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 variable to delete in this grammar. The left factorisation had to be applied on the rules of <InstList> and <If> so the rules <NextInst> and <EndIf> were created for that reason. Then the ambiguity in the <Cond> and <ExprArith> rules had to be remove with the informations given on the associativity and priority. For the condition "and" and "or" had to be separated. For the <ExprArith>, the operators separating terms (+ and -) and the ones separating factors (* and /) had to be differentiated. So these rules were obtained.

<Cond>	–>	<Cond> or <AndCond>
	–>	<AndCond>
<AndCond>	–>	<AndCond> and <CondTerm>
	–>	<CondTerm>
<CondTerm>	–>	<SimpleCond>
	–>	not <SimpleCond>

<ExprArith>	–>	<ExprArith> <TermOp> <Term>
	–>	<Term>
<Term>	–>	<Term> <FactorOp> <Factor>
	–>	<Factor>
<Factor>	–>	(<ExprArith>)
	–>	– <ExprArith>
	–>	[VarName]
	–>	[Number]
<TermOp>	–>	+
	–>	–
<FactorOp>	–>	*
	–>	/

As you can see <Op> were transformed in <TermOp> and <FactorOp> and <BinOp> has been deleted. You can also see that there is still left-recursions that has to be remove. The real deal in this part was to handle the correct associativity while removingit. 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]                 -> >
[43]                 -> <=
```

```
[44]                    -> <
[45]                    -> /=
[46]    <While>         -> while <Cond> do <Code> od
[47]    <For>           -> for [VarName] from <ExprArith> by <ExprArith> to
                               <ExprArith> do <Code> od
[48]    <Print>         -> print([VarName])
[49]    <Read>          -> read([VarName])
```

The action table

To build the action table, we first needed to calculate the first and follow sets. First(X) is made by taking the set of strings of terminals of maximum length which can start a string generated from X. Follow(X) is made by taking the set of strings of terminals of maximum length which can follow a string generated from X. This first and follow sets will help us to fill the action table.

FIRST :

```
_______
Program : {begin}
Code : {Epsilon,FIRST(InstList)}
InstList : {FIRST(Instruction)}
NextInst : {Epsilon,;}
Instruction :{FIRST(Assign,If,While,For,Print,Read)}
Assign : {VarName}
ExprArith : {FIRST(Term)}
ExprArith2 : {Epsilon, FIRST(TermOp)}
Term : {FIRST(Factor)}
Term2 : {Epsilon, FIRST(FactorOp)}
Factor : {(, -, VarName, Number}
TermOp : {+, -}
FactorOp : {*, /}
If : {if}
EndIf : {fi, else}
Cond : {FIRST(AndCond)}
Cond2 : {Epsilon, or}
AndCond : {FIRST(CondTerm)}
AndCond2 : {Epsilon, and}
CondTerm : {not, FIRST(SimpleCond)}
SimpleCond : {FIRST(ExprArith)}
Comp : {=, >=, >, <=, <, /=}
While : {while}
For : {for}
Print : {print}
Read : {read}
```

Follow:

```
_______
Program : {$}
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.

	VARNAM	NUMBER	BEG	END	;	ASSIGN	(	)	-	+	*	/	IF	THEN	FI	ELSE	NOT	AND	OR	=	>=	>	<=	<	/=	WHILE	DO	OD	FOR	FROM	BY	TO	PRINT	READ	$
Program	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Code	3	-	-	2	-	-	-	-	-	-	-	-	3	-	2	2	-	-	-	-	-	-	-	-	-	3	-	2	3	-	-	-	3	3	-
InstList	4	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	4	-	-	-	4	4	-
NextInst	-	-	-	5	6	-	-	-	-	-	-	-	-	-	5	5	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-
Instruction	7	-	-	-	-	-	-	-	-	-	-	-	8	-	-	-	-	-	-	-	-	-	-	-	-	9	-	-	10	-	-	-	11	12	-
Assign	13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ExprArith	14	14	-	-	-	-	14	-	14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ExprArith2	16	16	-	16	16	-	16	16	15	15	16	16	-	16	16	16	16	16	16	16	16	16	16	16	16	-	16	16	-	-	16	16	-	-	-
Term	17	17	-	-	-	-	17	-	17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Term2	-	-	-	19	19	-	-	19	19	19	18	18	-	19	19	19	-	19	19	19	19	19	19	19	19	-	19	19	-	-	19	19	-	-	-
Factor	22	23	-	-	-	-	20	-	21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TermOp	-	-	-	-	-	-	-	-	25	24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FactorOp	-	-	-	-	-	-	-	-	-	-	26	27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
If	-	-	-	-	-	-	-	-	-	-	-	-	28	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
EndIf	-	-	-	-	-	-	-	-	-	-	-	-	-	-	29	30	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Cond	31	31	-	-	-	-	31	-	31	-	-	-	-	-	-	-	31	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Cond2	-	-	-	-	-	-	-	-	-	-	-	-	-	33	-	-	-	-	32	-	-	-	-	-	-	-	33	-	-	-	-	-	-	-	-
AndCond	34	34	-	-	-	-	34	-	34	-	-	-	-	-	-	-	34	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
AndCond2	-	-	-	-	-	-	-	-	-	-	-	-	-	36	-	-	-	35	36	-	-	-	-	-	-	-	36	-	-	-	-	-	-	-	-
CondTerm	37	37	-	-	-	-	37	-	37	-	-	-	-	-	-	-	38	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SimpleCond	39	39	-	-	-	-	39	-	39	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Comp	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	40	41	42	43	44	45	-	-	-	-	-	-	-	-	-	-
While	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	46	-	-	-	-	-	-	-	-	-
For	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	47	-	-	-	-	-	-
Print	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	48	-	-
Read	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	49	-