INFO-F-403 Introduction to Language Theory and Compilation Project Part 2: S-COBOL Parsing, Semantic Analysis and Code Generation.

Delhaye Quentin

December 24, 2013

1 Introduction

The aim of this second part was to build a syntax analyzer (parser), semantic analyzer and code generator (to LLVM IR).

Firstly, the grammar had to be made LL(1) compliant, then the parser could be build.

2 Grammar

The table 2 present the rules of the grammar.

The following modifications were applied:

- removal of the left recursion in the various rules. Each time a left recursion occured, the rule has been split as presented in table 1.
- Fusion of the common left parts.
- Transformations to respect the priorities of the operators.
- Addition of the rules `4` and `6` so that `WORD` can be an `INTEGER`.
- Removal of the right recursion in the rules derived from the variable `<EXP_EQUAL_LR>`. Indeed, having a language accepting semantics like `a < b > c = d` makes no sense.

A	→	Aa
	becomes	
A	→	aA_LR
A_LR	→	aA_LR
	→	ε

Table 1: Example of the removal of the left recursion in rule A.

[1]	<PROGRAM>	→	<IDENT><ENV><DATA><PROC>
[2]	<IDENT>	→	identification division <END_INST> program-id. ID<END_INST> author. <WORDS><END_INST> date-written. <WORDS><END_INST>
[3]	<WORDS>	→	ID <WORDS_LR>
[4]		→	INTEGER <WORDS_LR

[5] <WORDS_LR> → ID <WORDS_LR>
[6] → INTEGER <WORDS_LR
[7] → ε
[8] <END_INST> → .\n
[9] <ENV> → environment division<END_INST> configuration section<END_INST> source-computer. <WORDS><END_INST> object-computer. <WORDS><END_INST>
[10] <DATA> → data division<END_INST> working-storage section<END_INST> <VAR_LIST>
[11] <VAR_LIST> → <VAR_DECL> <VAR_LIST>
[12] → ε
[13] <VAR_DECL> → <LEVEL> ID pic IMAGE <VAR_DECL_TAIL>
[14] <VAR_DECL_TAIL> → value INTEGER<END_INST>
[15] → <END_INST>
[16] <LEVEL> → INTEGER
[17] <PROC> → procedure division<END_INST> ID section<END_INST> <LABELS> end program ID.
[18] <LABELS> → <LABEL><END_INST> <INSTRUCTION_LIST> <LABELS_LR>
[19] <LABELS_LR> → <LABEL><END_INST> <INSTRUCTION_LIST> <LABELS_LR>
[20] → ε
[21] <LABEL> → ID
[22] <INSTRUCTION_LIST> → <INSTRUCTION> <INSTRUCTION_LIST>
[23] → ε
[24] <INSTRUCTION> → <ASSIGNATION>
[25] → <IF>
[26] → <CALL>
[27] → <READ>
[28] → <WRITE>
[29] → stop run<END_INST>
[30] <ASSIGNATION> → move <EXPRESSION> to ID<END_INST>
[31] → compute ID = <EXPRESSION><END_INST>
[32] → add <EXPRESSION> to ID<END_INST>
[33] → subtract <EXPRESSION> from ID<END_INST>
[34] → multiply <ASSIGN_END><END_INST>
[35] → divide <ASSIGN_END><END_INST>
[36] <ASSIGN_END> → <EXPRESSION>,<EXPRESSION> giving ID
[37] <EXPRESSION> → <EXP_AND> <EXPRESSION_LR>
[38] <EXPRESSION_LR> → or <EXP_AND> <EXPRESSION_LR>
[39] → ε
[40] <EXP_AND> → <EXP_EQUAL> <EXP_AND_LR>
[41] <EXP_AND_LR> → and <EXP_EQUAL> <EXP_AND_LR>
[42] → ε
[43] <EXP_EQUAL> → <EXP_ADD> <EXP_EQUAL_LR>
[44] <EXP_EQUAL_LR> → = <EXP_ADD>
[45] → < <EXP_ADD>

[46]		→	> <EXP_ADD>
[47]		→	<= <EXP_ADD>
[48]		→	>= <EXP_ADD>
[49]		→	ε
[50]	<EXP_ADD>	→	<EXP_MULT> <EXP_ADD_LR>
[51]	<EXP_ADD_LR>	→	+ <EXP_MULT> <EXP_ADD_LR>
[52]		→	- <EXP_MULT> <EXP_ADD_LR>
[53]		→	ε
[54]	<EXP_MULT>	→	<EXP_NOT> <EXP_MULT_LR>
[55]	<EXP_MULT_LR>	→	* <EXP_NOT> <EXP_MULT_LR>
[56]		→	/ <EXP_NOT> <EXP_MULT_LR>
[57]		→	ε
[58]	<EXP_NOT>	→	-<EXP_NOT>
[59]		→	not <EXP_NOT>
[60]		→	<EXP_PARENTHESIS>
[61]	<EXP_PARENTHESIS>	→	(<EXPRESSION>)
[62]		→	<EXP_TERM>
[63]	<EXP_TERM>	→	ID
[64]		→	INTEGER
[65]		→	true
[66]		→	false
[67]	<IF>	→	if <EXPRESSION> then <INSTRUCTION_LIST> <IF_END>
[68]	<IF_END>	→	else <INSTRUCTION_LIST> end-if
[69]		→	end-if
[70]	<CALL>	→	perform ID <CALL_TAIL>
[71]	<CALL_TAIL>	→	until <EXPRESSION><END_INST>
[72]		→	<END_INST>
[73]	<READ>	→	accept ID<END_INST>
[74]	<WRITE>	→	display <WRITE_TAIL>
[75]	<WRITE_TAIL>	→	<EXPRESSION><END_INST>
[76]		→	STRING<END_INST>

Table 2: LL(1) grammar of the S-COBOL language.

Variable	First[1]	Follow[1]
<PROGRAM>	identification	
<IDENT>	identification	environment
<WORDS>	ID, INTEGER	.
<WORDS_LR>	ID, INTEGER, ε	.
<END_INST>	.	program-id, date-written, environment, configuration, source-computer, object-computer, data, working-storage, INTEGER, ε, ID, move, compute, add, substract, multiply, divide, if, perform, accept, display, stop
<ENV>	environment	data

<DATA>	data	procedure
<VAR_LIST>	INTEGER, ε	procedure
<VAR_DECL>	INTEGER	INTEGER, ε
<VAR_DECL_TAIL>	value, .	INTEGER, ε
<LEVEL>	INTEGER	ID
<PROC>	procedure	
<LABELS>	ID	end
<LABELS_LR>	ID, ε	end
<LABEL>	ID	.
<INSTRUCTION_LIST>	move, compute, add, substract, multiply, divide, if, perform, accept, display, stop, ε	ID, ε
<INSTRUCTION>	move, compute, add, substract, multiply, divide, if, perform, accept, display, stop	move, compute, add, substract, multiply, divide, if, perform, accept, display, stop, ε
<ASSIGNATION>	move, compute, add, substract, multiply, divide	move, compute, add, substract, multiply, divide, if, perform, accept, display, stop, ε
<ASSIGN_END>	-, not, (, ID, INTEGER, true, false	.
<EXPRESSION>	-, not, (, ID, INTEGER, true, false	to, ., from, , , giving,), then
<EXPRESSION_LR>	or, ε	to, ., from, , , giving,), then
<EXP_AND>	-, not, (, ID, INTEGER, true, false	or, ε
<EXP_AND_LR>	and, ε	or, ε
<EXP_EQUAL>	-, not, (, ID, INTEGER, true, false	and, ε
<EXP_EQUAL_LR>	=, <, >, <=, >=, ε	and, ε
<EXP_ADD>	-, not, (, ID, INTEGER, true, false	=, <, >, <=, >=, ε
<EXP_ADD_LR>	+, -, ε	=, <, >, <=, >=, ε
<EXP_MULT>	-, not, (, ID, INTEGER, true, false	+, -, ε
<EXP_MULT_LR>	*, /, ε	+, -, ε
<EXP_NOT>	-, not, (, ID, INTEGER, true, false	*, /, ε
<EXP_PARENTHESIS>	(, ID, INTEGER, true, false	*, /, ε
<EXP_TERM>	ID, INTEGER, true, false	*, /, ε
<IF>	if	move, compute, add, substract, multiply, divide, if, perform, accept, display, stop, ε
<IF_END>	else, end-if	move, compute, add, substract, multiply, divide, if, perform, accept, display, stop, ε
<CALL>	perform	move, compute, add, substract, multiply, divide, if, perform, accept, display, stop, ε
<CALL_TAIL>	until, .	move, compute, add, substract, multiply, divide, if, perform, accept, display, stop, ε
<READ>	accept	move, compute, add, substract, multiply, divide, if, perform, accept, display, stop, ε

<WRITE>	display	move, compute, add, substract, multiply, divide, if, perform, accept, display, stop, ε
<WRITE_TAIL>	STRING, -, not, (, ID, INTEGER, true, false	move, compute, add, substract, multiply, divide, if, perform, accept, display, stop, ε

Table 3: First and Follow table.

	identification	ID	.	environment	data	INTEGER	value
<PROGRAM>	1						
<IDENT>	2						
<WORDS>		3					
<WORDS_LR>		4	7				
<END_INST>			8				
<ENV>				9			
<DATA>					10		
<VAR_LIST>					11		12
<VAR_DECL>					11		
<VAR_DECL_TAIL>			15				14
<LEVEL>						16	
<PROC>							
<LABELS>		18					
<LABELS_LR>		19					
<LABEL>		21					
<INSTRUCTION_LIST>		23					
<INSTRUCTION>							
<ASSIGNATION>							
<ASSIGN_END>		36				36	
<EXPRESSION>		37				37	
<EXPRESSION_LR>			39				
<EXP_AND>		40				40	
<EXP_AND_LR>							
<EXP_EQUAL>		43				43	
<EXP_EQUAL_LR>							
<EXP_ADD>		50				50	
<EXP_ADD_LR>							
<EXP_MULT>		54				54	
<EXP_MULT_LR>							
2<EXP_NOT>		60				60	
<EXP_PARENTHESIS>		61				61	
<EXP_TERM>		63				64	
<IF>							
<IF_END>							
<CALL>							
<CALL_TAIL>							
<READ>							
<WRITE>							
<WRITE_TAIL>		75				75	

Table 4: Action table (1/5).

	procedure	move	compute	add	substract	multiply	divide
<PROGRAM>							
<IDENT>							
<WORDS>							
<WORDS_LR>							
<END_INST>							
<ENV>							
<DATA>							
<VAR_LIST>	12						
<VAR_DECL>							
<VAR_DECL_TAIL>							
<LEVEL>							
<PROC>	15						
<LABELS>							
<LABELS_LR>							
<LABEL>							
<INSTRUCTION_LIST>		22	22	22	22	22	22
<INSTRUCTION>		24	24	24	24	24	24
<ASSIGNATION>		30	31	32	33	34	35
<ASSIGN_END>							
<EXPRESSION>							
<EXPRESSION_LR>							
<EXP_AND>							
<EXP_AND_LR>							
<EXP_EQUAL>							
<EXP_EQUAL_LR>							
<EXP_ADD>							
<EXP_ADD_LR>							
<EXP_MULT>							
<EXP_MULT_LR>							
<EXP_NOT>							
<EXP_PARENTHESIS>							
<EXP_TERM>							
<IF>							
<IF_END>							
<CALL>							
<CALL_TAIL>							
<READ>							
<WRITE>							
<WRITE_TAIL>							

Table 5: Action table (2/5).

	perform	accept	display	stop	-	not	(	true	false
<PROGRAM>									
<IDENT>									
<WORDS>									
<WORDS_LR>									
<END_INST>									
<ENV>									
<DATA>									
<VAR_LIST>									
<VAR_DECL>									
<VAR_DECL_TAIL>									
<LEVEL>									
<PROC>									
<LABELS>									
<LABELS_LR>									
<LABEL>									
<INSTRUCTION_LIST>	22	22	22	22					
<INSTRUCTION>	26	27	28	29					
<ASSIGNATION>									
<ASSIGN_END>					36	36	36	36	36
<EXPRESSION>					37	37	37	37	37
<EXPRESSION_LR>									
<EXP_AND>					40	40	40	40	40
<EXP_AND_LR>									
<EXP_EQUAL>					43	43	43	43	43
<EXP_EQUAL_LR>									
<EXP_ADD>					50	50	50	50	50
<EXP_ADD_LR>					52				
<EXP_MULT>					54	54	54	54	54
<EXP_MULT_LR>					57				
<EXP_NOT>					58	59	60	60	60
<EXP_PARENTHESIS>							61	62	62
<EXP_TERM>								65	66
<IF>									
<IF_END>									
<CALL>	70								
<CALL_TAIL>									
<READ>		72							
<WRITE>			73						
<WRITE_TAIL>					75	75	75	75	75

Table 6: Action table (3/5).

	or	and	=	<	>	<=	>=	*	/	if	else	end-if	+	until
<PROGRAM>														
<IDENT>														
<WORDS>														
<WORDS_LR>														
<END_INST>														
<ENV>														
<DATA>														
<VAR_LIST>														
<VAR_DECL>														
<VAR_DECL_TAIL>														
<LEVEL>														
<PROC>														
<LABELS>														
<LABELS_LR>														
<LABEL>														
<INSTRUCTION_LIST>										22				
<INSTRUCTION>										25				
<ASSIGNATION>														
<ASSIGN_END>														
<EXPRESSION>														
<EXPRESSION_LR>	38													
<EXP_AND>														
<EXP_AND_LR>	42	41												
<EXP_EQUAL>														
<EXP_EQUAL_LR>		49	44	45	46	47	48							
<EXP_ADD>														
<EXP_ADD_LR>			53	53	53	53	53						51	
<EXP_MULT>														
<EXP_MULT_LR>								55	56				57	
<EXP_NOT>														
<EXP_PARENTHESIS>														
<EXP_TERM>														
<IF>										67				
<IF_END>											68	69		
<CALL>														
<CALL_TAIL>														71
<READ>														
<WRITE>														
<WRITE_TAIL>														

Table 7: Action table (4/5).

	until	STRING	,	to	from	giving	)	then
<PROGRAM>								
<IDENT>								
<WORDS>								
<WORDS_LR>								
<END_INST>								
<ENV>								
<DATA>								
<VAR_LIST>								
<VAR_DECL>								
<VAR_DECL_TAIL>								
<LEVEL>								
<PROC>								
<LABELS>								
<LABELS_LR>								
<LABEL>								
<INSTRUCTION_LIST>								
<INSTRUCTION>								
<ASSIGNATION>								
<ASSIGN_END>								
<EXPRESSION>								
<EXPRESSION_LR>			39	39	39	39	39	39
<EXP_AND>								
<EXP_AND_LR>								
<EXP_EQUAL>								
<EXP_EQUAL_LR>								
<EXP_ADD>								
<EXP_ADD_LR>								
<EXP_MULT>								
<EXP_MULT_LR>								
<EXP_NOT>								
<EXP_PARENTHESIS>								
<EXP_TERM>								
<IF>								
<IF_END>								
<CALL>								
<CALL_TAIL>	71							
<READ>								
<WRITE>								
<WRITE_TAIL>		76						

Table 8: Action table (5/5).

3 Implementation

3.1 `Parser` Class

This class mainly handles the parsing and the semantic analysis.

Each rule of the grammar has been implemented in recursive methods. The parser can be accessed from the outside via the `parse(String)` method, which takes as argument the String to parse (typically the source code).

3.2 `RaiseError` Class

Whenever a problematic situation is encountered, one of those object is created. Its main purpose is to write on the error output the error message.

3.3 `WarningError` Class

Works just like `RaiseError`, but throw a warning instead of an error.

3.4 `LLVMGenerator` Class

This class has a method for every functionnality of the S-COBOL language that may be translated into LLVM IR.

It writes the code in a String, and finally write it into a file upon external sollicitation of `toFile(String)` (the argument being the filename.