
# FORTRESS
# Introduction to language theory and compiling Project
# Part 2

## INFO-F-403

Van der Cam Matthias and Houart Benjamin
Matricule : 494559 and 501662

November 2022

# Contents

# 1 Modified Grammar

| | | |
|---|---|---|
| (1) | <Program> | → BEGIN [ProgName] <Code> END |
| (2) | <Code> | → <Instruction> , <Code> |
| (3) | | → $\varepsilon$ |
| (4) | <Instruction> | → <Assign> |
| (5) | | → <If> |
| (6) | | → <While> |
| (7) | | → <Print> |
| (8) | | → <Read> |
| (9) | <Assign> | → [VarName] := <ExprArith> |
| (10) | <Cond> | → <ExprArith> <Comp> <ExprArith> |
| (11) | <Comp> | → = |
| (12) | | → > |
| (13) | | → < |
| (14) | <ExprArith> | → <Prod> <ExprArith'> |
| (15) | <ExprArith'> | → + <Prod> <ExprArith'> |
| (16) | | → - <Prod> <ExprArith'> |
| (17) | | → $\varepsilon$ |
| (18) | <Prod> | → <Atom> <Prod'> |
| (19) | <Prod'> | → * <Atom> <Prod'> |
| (20) | | → / <Atom> <Prod'> |
| (21) | | → $\varepsilon$ |
| (22) | <Atom> | → - <Atom> |
| (23) | | → ( ExprArith ) |
| (24) | | → [VarName] |
| (25) | | → [Number] |
| (26) | <If> | → IF (<Cond>) THEN <Code> <IfSeq> |
| (27) | <IfSeq> | → END |
| (28) | | → ELSE <Code> END |
| (29) | <While> | → WHILE (<Cond>) DO <Code> END |
| (30) | <Print> | → PRINT([VarName]) |
| (31) | <Read> | → READ([VarName]) |

Figure 1: Grammar

## 1.1 Unproductive and unreachable variables

There wasn't any unproductive or unreachable variables.

## 1.2 Priority of the operators

We followed the method explained in the section 4.4.4 of the syllabus. It consist in the introduction of the notions of <Prod> and <Atom> and modifying the existing rules in order to respect the hierarchy of priorities. <Prod> account for the operators $*, /$, <Atom> is the basic building block of the this grammar (either another <ExpArith>, a Varname or a Number). Comparators haven't been modified since there is no ambiguity regarding their priority.

## 1.3 Left recursion and factorisation

We got rid of the ambiguity of the grammar by removing the left-recursion. Therefore we introduced <ExpArith'> and <Prod'>. We also used factorisation to avoid two rules having the same prefix. To do so

we introduced <IfSeq> and factored the `IF (<Cond>) THEN <Code>` prefix which was common to the old rules 19 & 20.

## 2 Action Table

The $first^1$ set is built following the algorithm we have seen during the course.

| Variable | First terminals | First variables |
|---|---|---|
| <Program> | BEGIN | |
| <Code> | $\varepsilon$ | <Instruction> |
| <Instruction> | | <Assign> <If> <While> <Print> <Read> |
| <Assign> | VarName | |
| <Cond> | | <ExprArith> |
| <Comp> | > < = | |
| <ExprArith> | | <Prod> |
| <ExprArith'> | + - $\varepsilon$ | |
| <Prod> | | <Atom> |
| <Prod'> | * / $\varepsilon$ | |
| <Atom> | - ( VarName Number | |
| <If> | IF | |
| <IfSeq> | END ELSE | |
| <While> | WHILE | |
| <Print> | PRINT | |
| <Read> | READ | |

Figure 2: First iteration

The following iterations are grouped together in the next table because they were very small.

| Variable | First terminals |
|---|---|
| <Program> | BEGIN |
| <Code> | $\varepsilon$ VarName IF WHILE PRINT READ |
| <Instruction> | VarName IF WHILE PRINT READ |
| <Assign> | VarName |
| <Cond> | - ( VarName Number |
| <Comp> | > < = |
| <ExprArith> | - ( VarName Number |
| <ExprArith'> | + - $\varepsilon$ |
| <Prod> | - ( VarName Number |
| <Prod'> | * / $\varepsilon$ |
| <Atom> | - ( VarName Number |
| <If> | IF |
| <IfSeq> | END ELSE |
| <While> | WHILE |
| <Print> | PRINT |
| <Read> | READ |

Figure 3: Following iterations

Because the grammar is LL(1), the follow set is only necessary when the first set of a variable contains the

$\varepsilon$ symbol. Only three variables are concerned by this : <Code>, <ExprArith'> and <Prod'>. Their follow set are :

- <Code> : END ELSE
- <ExprArith'> : = > < ) ,
- <Prod'> : + - = > < ) ,

| | BEGIN | NUMBER | VARNAME | IF | WHILE | PRINT | READ | END | ELSE | - | + | * | / | ( | ) | > | < | = | , |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Program | 1 | | | | | | | | | | | | | | | | | | |
| Code | | | 2 | 2 | 2 | 2 | 2 | 3 | 3 | | | | | | | | | | |
| Instruction | | | 4 | 5 | 6 | 7 | 8 | | | | | | | | | | | | |
| Assign | | | 9 | | | | | | | | | | | | | | | | |
| Cond | | 10 | 10 | | | | | | | 10 | | | | 10 | | | | | |
| Comp | | | | | | | | | | | | | | | | 12 | 13 | 11 | |
| ExprArith | | 14 | 14 | | | | | | | 14 | | | | 14 | | | | | |
| ExprArith' | | | | | | | | | | 16 | 15 | | | | 17 | 17 | 17 | 17 | 17 |
| Prod | | 18 | 18 | | | | | | | 18 | | | | 18 | | | | | |
| Prod' | | | | | | | | | | 21 | 21 | 19 | 20 | | 21 | 21 | 21 | 21 | 21 |
| Atom | | 25 | 24 | | | | | | | 22 | | | | 23 | | | | | |
| If | | | | 26 | | | | | | | | | | | | | | | |
| IfSeq | | | | | | | | 27 | 28 | | | | | | | | | | |
| While | | | | | 29 | | | | | | | | | | | | | | |
| Print | | | | | | 30 | | | | | | | | | | | | | |
| Read | | | | | | | 31 | | | | | | | | | | | | |

# 3 Remarks

The implementation of this part of the project is based on the implementation given as solution for the first part.

# 4 Example files

## 4.1 Factorial.fs

This is the provided example file.

## 4.2 Fibo.fs

This is an example of a program displaying the N first term of the famous Fibonacci sequence.

## 4.3 Operators.fs

This is an example allowing the test of the right priority of the operators with the Figure 4 on page 7 displaying the result.

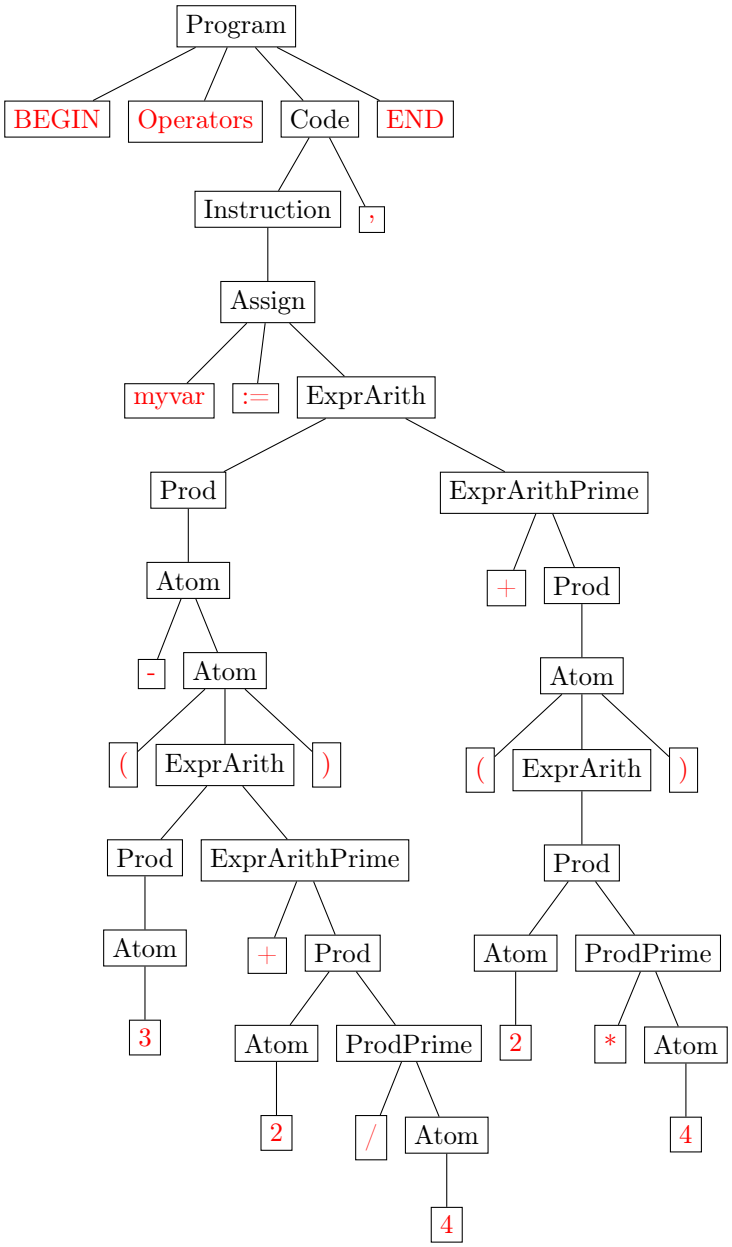


Figure 4: Tree of Operators.fs