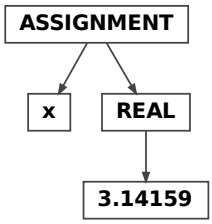
2.3.1 Affectation




```
n := 456
```



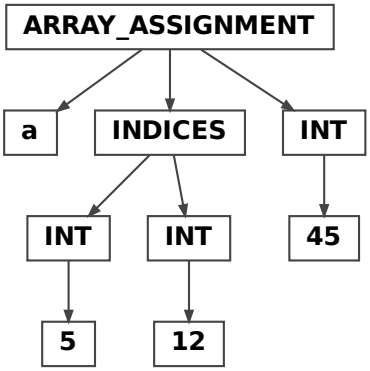

```
s := "hello"
```




```
x := 3.14169
```

2.3.2 Affectation dans un tableau

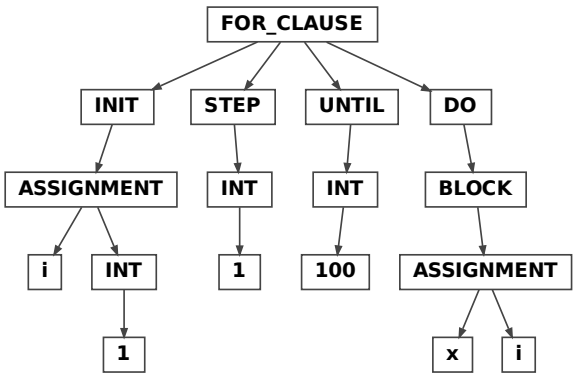
```
a[5, 12] := 45
```



2.4 Boucles et conditions

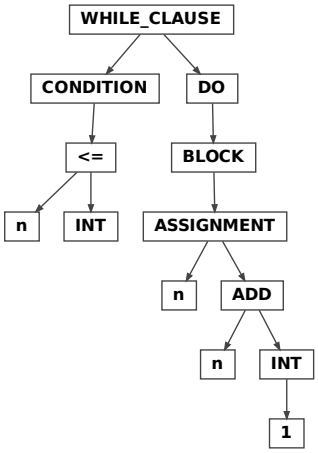
2.4.1 Boucle `for`

```
for i := 1 step 1 until 100 do
begin
  x := i
end
```



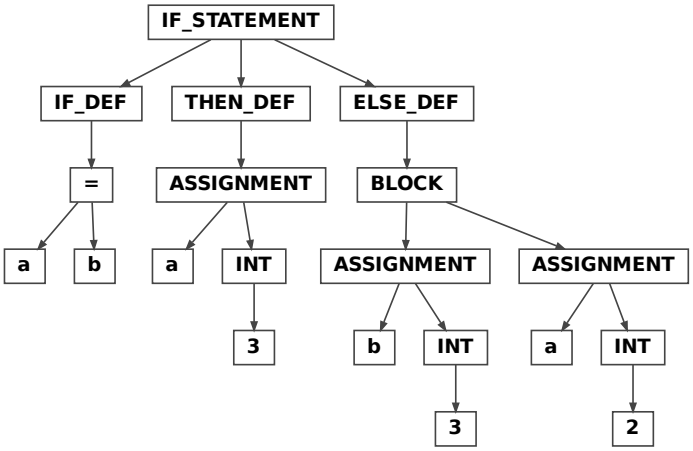
2.4.2 Boucle `while`

```
while i <= 10 do
begin
  n := n + 1
end
```



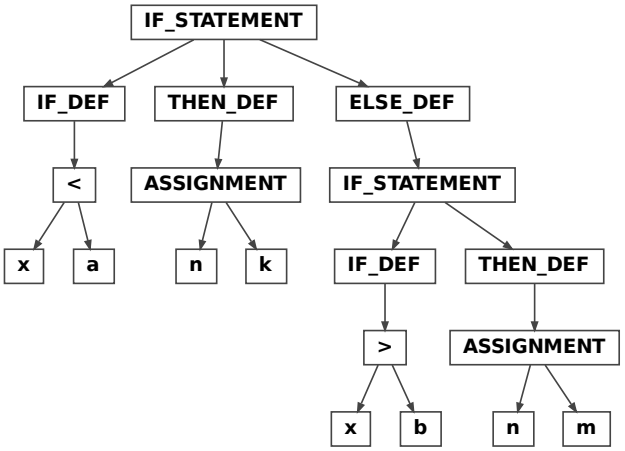
2.4.3 Clause `if` avec un block

```
if a = b then
  a := k
else
begin
  b := l;
  a := m
end
```



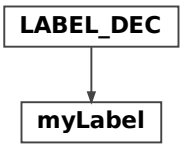
2.4.4 Clause `if` avec imbrication

```
if x < a then
  n := k
else if x > b then
  n := m
```

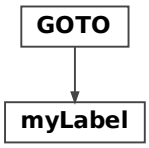


2.5 Autres

2.5.1 Déclaration de label et `goto`



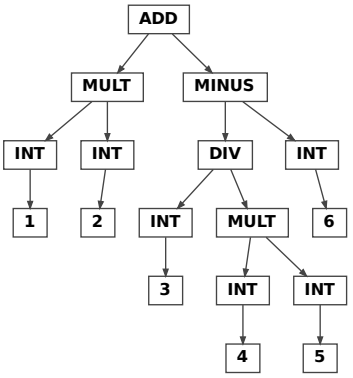

```
myLabel:
```




```
goto myLabel
```

2.5.2 Expression arithmétique

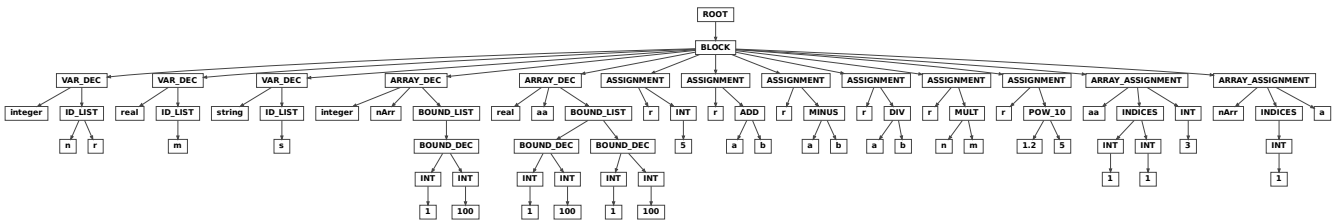
```
1 * 2 + 3 / 4 * 5 - 6
```



3 Jeux d'essais

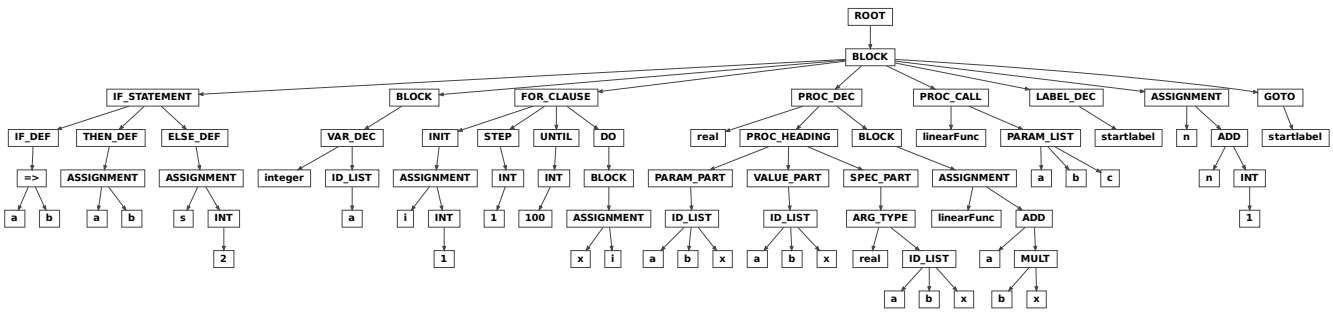
3.1 Test de base

```
1  begin
2    comment declaration et affectation de variables;
3
4    integer n,r;
5    real m;
6    string s;
7    integer array nArr[1:100];
8    real array aa[1:100,1:100];
9
10   r := 5 ;
11   r := a+b ;
12   r := a-b ;
13   r := a/b ;
14   r := n*m ;
15   r := 1.2#5 ;
16   aa[1,1] := 3;
17   nArr[1] := a
18
19 end
```



3.2 Test avancé

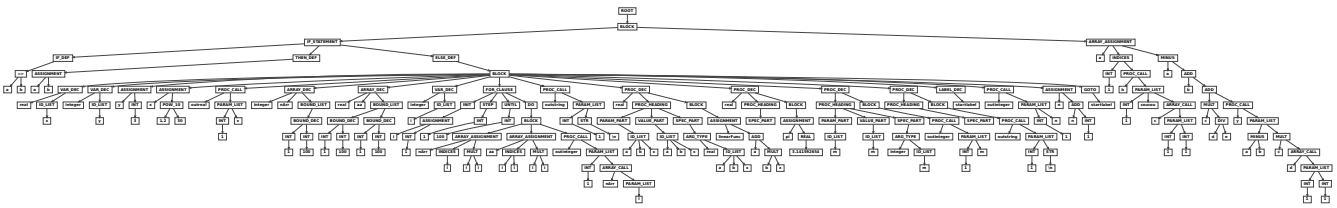
```
1  begin
2    comment IF ELSE ;
3    if a=>b then
4      a:=b
5    else
6      s:=2;
7
8    comment Block imbrique ;
9    begin
10     integer a
11   end;
12
13   comment BOUCLE FOR;
14   for i := 1 step 1 until 100 do
15   begin
16     x := i
17   end ;
18
19   comment DEFINITION DE PROCEDURE
20     (avec type de retour et arguments);
21   real procedure linearFunc(a, b, x);
22   value a, b, x; real a, b, x;
23   begin
24     linearFunc := a + b*x
25   end;
26
27   comment Appel de proc ;
28   linearFunc(a,b,c);
29
30   comment GOTO et LABEL ;
31   startlabel:;
32     n := n+1;
33   goto startlabel
34 end
```



3.3 Test complet

```
1  begin
2
3    comment IF ELSE ;
4    if a=>b then
5      a:=b
6    else
7
8      comment BLOCK IMBRIQUE ;
9      begin
10
11       comment scientific notation (using #);
12       real x;
13       integer y;
14       y := 3;
15       x := 1.2#30;
16
17       comment Appel de procedure ;
18       outreal(1,x);
19
20       comment array, matrix with square numbers ;
21       integer array nArr[1:100];
22       real array aa[1:100,1:100];
23       integer i;
24
25       comment BOUCLE FOR;
26       for i := 1 step 1 until 100 do
27       begin
28         nArr[i] := i*i;
29         aa[i,i] := i*i ;
30         outinteger (1, nArr[i])
31       end ;
32       outstring (1,"\n");
33
34       comment DEFINITION DE PROCEDURE
35         (avec type de retour et arguments);
36       real procedure linearFunc(a, b, x);
37       value a, b, x; real a, b, x;
38       begin
39         linearFunc := a + b*x
40       end;
41
42       comment DEFINITION DE PROCEDURE
43         (avec type de retour et sans arguments);
44       real procedure pi;
45       begin
46         pi := 3.141592654
47       end;
48
49       comment DEFINITION DE PROCEDURE
```

```
50          (sans type de retour et avec arguments);
51        procedure myPrint(m);
52        value m; integer m;
53        begin
54          outinteger(1,m)
55        end;
56
57        comment DEFINITION DE PROCEDURE
58          (sans type de retour et sans arguments);
59        procedure cr;
60        begin
61          outstring (1,"\n")
62        end;
63
64        comment GOTO et LABEL ;
65        startlabel:;
66          outinteger (1,n);
67          n := n+1;
68        goto startlabel
69
70      end;
71
72    comment assignement complique ;
73    a[1,b(1,coucou,c[1,1])] := a-b+c*d/e+y(a-b,c*d[1,1])
74
75  end
```



4 Gestion de projet

4.1 Fiche projet

4.2 Distribution des rôles

— Loïc Bertrand : Chef de projet
— Timon Fugier : Responsable code
— Tony Zhou : Responsable LaTeX
— Zineb Ziani El Idrissi : Responsable gestion de projet

4.3 Matrice RACI

	Loïc	**Timon**	**Tony**	**Zineb**
Gestion de projet	ACI	CI	CI	R
Écriture de la grammaire	R	ACI	CI	CI
Génération de l'AST	ACI	R	CI	CI
Rapport	ACI	CI	R	CI
Tests	ACI	R	CI	CI

Rappel : R = Réalisateur, A = Approbateur, C = Consulté, I = Informé.

4.4 Compte-rendus rédigés

Compte-rendus de réunions rédigés en ***Annexe A****.*

4.5 Outils utilisés

Pour optimiser notre gestion de projet, nous avons utilisé ces outils :

— **Google Sheets** pour renseigner les temps de travail de chacun puis convertir la feuille de calcul en tableau LaTeX
— **ShareLatex** pour écrire notre rapport et nos comptes-rendus de réunion de manière collaborative.

4.6 Diagramme de Gantt

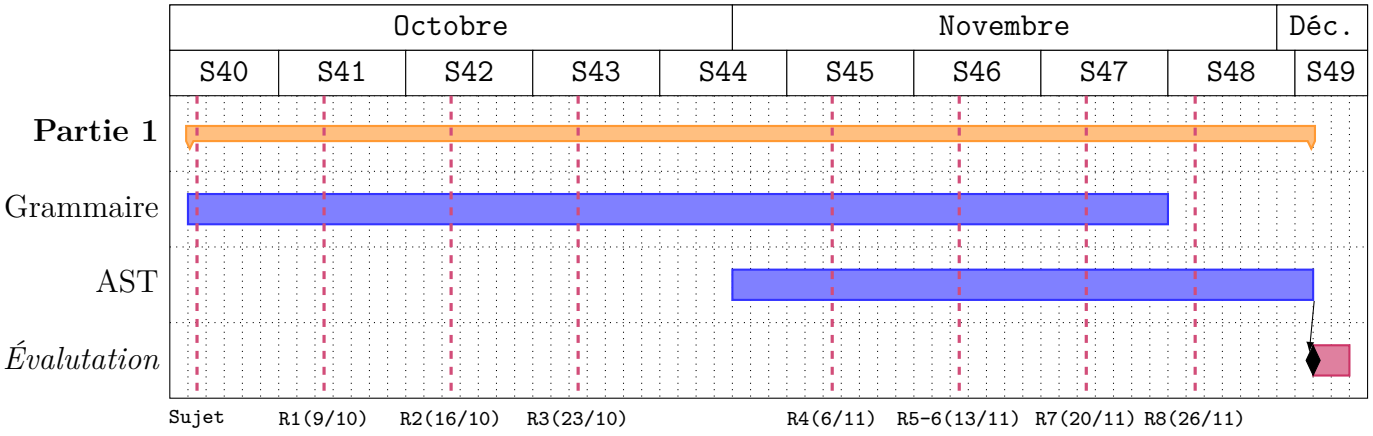


FIGURE 4.1 – Diagramme de Gantt (1e partie du projet)

4.7 Matrice SWOT

Forces : — Formation préalable par un MOOC gestion de projet. — Documentation fournie pour la compréhension du langage ALGOL 60 . — Communication fréquente dans l'équipe sur Messenger . — Formation préalable par les modules MI1 , MI2 et TRAD 1 .	***Faiblesses :*** — Faible connaissance du langage ALGOL 60 . — Découverte de l'outil ANTLR
Opportunités : — Premier projet ou on réalise un code compilateur passant de tout les étapes de la grammaire jusqu'à la génération du code finale . — Plateforme de gestion des versions : `gitlab.telecomnancy.univ-lorraine.fr`	***Menaces :*** — Plusieurs projets a réaliser en même temps . — Exactitude de la grammaire .

TABLE 4.2 – Matrice SWOT

4.8 Évaluation de la répartition du travail

Membre	Tâche	Date	Temps
Loïc	Participation réunion 1	10/10/2019	02h00
Tony	Participation réunion 1	10/10/2019	02h00
Zineb	Participation réunion 1	10/10/2019	02h00
Timon	Participation réunion 1	10/10/2019	02h00
Zineb	Élaboration de la matrice SWOT	15/10/2019	01h00
Loïc	Création du diagramme de Gantt	15/10/2019	00h40
Loïc	Participation réunion 2	17/10/2019	02h00
Tony	Participation réunion 2	17/10/2019	02h00
Zineb	Participation réunion 2	17/10/2019	02h00
Timon	Participation réunion 2	17/10/2019	02h00
Loïc	Nettoyage et vérification de la grammaire	19/10/2019	00h30
Loïc	Installation de Gradle sur le projet	19/10/2019	01h30
Tony	Dérécursivation	22/10/2019	01h00
Loïc	Participation réunion 3	23/10/2019	04h00
Tony	Participation réunion 3	23/10/2019	04h00
Zineb	Participation réunion 3	23/10/2019	04h00
Timon	Participation réunion 3	23/10/2019	04h00
Loïc	Écriture d'une grammaire simplifiée qui compile	27/10/2019	01h30
Loïc	Grammaire simplifiée ajoutée au projet Gradle	29/10/2019	01h30
Loïc	Participation réunion 4	06/11/2019	04h00
Tony	Participation réunion 4	06/11/2019	04h00
Zineb	Participation réunion 4	06/11/2019	04h00
Timon	Participation réunion 4	06/11/2019	04h00
Loïc	Révision du diagramme de Gantt	07/11/2019	00h10
Loïc	Participation réunion 5	13/11/2019	02h00
Tony	Participation réunion 5	13/11/2019	02h00
Zineb	Participation réunion 5	13/11/2019	02h00
Timon	Participation réunion 5	13/11/2019	02h00
Loïc	Participation réunion 6	13/11/2019	04h00
Tony	Participation réunion 6	13/11/2019	04h00
Zineb	Participation réunion 6	13/11/2019	04h00
Timon	Participation réunion 6	13/11/2019	04h00
Loïc	Participation réunion 7	20/11/2019	04h00
Tony	Participation réunion 7	20/11/2019	04h00
Zineb	Participation réunion 7	20/11/2019	04h00
Timon	Participation réunion 7	20/11/2019	04h00
Loïc	Structure de l'AST dans le rapport	01/12/2019	00h45
Timon	Création de tests pour l'évaluation	28/11/2019	01h00
Tony	Modification minime AST pour les booléens	28/11/2019	00h30
Zineb	Correction de l'AST concernant un bug (INT)	02/12/2019	00h30

4.9 Répartition des tâches au sein du groupe

Tâche	**Partie**	**Membre**
Procédure	Déclaration	Timon
	Appel	Loïc
	Déclaration multiple	Loïc
	Type de retour	Loïc
Structure	If... then... else...	Tony, Zineb
	Boucle For	Tony, Zineb
	Tableaux	Timon
	Boucle While	Tony, Zineb
	Label et Goto	Timon
Opération	Booléen	Tony, Zineb
	Arithmétique	Tony, Zineb

Bibliographie

[1] Erik Schoenfelder. Syntax the Algorithmic Language Algol 60. http://www.csci.csusb.edu/dick/samples/algol60.syntax.html.

[2] SDS. SOS ALGOL 60 REFERENCE MANUAL. http://www.bitsavers.org/pdf/sds/9xx/lang/900699C_Algol60_Ref_Nov66.pdf.

[3] Athafoud & Lucas Trzesniewski. ANTLR Priority rules. https://riptutorial.com/antlr/example/11235/priority-rules.

[4] Bond. Algol 60 interpreter. https://www.bertnase.de/a60/a60.html.

[5] David A. Plaisted. Algol 60 grammar in BNF. https://www.cs.unc.edu/~plaisted/comp455/Algol60.pdf.

[6] hlevkin. Algol 60 Forever ! http://www.algol60.org/. Site dédié au language Algol60.

[7] J.W. Backus, F.L. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur, A.J. Perlis, H. Rutishauser, K. Samelson, B. Vauquoi, J.H. Wegstein, A. van Wijngaarden, M. Woodger. Revised report of algorithmic language algol 60. http://www.algol60.org/reports/algol60_rr.pdf. Édité par Peter Naur.

[8] Lamprecht. ALGOL 60 References. https ://www.masswerk.at/algol60/.

[9] LeGodais Tristan. Algol 60 grammar (tncy). https://tristoon.github.io/algol60-grammar-tncy/.

[10] Rupert Lane. ALGOL 60 Features. https://try-mts.com/algol-60-language-features/.

[11] Wikipedia. Algol 60 . https://en.wikipedia.org/wiki/ALGOL_60.

Annexes

A Compte-rendus de réunion rédigés

Compte-rendu de réunion 1

Mercredi 9 octobre 2019, 14h-16h

Type : Réunion de lancement

Lieu : Salle PI

Présents : Loïc Bertrand, Timon Fugier, Tony Zhou, Zineb Ziani El Idrissi *(Secrétaire)*

Absents : Aucun absent

Ordre du jour :

- Prise de connaissance du sujet
- Création du répertoire Git
- Attribution des rôles

Informations échangées :

- Algol W est un langage basé sur Algol 60
- Échéances du projet : le premier rendu est pour le 15 janvier 2020 , devra inclure la grammaire , l'arbre abstrait , jeux d'essais et de la gestion projet .
- Rôles au sein du groupe :
  - Loïc : Chef de projet
  - Zineb : Responsable gestion de projet
  - Timon : Responsable code
  - Tony : Responsable LaTeX
- Documentation du langage : ajout des documents utiles dans la bibliographie du rapport (grammaire, syntaxe du language)

Actions à suivre :

- Apprentissage du langage Algol
- Matrice SWOT : Timon, Zineb
- Diagramme de GANTT : Loïc
- Étude de la grammaire : Zineb, Tony, Timon, Loïc

Date de la prochaine réunion : Mercredi 16 octobre 2019

Compte-rendu de réunion 2

Mercredi 16 octobre 2019, 14h-16h

Type : Réunion d'avancement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Timon Fugier, Tony Zhou, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- Faire l'état des lieux de la gestion de projet (fixer les rôles et responsabilités de chacun)
- Étudier des morceaux de programmes en Algol60 (fournis dans les ressources du sujet)
- Écrire la grammaire (ou au moins une portion de la grammaire) en format ANTLR, en LL(1)

Informations échangées :

- Consignes données en amphithéâtre :
  - Récupérer la grammaire d'Algol60 et la rendre LL(1) (au format EBNF)
  - Comprendre le langage pour pouvoir construire un arbre abstrait
- Pour rendre la grammaire LL(1), nous avons enlevé les ambiguïtés et les récursivités à gauche.

Actions à suivre :

- Terminer la correction de la grammaire pour la rendre LL(1)
- Chercher des symboles équivalents pour les caractères spéciaux de la grammaire (à partir d'exemples de code Alg60, notamment ceux fournis sur le site `www.algol60.org` (Lego pieces))
- Matrice RACI : Zineb, Tony
- Commencer à tester (au moins des portions de grammaire) sur ANTLR
- Appliquer Gradle sur le projet (se référencer au sujet du devoir de PCD)

Date de la prochaine réunion : Mercredi 23 octobre 2019

Compte-rendu de réunion 3

Mercredi 23 octobre 2019, 14h-17h

Type : Réunion d'avancement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Timon Fugier, Tony Zhou, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- Mise au point sur les caractères spéciaux et leur remplacement
- Mettre les terminaux (rules) en majuscules
- Placement de la grammaire sur le dépôt et tentatives de compilation
- Ajout de fichiers exemples dans `src/test/antlr` (récupérer des programmes donnés en exemple sur `www.algol60.org`, qui devront pouvoir être compilés avec succés par notre comilateur

Informations échangées :

- L'Échéance de la première partie du projet est décalée de la semaine 46 à la semaine 48
- Mettre les terminaux en majuscule et résoudre les erreurs de syntaxe

Actions à suivre :

- Aboutir à une sous-grammaire qui compile.
- Tester les exemples fournis en ALGOL60.

Date de la prochaine réunion : Mercredi 6 novembre 2019

Compte-rendu de réunion 4

Mercredi 6 novembre 2019, 14h-18h

Type : Réunion d'avancement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Timon Fugier, Tony Zhou, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

— Travailler à partir du début de grammaire créé pendant les vacances
— Zineb et Tony : ajouter `if_clause` et `for_loop`
— Timon et Loïc : ajouter `procedure_declaration` et `procedure_call`

Informations échangées :

— Ajouter des éléments du langage Algol60 au fur et à mesure dans notre grammaire nous permet de comprendre chacun de ces éléments. Cela nous sera très utile pour réaliser l'analyse sémantique ultérieurement.
— Pour tester notre grammaire, la commande à effctuer est : `./gradlew run --console=plain < fichier_a_tester`.

Actions à suivre :

— Corriger les quelques warnings concernant les chemins multiples
— Faire l'AST concernant les `if_clause`
— Comprendre précisément le fonctionnement les `if_clause` (fin de condition, imbrication de `else`...)
— Dans le rapport, faire une description de certains éléments de langage ajoutés à la grammaire (déclaration de procédure, arguments passés par référence, par valeur, if/else...)

Date de la prochaine réunion : Mercredi 13 novembre 2019

Compte-rendu de réunion 5

Mercredi 13 novembre 2019, 8h-10h

Type : Réunion d'avancement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Timon Fugier, Tony Zhou, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

— Tony et Zineb : corriger `if_clause`
— Loïc : corriger les problèmes d'ambiguïté de la grammaire

Informations échangées :

— Résolution du problème du *Dangling else* à l'aide de l'instruction ANTLR : `options{greedy=true;}`:

Actions à suivre :

— Tony et Zineb : ajouter `for_clause`
— Timon : ajouter `array_assignement`
— Loïc : former un AST convenable sur des règles séparées (pour éviter les ambiguités)

Date de la prochaine réunion : Le jour même, 14h-18h

Compte-rendu de réunion 6

Mercredi 13 novembre 2019, 14h-18h

Type : Réunion d'avancement

Lieu : Salle N0.42

Présents : Loïc Bertrand, Timon Fugier, Tony Zhou, Zineb Ziani El Idrissi *(Secrétaire)*

Absents : Aucun absent

Ordre du jour :

- Tony et Zineb : ajouter `for_clause`
- Timon : ajouter `array_assignement`
- Loïc : former un AST convenable sur des règles séparées (pour éviter les ambiguités)

Informations échangées :

- Possibilité de continuer la grammaire même si elle n'est pas $LL(1)$
- Les règles du parser peuvent prendre des paramètres. On peut transférer un token aux sous-règles afin de le placer au bon endroit dans l'AST :

```
statement
    :   IDF! idf_statement_end[$IDF]
    ;
idf_statement_end[Token id]
    :   proc_call[$id]
    ;
proc_call[Token id]
    :   '()' ';' -> ^(CALL {new CommonTree($id)})
    ;
```

Actions à suivre :

- Générer l'AST pour les éléments restants (expressions arithmétiques)
- Poursuivre l'amélioration de la grammaire (ajout des éléments restants)
- Création de tests nommés, prêts pour l'évaluation

Date de la prochaine réunion : Mercredi 20 novembre 2019

Compte-rendu de réunion 7

Mercredi 20 novembre 2019, 14h-18h

Type : Réunion d'avancement

Lieu : Salle PI

Présents : Loïc Bertrand *(Secrétaire)*, Timon Fugier, Tony Zhou, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

- Générer l'AST pour les éléments restants (expressions arithmétiques)
- Poursuivre l'amélioration de la grammaire (ajout des éléments restants)
- Création de tests nommés, prêts pour l'évaluation
- Mettre en forme les temps de travail pour qu'il soient facilement insérés dans le rapport de projet en LaTeX

Informations échangées :

- Explication du concept de paramètres de règles (cf. CR06) et son utilisation pour réordonner l'AST dans le cas où une factorisation a été effectuée
- L'échéance de l'évaluation a encore été repoussée, elle aura lieu les 3 et 4 décembre. Mettre à jour le Gantt.
- Ajout des labels et `goto`
- Création de l'AST pour les opérations arithmétiques

Actions à suivre :

- Ajouter `procedure_call` dans `expression` (résoudre les ambiguïtés liées à cela)
- Bug : (edit : pas un bug) end dans l'AST quand dernier statement sans point virgule
- Bug : `SIGNED_INTEGER` : `+5` est pris comme un token, exemple :

```
begin
  a := y**z*q+656
end
```

différent de :

```
begin
  a := y**z*q+ 656
end
```

Date de la prochaine réunion : Mardi 26 novembre 2019

Compte-rendu de réunion 8

Mercredi 26 novembre 2019, 8h-12h

Type : Réunion d'avancement

Lieu : Salle PI

Présents : Loïc Bertrand, Timon Fugier *(Secrétaire)*, Tony Zhou, Zineb Ziani El Idrissi

Absents : Aucun absent

Ordre du jour :

— Générer l'AST pour les éléments restants (expressions arithmétiques)
— Poursuivre l'amélioration de la grammaire (ajout des éléments restants)
— Création de tests nommés, prêts pour l'évaluation
— Résolution des bugs `SIGNED_INTEGER : +5`

Informations échangées :

— Ajout des `procedure_call` et `array_call` dans `expression`
— Création de l'AST pour les expressions
— Résolution d'un problème pour les additions et soustraction
— Avancement du compte-rendu
— Réalisation de plusieurs nouveaux test pour notre grammaire

Actions à suivre :

— Mettre à jour la grammaire dans le rapport, veiller à ce que les lignes ne dépassent pas du document
— Création de test nommés, prêts pour l'évaluation + tests et AST correspondants dans le rapport (Timon)
— Créer la bibliographie (Zineb)
— Liste des éléments de la grammaire et noms de ceux qui les ont implémentés (Tony)
— Structure de l'arbre abstrait (extraits d'arbre) (Loïc)