PCL : Rapport d'activité 1

Grammaire

Version finale

Guillaume BOURGEON, Louen GRÉAU, Arno KEMPF, Paul NICOLAS

TELECOM Nancy

18 novembre 2022

Table des matières

1 Rédaction de la grammaire **3**
1.1 Exemples d'application 3
1.1.1 Exemple fonctionnel 3
1.1.2 Exemple non-fonctionnel 4
1.2 Difficultés rencontrées 4
1.3 Déroulement du travail de groupe 4
1.4 Prochaines étapes 5

2 Annexe : Grammaire du langage **6**

1 Rédaction de la grammaire

1.1 Exemples d'application

1.1.1 Exemple fonctionnel

```
let
    function f(x:int) = (
        x := x*x
    )
in
end
```

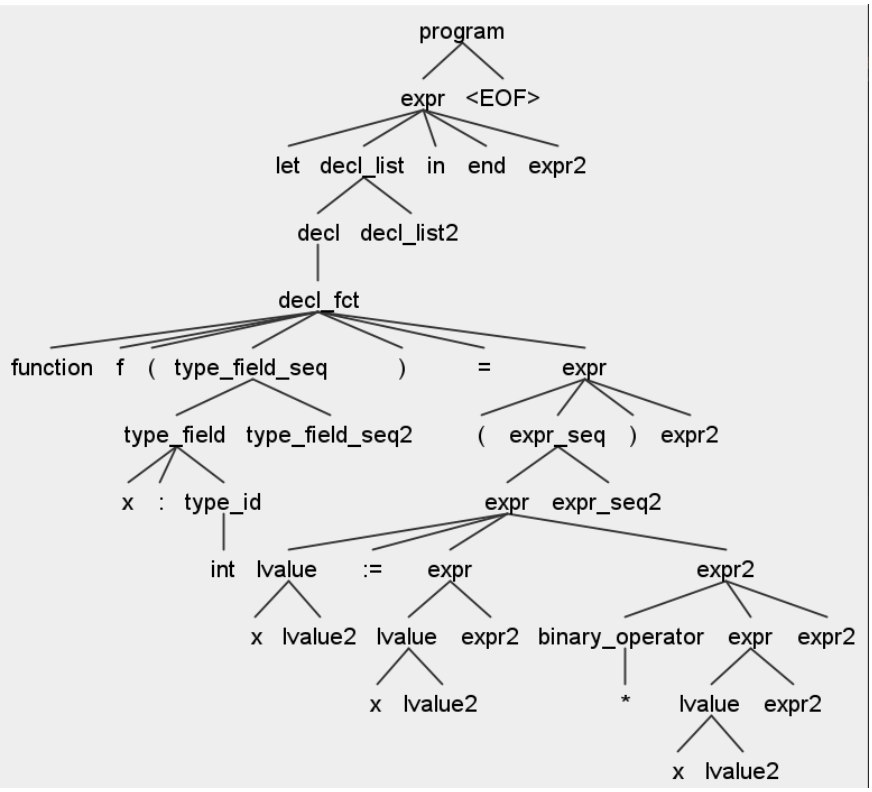

FIGURE 1.1 – Bon exemple : Déclaration de fonction

1.1.2 Exemple non-fonctionnel

Cet exemple ne respecte pas la syntaxe fournie dans le manuel qui veut qu'un "do" soit présent directement après l'instruction for.

```
for i:=0 to 20
    i := i+1
```

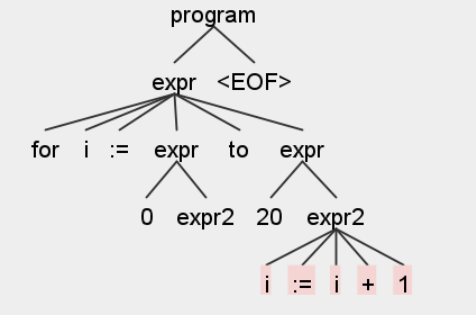

FIGURE 1.2 – Mauvais exemple : Boucle for

1.2 Difficultés rencontrées

Dans un premier temps, nous avons essayé de comprendre la syntaxe du langage Tiger. Cette étape a posé problème à certains membres du groupe. Les personnes ayant le mieux compris le langage ont pu s'atteller à l'écriture de la grammaire et à la rédaction d'exemples pour tester cette dernière.

Nous avons donc écrit la grammaire en nous appuyant sur celle fournit dans le manuel de référence du langage. Cette dernière étant récursive gauche, nous avons dû la dérécursiver pour être au point sur les attentes du sujet. Nous avons pour celà revu la méthode pour dérécursiver une grammaire.

1.3 Déroulement du travail de groupe

Pour ce projet, nous faisons environ une réunion par semaine pour nous répartir les tâches et égaliser la compréhension du projet sur tous les membres du groupe. Des groupes de discussions ont été créés pour répondre aux questions de chacun et pour faire des réunions en visioconférence. Nous avons fait quatre réunions de projet et une réunion pour la rédaction du rapport.

1.4 Prochaines étapes

Nous allons maintenant devoir nous assurer, par la rédaction de nouveaux tests, que la grammaire fonctionne correctement pour pouvoir commencer la construction de l'AST. Nous avons déjà, lors d'une précédente réunion, fait une mise au point sur le design pattern à utiliser pour l'AST. Cela nous permettra de nous répartir le travail et d'aller plus vite lors de l'implémentation des classes et fonctions nécessaires.

2 Annexe : Grammaire du langage

```
grammar expr;
@header{
    package parser;
}
program
    :   expr EOF
    ;
expr
    :
    // Constants
        STRING_CONSTANT expr2
    |   INTEGER_CONSTANT expr2
    |   'nil' expr2
    // Assignment
    |   lvalue expr2
    |   lvalue ':=' expr expr2
    // Function call
    |   IDENTIFIER '(' (expr_list)* ')' expr2
    // Operations
    |   '-' expr expr2
    |   '(' expr_seq ')' expr2
    // Array and Record creation
    |   type_id '{' field_list? '}' expr2
    |   type_id '[' expr ']' 'of' expr expr2
    // structures
    |   'if' expr 'then' expr ('else' expr)? expr2
    |   'while' expr 'do' expr expr2
    |   'for' IDENTIFIER ':=' expr 'to' expr 'do' expr expr2
    |   'break' expr2
    |   'let' decl_list 'in' expr_seq? 'end' expr2
    //
    ;
expr2 :
    | binary_operator expr expr2
    | /* empty */
    ;
lvalue
```

```
    :   IDENTIFIER lvalue2
    ;
lvalue2 :
    '.' IDENTIFIER lvalue2
    | '[' expr ']' lvalue2
    | /* empty */
    ;
binary_operator
    :   ('|' | '&' | '<=' | '>=' | '<' | '>' | '<>' | '='
    | '-' | '+' | '/' | '*')
    ;
expr_list
    :   expr expr_list2
    ;
expr_list2 :
    | ',' expr expr_list2
    | /* empty */
    ;
expr_seq
    :   expr expr_seq2
    ;
expr_seq2 :
    | ';' expr expr_seq2
    | /* empty */
    ;
field_list
    :   IDENTIFIER '=' expr field_list2
    |   field_list ',' IDENTIFIER '=' expr
    ;
field_list2 :
    | ',' IDENTIFIER '=' expr field_list2
    | /* empty */
    ;
// Declaration
decl_list
    :   decl decl_list2
    ;
decl_list2 :
    | decl decl_list2
    | /* empty */
    ;
decl
    :   decl_type
    |   decl_var
```

```
    |   decl_fct
    ;
// Type
decl_type
    :   type type_id '=' type
    ;
type
    :   type_id
    |   '{' type_field_seq '}'
    |   'array of' type_id
    ;
type_field_seq
    :   type_field  type_field_seq2
    ;
type_field_seq2 :
    | ',' type_field type_field_seq2
    | /* empty */
    ;
type_field
    :   IDENTIFIER ':' type_id
    ;
type_id
    :   'int'
    |   'string'
    |   IDENTIFIER
    ;
// Variable
decl_var
    :   'var' IDENTIFIER (':' type_id)? ':=' expr
    ;
// Function
decl_fct
    :   'function' IDENTIFIER '(' type_field_seq? ')' (':' type_id)? '=' expr
    ;
STRING_CONSTANT
    : '"' ( ~[\r\n"] | (('!'..'~')|' ') )* '"'
    ;
INTEGER_CONSTANT
    :   ('0'..'9')+
    ;
IDENTIFIER
    :   (('a'..'z')|('A'..'Z'))
        (('a'..'z')|'_'|('A'..'Z')|('0'..'9'))*
    ;
```

```
COMMENT
    :   '/*' .*? '*/' -> skip
    ;
LINE_COMMENT
    :   '//' ~[\r\n]* -> skip
    ;
WS
    :   ('\n'|' '|'\t'|'\r')+ -> skip
    ;
```