CONTEXT-FREE GRAMMAR DEFINED FOR LADDER LOGIC

Marina PETICÎ[1*],
Ecaterina COTELNIC[1],
Dina CIORBA[1],
Dacian RUSU[1]

[1]Technical University of Moldova, Faculty of Computers, Informatics and Microelectronics, Department of Software Engineering and Automatics, FAF – 181/182, Chişinău, Republic of Moldova

*Reprinth Author: Peticî Marina, marina.petici@isa.utm.md

Abstract: *This article describes the grammar and the lexical parser of a domain-specific language (DSL) made for a Programmable Logic Controller (PLC). Furthermore, this paper expounds how the DSL, which is being developed, will work, what functions will be implemented and how this language will, with the help of Ladder Logic (LL), ease the interaction between humans and PLCs.*

Keywords: *domain-specific language, programmable logic controller, context-free grammar, ladder logic, graphic language, ladder diagram*

Introduction

The genesis of all reliable automated processes lies at the core of programmable logic controllers (PLSs) and they continue to evolve once new technologies are added to their capabilities. Starting as a replacement for electromechanical relay system, a PLC is an industrial grade computer that uses programmable memory, to implement and store multiple input and output arrangements, arithmetic logic, counting and timing, in order to control and process the given information. “The programmable logic controller is, then, basically a digital computer designed for use in machine control” [1].

A widely used programming language for PLCs is ladder logic (LL), the reason behind its popularity being a close replica of the relay system. Its structure reassemble a ladder and has two vertical bars (system power) filled with a series of vertical “rungs” between them, each one representing a control circuit. It also uses contacts and coils that act as inputs and outputs, the last ones not being physical but represented as a single bit in the PLC’s memory. In addition, contacts can be arranged in series to represent AND logic and in parallel for OR one.

Due to the fact that PLCs were designed to be operated by engineers that may not be familiar with computer programming languages, the simplest way to coalesce those fields is by creating a domain-specific language which will ease the “communication” between them. By having at the base the LL method of representing a graphical diagram based on the circuit diagrams of relay logic hardware, the process of creating a graphical language for programming PLCs is becoming substantially easier, as for the inputs we have only 0 and 1 and the variables alternate from contacts to coils [2].

Reference grammar

The language itself can be seen as a set of connections between logical checkers (contacts) and actuators (coils). If a path can be traced between the left side of the rung and the output, through asserted (true or "closed") contacts, the rung is true and the output coil storage bit is asserted (1) or true. If no path can be traced, then the output is false (0) and the "coil" by analogy to electromechanical relays is considered "de-energized". Ladder logic has contacts that make or break circuits to control coils. Each coil or contact corresponds to the status of a single bit in the programmable controller's memory. Unlike electromechanical relays, a ladder program can refer any number of times to the status of a single bit, equivalent to a relay with an indefinitely large number of contacts. So-called "contacts" may refer to physical ("hard") inputs to the programmable controller from physical devices such as limit switches via an integrated or external input module.

A program consists of a single chain, enclosed in two symbols "|" in the beginning and in the end. In a program must firstly appear an input (contact) and ends with an output (coil). Each problem should have at least one contact, one coil and one operation.

There are 4 types of variables, text form: openContact, closedContact, activeCoil and inactiveCoil. Graphic form is presented in the Table 1.

Table 1

Elements of the DSL

rung input: checkers (contacts)	open contact	⊣ ⊢	input
	closed contact	⊣/⊢	input with a logical NOT
rung output: actuators (coils)	active coil	−( )−	output
	inactive coil	−(/)−	output with a logical NOT

The variables of types openContact or closedContact are better to be called with an ‘I’ or ‘M’ uppercase and a number. The variables of types inactiveCoil or activeCoil are recommended to be called with an "M" or "Q"00 and a number. All variables are actually of the type boolean in a form of bits (0 or 1).

All closed and open contacts, as inputs, must have some values. Active and inactive coils, as outputs, and memory should not have values.

Setting to a variable value is made through a point, but not ‘=’. For instance, in another language program *“int a = 1”*. In DSL language for microcontrollers program *“closedContact I0.1”*. This means, that first input - closed contact *I*0 - has a value 1. Operators are represented in Table 2.

Table 2

Operators

Graphical representation	Text representation	Meaning
—	&	AND (serial connection)
⌊ ⌋	V	OR (parallel connection)

Each program represents a logical expression and returns and prints the values of coils. working of serial and parallel connections (Table 3).

Table 3

Logical expressions

first bit	second bit	& (serial connection)	V (parallel connection)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Graphical representation is pictured in Figure 1:

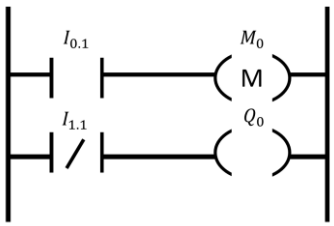

Figure 1. Graphical representation of our context-free developed grammar

Presentation of the Context Free Grammar

V_T = {start, (,), V, &, ., open contact, closed contact, active coil, inactive coil, I, M, Q, 0..9, end}.

V_N = {<source code>, <program>, <contact>, <coil>, <digit>, <value>, <operators>}.

Rules:

P = { <source code> → **start** <program>
<program> → **(** <program> | <contact> | <coil>
<contact> → **open contact I** <digit> **.** <value> <operators> | **open contact M** <digit> <operators> | **closed contact I** <digit> **.** <value> <operators> | **closed contact M** <digit>
<coil> → ***active coil Q*** *<digit> <operators>* | ***active coil M*** *<digit> <operators>* | ***inactive coil Q*** *<digit> <operators>* | ***inactive coil M*** *<digit> <operators>*
<operators> → ***&*** *<operators> <contact> <operators>*$^+$ | ***V*** *<operators>*
<contact><operators>$^+$ | ***&*** *<operators> <coil> <operators>*$^+$ | ***V*** *<operators> <coil>*
<operators>$^+$ | ***)*** *<operators>*$^+$ | ***(*** *<operators>*$^+$ | ***end*** | ε
<value> → ***0*** | ***1***
<digit> → ***0*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** | ***7*** | ***8*** | ***9*** | ***1*** *<digit>* | ***2*** *<digit>* | ***3*** *<digit>* | ***4*** *<digit>* | ***5*** *<digit>* | ***6*** *<digit>* | ***7*** *<digit>* | ***8*** *<digit>* | ***9*** *<digit>*
}

Program with one connection : start closed contact I0.1 & active coil Q0 end. The implementation is presented in the Figure 2 and Figure 3.

<source code> → start <program> → start <contact> → start closed contact *I* <digit> . <value> <operators> → start closed contact *I*0.1 <operators> → start closed contact *I*0.1 & <operators> <coil> <operators>$^+$ → start closed contact *I*0.1 & <coil> <operators>$^+$ → start closed contact *I*0.1 & active coil Q<digit> <operators> <operators>$^+$ → start closed contact *I*0.1 & active coil Q0 <operators> <operators>$^+$ → start closed contact *I*0.1 & active coil Q0 <operators>$^+$ → start closed contact *I*0.1 & active coil Q0 end

Figure 2. Analysis of the string derivation

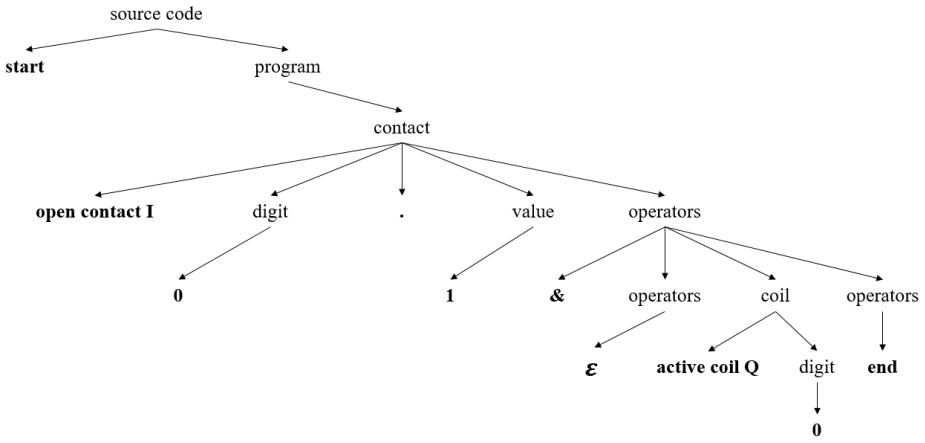

Figure 3. Parse tree for a program with one connection

Conclusions

This paper is meant to present a context-free grammar defined for ladder diagram as a DSL, whose schematics would help in programming a PLC to perform the same control functions. The programming language of a PLC was designed to resemble ladder logic diagrams because it makes them easier to program. In this way, it represents an important step in using PLC system, which makes machinery and systems work automatically and, thus, becoming very important and needed in all kinds of industry. For this domain specific language were defined lexical considerations, semantic rules and grammar, due to which can be built a working program by means of which is established the actual logic of the control system inside the PLC. The program is entered and viewed via a personal computer connected to the PLC programming port.

References

1. PETRUZELLA, FRANK D. *Programmable Logic Controllers*. Programmable Logic Controllers - 4th edition. New York: McGraw-Hill, 1989.
2. BOLTON, W. *Programmable Logic Controllers – 5th edition.* Oxford: Elsevier Ltd., 2009.
3. Sharif University of Techonology - Electrical Engineering Department, *Ladder Logic* [online], [accessed 03.03.2020]. Available: http://ee.sharif.edu/~industrialcontrol/LADDER_LOGIC_Tutorial.pdf
4. Ykanchanam, *PLC Introduction* [online], 2019, [accessed 05.03.2020]. Available: https://medium.com/@ykanchanam/plc-introduction-bb037a447d58
5. Atul Wadhai, *Explain Ladder Logic and its advantages* [online], 2007, [accessed 05.03.2020]. Available: https://medium.com/@atulwadhai/explain-ladder-logic-and-its-advantages-83b22cbf3e96