

ULB  
INFO-F403 - Introduction to language theory and  
compiling  
Introduction to language theory and compiling

BASTOGNE Jérôme,  
HEREMAN Nicolas

Academic year 2015-2016 - November 26, 2015

# Part 2 - Grammar

## The modified grammar

There was no unproductive or unreachable variables to delete in this grammar. The left factorisation had to be applied on the rules of  $\langle \text{InstList} \rangle$  and  $\langle \text{If} \rangle$ , therefore the rules  $\langle \text{NextInst} \rangle$  and  $\langle \text{EndIf} \rangle$  were created. Then, the ambiguity in the  $\langle \text{Cond} \rangle$  and  $\langle \text{ExprArith} \rangle$  rules had to be removed with the informations given on the associativity and priority. For the conditions, "and" and "or" had to be separated. For the  $\langle \text{ExprArith} \rangle$ , the operators separating terms (+ and -) and the ones separating factors (\* and /) had to be differentiated. So these rules were obtained.

$\langle \text{Cond} \rangle$	$\rightarrow$	$\langle \text{Cond} \rangle$ or $\langle \text{AndCond} \rangle$
	$\rightarrow$	$\langle \text{AndCond} \rangle$
$\langle \text{AndCond} \rangle$	$\rightarrow$	$\langle \text{AndCond} \rangle$ and $\langle \text{CondTerm} \rangle$
	$\rightarrow$	$\langle \text{CondTerm} \rangle$
$\langle \text{CondTerm} \rangle$	$\rightarrow$	$\langle \text{SimpleCond} \rangle$
	$\rightarrow$	not $\langle \text{SimpleCond} \rangle$
$\langle \text{ExprArith} \rangle$	$\rightarrow$	$\langle \text{ExprArith} \rangle$ $\langle \text{TermOp} \rangle$ $\langle \text{Term} \rangle$
	$\rightarrow$	$\langle \text{Term} \rangle$
$\langle \text{Term} \rangle$	$\rightarrow$	$\langle \text{Term} \rangle$ $\langle \text{FactorOp} \rangle$ $\langle \text{Factor} \rangle$
	$\rightarrow$	$\langle \text{Factor} \rangle$
$\langle \text{Factor} \rangle$	$\rightarrow$	$(\langle \text{ExprArith} \rangle)$
	$\rightarrow$	$-\langle \text{ExprArith} \rangle$
	$\rightarrow$	$[\text{VarName}]$
	$\rightarrow$	$[\text{Number}]$
$\langle \text{TermOp} \rangle$	$\rightarrow$	+
	$\rightarrow$	-
$\langle \text{FactorOp} \rangle$	$\rightarrow$	*
	$\rightarrow$	/

As you can see  $\langle \text{Op} \rangle$  was transformed in  $\langle \text{TermOp} \rangle$ , and  $\langle \text{FactorOp} \rangle$  and  $\langle \text{BinOp} \rangle$  have been deleted. You can also see that there are still left-recursions that have to be removed. The real deal in this part was to handle the correct associativity while removing left-recursions. Keeping the order of priority was not a problem. The problem is that we lost the associativity

to the left when we tried to remove left-recursions. We tried multiple solutions to keep left associativity while removing left-recursions but none worked. So from here, we only could satisfy one of those constraints. We chose to remove left-recursion because otherwise the algorithm wouldn't work. This means that our compiler works but is kind of false because left associativity is not respected.

This is our new modified grammar we ended up with :

[ 1]	<Program>	→	begin <Code> end
[ 2]	<Code>	→	Epsilon
[ 3]		→	<InstList>
[ 4]	<InstList>	→	<Instruction> <NextInst>
[ 5]	<NextInst>	→	Epsilon
[ 6]		→	; <InstList>
[ 7]	<Instruction>	→	<Assign>
[ 8]		→	<If>
[ 9]		→	<While>
[10]		→	<For>
[11]		→	<Print>
[12]		→	<Read>
[13]	<Assign>	→	[ VarName] := <ExprArith>
[14]	<ExprArith>	→	<Term> <ExprArith2>
[15]	<ExprArith2>	→	<TermOp> <Term> <ExprArith2>
[16]		→	Epsilon
[17]	<Term>	→	<Factor> <Term2>
[18]	<Term2>	→	<FactorOp> <Factor> <Term2>
[19]		→	Epsilon
[20]	<Factor>	→	(<ExprArith>)
[21]		→	- <ExprArith>
[22]		→	[ VarName]
[23]		→	[ Number]
[24]	<TermOp>	→	+
[25]		→	-
[26]	<FactorOp>	→	*
[27]		→	/
[28]	<If>	→	if <Cond> then <Code> <EndIf>
[29]	<EndIf>	→	fi
[30]		→	else <Code> fi
[31]	<Cond>	→	<AndCond> <Cond2>
[32]	<Cond2>	→	or <AndCond> <Cond2>
[33]		→	Epsilon
[34]	<AndCond>	→	<CondTerm> <AndCond2>
[35]	<AndCond2>	→	and <CondTerm> <AndCond2>
[36]		→	Epsilon
[37]	<CondTerm>	→	<SimpleCond>
[38]		→	not <SimpleCond>
[39]	<SimpleCond>	→	<ExprArith> <Comp> <ExprArith>
[40]	<Comp>	→	=
[41]		→	>=
[42]		→	>



```

Code : {end, FIRST(EndIf), fi, od}
InstList : {FOLLOW(Code)}
NextInst : {FOLLOW(InstList)}
Instruction : {FIRST(NextInst)}
Assign : {FOLLOW(Instruction)}
ExprArith : {}, by, to, do, FOLLOW(Assign, Factor, SimpleCond), FIRST(Comp)}
ExprArith2 : {FOLLOW(ExprArith)}
Term : {FIRST(ExprArith2)}
Term2 : {FOLLOW(Term)}
Factor : {FIRST(Term2)}
TermOp : {FIRST(Term)}
FactorOp : {FIRST(Factor)}
If : {FOLLOW(Instruction)}
EndIf : {FOLLOW(If)}
Cond : {then, do}
Cond2 : {FOLLOW(Cond)}
AndCond : {FIRST(Cond2)}
AndCond2 : {FOLLOW(AndCond)}
CondTerm : {FIRST(AndCond2)}
SimpleCond : {FOLLOW(CondTerm)}
Comp : {FIRST(ExprArith)}
While : {FOLLOW(Instruction)}
For : {FOLLOW(Instruction)}
Print : {FOLLOW(Instruction)}
Read : {FOLLOW(Instruction)}

```

Now that we have our first and follow sets, we can build our action table (available in the last page of this report). This is how we fill the table :

For each token in First(X), we add the corresponding rule to the corresponding cell in the table. If epsilon is in First(X), for each token in Follow(X), we add the corresponding rule to the corresponding cell in the table.

Our grammar is LL(1), this means that LL(1) parsing uses only one symbol of input to predict the next grammar rule that should be used. Therefore each cell of our action table contains at most one rule. This table will help us to decide which decision should be made if a given nonterminal N is at the top of the parsing stack, based on the current input symbol.

[illegible]