Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Formalization and Visualization of Non-binary PLC Programs

M. Bani Younis and G. Frey

Abstract— **Programmable Logic Controllers (PLCs) have been of great eminence in manufacturing systems and will probably remain predominant for some time to come. To allow re-implementation on a new hardware and visualization of existing code, a formalization approach for PLC programs is proposed. The method presented here is not restricted to binary operations in the PLC code but also considers digital operations. In order to achieve compact visualization and efficient re-implementation an abstraction of the low level Instruction List (IL) programs is developed. The formalization of the abstracted code results in a compact finite state machine representation. The method is implemented using JAVA and XML technologies. The IL is converted to XML, the XML document object model (DOM) is used for parsing and scalable vector graphics (SVG) is employed to graphically represent the resulting automata. The presented approach is illustrated using STEP 5 IL from Siemens. The method is however generic, other IL dialects could be parsed if the corresponding description files are built.**

I. Introduction

COMPUTER driven hardware has become vital in almost all areas of private and business live. Most of the electrical (and mechanical) appliances we use every day are controlled by micro controllers or computers in general (often without us noticing it). These computers stabilize cars during driving, ignite air bags, control elevators and fridges. However, one of the main tasks of computer control was and still is the control of manufacturing processes. In manufacturing a special kind of computer is used, the Programmable Logic Controller (PLC). PLCs have been introduced in the 1970s and soon became the major workhorse of industrial automation [1].

Because of the widespread use of PLCs in manufacturing, industry experts and researchers on manufacturing technology recognize the importance of simulation, verification, analysis, visualization, and re-implementation of PLC programs [2] [3].

Re-engineering of PLC programs is necessary when the PLC hardware should be changed or the intellectual property hidden in the code should be recovered to implement it in a different setting.

To achieve these pursuits a formal model of the PLC program under consideration is required. Formalization of PLC programs is an important area of active research. However, formalization alone and the methods based on a formal description do not solve all problems connected with PLC programs. A formal model allows formal verification and also re-implementation on a new platform but it often does not help the user in understanding the programs.

Generally, there is a lack of knowledge about the implemented code due to missing documentation or on-the-fly changes of the code after the first implementation. This happens for example when the mechanical setup of a manufacturing system is changed and the code has to be adapted to some new configuration. The longevity of PLC programs (often more than 10 years) accrues this problem. Furthermore, PLCs are programmed using low-level machine languages like Ladder Diagram (LD) or Instruction List (IL) [4]. Larger and complex programs in these languages are naturally hard to read and understand. Therefore, visualization of existing PLC programs is also an important step in re-engineering.

XML [5] and related technologies provide a good medium to visualize PLC programs [6]. The XML generated from IL programs as presented in [6] not only can be transformed into other formats using XSL, but also can be converted under JAVA to other forms like e.g. XML Metadata Interchange (XMI) [7], the XML format of UML [8]. A further technology related to XML is the basis for parts of the work depicted in this paper: The Scalable Vector Graphics (SVG) [9] can be used to graphically visualize the formal description generated from the PLC program.

The rest of the paper is structured as follows. A short introduction to PLCs and the proprietary STEP 5 language from Siemens is given in the following section. Section III explains the formalization of digital programs in general. In IV the abstraction method is explained in some detail. Section V gives some information on implementation issues. An example of the application of the method to a given PLC code is presented in Section VI. Section VII concludes the paper and gives an outlook on further work.

II. PLCs and STEP 5

The hardware of a PLC consists of a microprocessor based CPU, a memory, and input and output ports where external signals can be handled, e.g. received from sensors and sent to actuators.

PLCs operate in a polling mode with a precise execution cycle. This cycle basically consists of three steps (cf. Fig 1) which are continuously executed. In the "Read Inputs" step, the PLC kernel reads all the input values and copies them into its internal input memory PAE (cf. Fig 2). In the

This work was supported by "Stiftung Rheinland-Pfalz für Innovation", under project number 616.

G. Frey is head of the Juniorprofessorship Agentbased Automation (JPA²), University of Kaiserslautern, 67653 Kaiserslautern, Germany. (phone: +49-631-205-4455; fax: +49-631-205-4462; e-mail: frey@eit.uni-kl.de).

M. Bani Younis is with JPA², (e-mail: baniy@eit.uni-kl.de)

0-7803-9568-9/05/$20.00 ©2005 IEEE

"Execute User Program" step, the PLC kernel executes the user program which has access to all PLC memory areas. The algorithm stores the execution results in the output internal memory (PAA). Furthermore it can read from and write to the internal memory of the PLC. In the "Write Outputs" step, the PLC kernel copies the internal output memory to the output modules. Note that the program can also read form PAA.

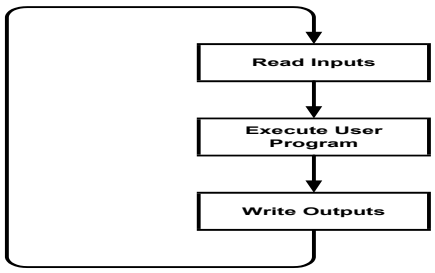

Fig 1. PLC cycle

Though PLCs have become prominent in the industry, they are devoid of the possibility to enhance compatibility, openness and interoperability, because of different vendor specific platforms. A working group within the International Electro-technical Commission (IEC) [10] was set up to look at the complete design of PLCs, including hardware design, installation, testing, documentation, programming and communications. The result of this process is the IEC standard 61131. Part 3 of this standard defines a set of programming languages including IL.

The approach presented here is not based on the standard form of IL proposed by the IEC but on the vendor-specific STEP 5 IL (used on the last generation of Siemens PLCs) [11], [12], [13], this language was a quasi-standard in Germany for several years but the corresponding hardware is now no longer in production. This means that there is a need for the application of re-engineering if the implemented algorithms should be transferred to new hardware. The approach itself however is easily adaptable to other IL dialects including the standard form.

The software on a STEP 5 PLC is implemented in a quasi-hierarchical form using four main types of modules as follows:

OB: Organization Modules serve for the management of the user program in form of a listing of the program Modules to be worked on.

PB: Program Modules contain the user program structured in groups. PBs contain only binary operations.

DB: Data Modules contain data, on which the user program works.

FB: Function Modules are employed to realize frequently used or very complicated functions. The PLC operations in FBs are in a hybrid form i.e. digital and binary operations.

In addition to these general types of modules STEP 5 also holds special types of modules like timers and counters.

The formalization of digital operations implemented in the FBs will be shown in this paper. The formalization of binary programs was presented in [8] and an extension on timers and counters can be found in [14].

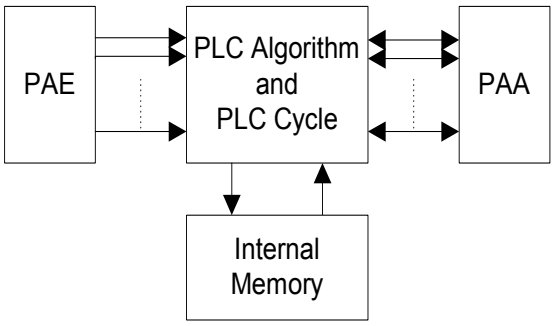

Fig 2. PLC memory access

III. Formalization of Digital Programs

A. Classification of Digital Operations

Binary operations allow simple logical decisions. Digital Operations in the Function Modules extend basic binary operations to allow other types of control, for instance, Data Handling, Numerical Logic, and Lists. A list of different types of digital operations is shown in Table I below.

TABLE I
TYPES OF BINARY OPERATIONS

Type	Operations
1	Load operation
2	Transfer operation
3	Arithmetic operation
4	Compare operation
5	Digital logical operation
6	1s complement operation
7	2s complement operation
8	Shift and rotate operation
9	Jump operation
10	Other operation

B. Transformation of Digital Programs

The most important step in the investigations to formalize PLC programs is the choice of a state definition where the influence of the PLC operations in a program on the different possible state variables has to be investigated. Besides the influence of these operations on the PAE, PAA and the internal memory, the operations can also affect the program counter of the execution algorithm (PC) as well as the status word. The status word is 8-bit wide. Every bit stands for a result display. This again can be a bit or a word display. The displays are evaluated or changed by the operations. Table II shows the content of the status word. In case binary operations are under consideration only the VKE (German for Current Result) is of importance, but in case digital operations or timers and counters are to be formalized it is important to check other elements of the Status Word besides the other parts of the PLC namely, AKKU1 and AKKU2 (accumulators in STEP 5).

TABLE II
CONFIGURATION OF THE STATUS WORD (RESULT DISPLAYS)

Type (Word/Bit)	Bit	Abbr.	Annotation
w	0/1	ANZ1/ANZ0	Result display 1/0
w	2	OV	Overflow
w	3	OS	Storing Overflow
b	4	OR	Or; Internal CPU Display
b	5	STA	STATUS
b	6	VKE	CR Current Result
b	7	$\overline{\text{ERAB}}$	First Inquiry

In the following two possibilities are indicated to transform a control program into a formal description. The first description is based on the transformation according to [15]. The second description refers to the conversion of the program into algorithms:

Using the method presented in [15] for binary operations, a state extension is necessary for digital programs. The extended state definition not only contains the variables, the Current Result (VKE corresponds to the accumulator after [15]) and the program counter, but also the accumulators AKKU1, AKKU2, the result displays ANZ1, ANZ0, and the Overflow-Display OV.

The accumulators (AKKU1, AKKU2) are necessary for example to compare two values against each other. The displays are affected by arithmetic operations depending on the value of the AKKU1 and are questioned by jump operations, i.e. the jump operations are executed depending on the signal states. Every operation is evaluated individually and is described by a transition in a transition system. The resulting transition system shows a good structure with few branchings. However, the state definition contains a lot of elements and the number of states (especially with jump operations) is quite high.

Because of this high number of states a different conversion for digital programs is presented here. The resulting model is a Mealy machine expressing each operation individually. For every non-binary operation in the code at least one state in the Mealy machine is necessary. The inputs and outputs of the PLC program, internal variables, VKE, AKKU1, AKKU2, ANZ1, ANZ0 and OV are represented as external variables of the automaton. Note again, the accumulators and displays are not contents of the state, but are treated like variables and are stored in the external state memory.

IV. ABSTRACTION OF DIGITAL PROGRAMS

The abstraction of the IL program to higher level algorithms is achieved through line-by-line conversion of the existing code into IF-THEN-ELSE statements. The process results in a series of IF-THEN-ELSE statements in the form: IF <expr> THEN <statm1> ELSE <statm2>. These can be converted to a state automaton. During this conversion the Program Counter is removed from the model. Starting from the initial state each IF-THEN-ELSE statement leads to a new state. expr corresponds to the input and statm1 to the matching output of the transition. If there is an ELSE clause then a second transition between the states is generated with NOT expr as an input and statm2 as an output. If the instruction contains no ELSE clause, no output takes place. Branching results only from conditional jumps in the PLC code. After this abstraction an additional step of optimization explained in the following is necessary. In the following sub-sections this process is explained for different operation types.

A. Load and Transfer Operations

A load operation (L) consists of two assignments, which change the contents of AKKU 1 and AKKU 2:

```
L MW 150 →  AKKU 2 = AKKU 1
            AKKU 1 = MW 150
```

A transfer operation (T) initiates the assignment of the contents of the AKKU 1 into an operand.

```
T MW 150 →  MW 150=AKKU1
```

B. Arithmetic Operation

Arithmetic operations combine the values of AKKU1 and AKKU2, the result is stored in AKKU1. In addition, arithmetic operations influence the result displays ANZ1, ANZ0 and the Overflow-Display OV (cf. Table III).

TABLE III
INFLUENCE ON RESULT DISPLAYS FOR ARITHMETIC OPERATIONS

Contents of AKKU 1 after execution	ANZ1	ANZ0	OV
Below the allowed lower limit (< -32768)	1	0	1
In the allowed negative domain (-1 to -32768)	0	1	0
Zero	0	0	0
In the allowed positive domain (+1 to +32768)	1	0	0
Above the allowed upper limit (> +32768)	0	1	1

This influence is realized in IF-THEN-ELSE statements by which all arithmetic operations are treated similar. The conversion of the arithmetic operation –F (integer subtraction of AKKU2 from AKKU1) results in an IF-THEN-ELSE-description as shown below:

```
-F →  AKKU1=AKKU2 - AKKU1
      IF   AKKU1 <= -1
      THEN IF   AKKU1 < -32768
           THEN   ANZ1=1 AND ANZ0=0 AND OV=1
           ELSE   ANZ1=0 AND ANZ0=1 AND OV=0
      ELSE IF   AKKU 1 >= 1
           THEN IFAKKU1 > 32768
                THEN   ANZ1=0 AND ANZ0=1 AND OV=1
                ELSE   ANZ1=1 AND ANZ0=0 AND OV=0
           ELSE ANZ1=0 AND ANZ0=0 AND OV=0
```

C. Compare Operation

By a compare operation the contents of AKKU1 and AKKU2 are compared. The result of the comparison is binary and is stored in the VKE. In addition, compare operations have influence on the result displays ANZ1, ANZ0 and OV (cf. Table IV).

TABLE IV
INFLUENCE ON RESULT DISPLAYS FOR COMPARE OPERATIONS

Operation	ANZ1	ANZ0	OV
Equal	0	0	0
Smaller	0	1	0
Greater	1	0	0

The conversion of the compare operation >F (check whether the integer in AKKU2 is greater than the one in AKKU1) results in an IF-THEN-ELSE-description as shown below:

```
>F →  IF   AKKU2>AKKU1
      THEN   VKE=1
      ELSE   VKE=0
      IF   AKKU2=AKKU1
      THEN   ANZ1=0 AND ANZ0=0 AND OV=0
      ELSE IF   AKKU2<AKKU1
           THEN   ANZ1=0 AND ANZ0=1 AND OV=0
           ELSE   ANZ1=1 AND ANZ0=0 AND OV=0
```

D. Digital Logical Operation

With the digital logical operations the values from AKKU1 and AKKU2 are logically combined. The result is saved in AKKU1. This corresponds to an allocation. Again an influence on the displays exists. ANZ0 and OV are set independently of the result to zero. The influence on ANZ1 is described by an IF-THEN-ELSE statement. Below the transformation for OW (Or for a word format) is shown.

```
OW→   AKKU1=AKKU1 OW AKKU2
        ANZ0=0
        OV=0
        IF   AKKU1=0
             THEN   ANZ1=0
             ELSE   ANZ1=1
```

E. Conversion Operation

These operations are used to convert the contents of the AKKU 1, for example into 2's complement using the operation KZW. This assignment influences the values of ANZ1, ANZ0, and OV.

TABLE V
INFLUENCE ON RESULT DISPLAYS FOR 2'S COMPLEMENT (KWZ)

Value of AKKU 1 after the 2' Complement	ANZ1	ANZ0	OV
(-)65536 (Result of KZW for KH=0000)	0	0	1
Below the allowed lower limit < -32768	1	0	1
In the allowed negative domain-1 to -32768	0	1	0
In the allowed positive domain +1 to +32768	1	0	0
Above the allowed upper limit > +32768	0	1	1

The transformation of the conversion operation KZW (2's complement) results in an IF-THEN-ELSE description as shown below:

```
KZW →   AKKU1= inverse AKKU1+1
        IF AKKU1 = ± 65536
        THEN ANZ1=0 AND ANZ0=0 AND OV=1
        ELSE IF AKKU1 < -1
             THEN IF     AKKU1 < - 32768
                  THEN   ANZ1=1 AND ANZ0=0 AND OV=1
                  ELSE   ANZ1=0 AND ANZ0=1 AND OV=0
             ELSE IF     AKKU1 > 32768
                  THEN   ANZ1=0 AND ANZ0=1 AND OV=1
               ELSE   ANZ1=1 AND ANZ0=0 AND OV=0
```

F. Bit- Shift and Rotate Operation

Applied on the AKKU 1 to shift or rotate it bitwise left or right. The rotate bit is evaluated according to ANZ1. ANZ0 and OV are always set to zero by these operations. Therefore, the conversion of the operation SLW 1 (shift to left by one bit) results in an IF-THEN-ELSE-description as shown below:

```
SLW 1 →   AKKU1= AKKU1 shifted to left by one Bit
          ANZ0=0
          OV=0
          IF   Bit 15 of AKKU1 =1
          THEN ANZ1=1
          ELSE ANZ1=0
```

G. Jump Operation

The declaration of the jump target is carried out symbolically through a jump label. Conditional jump operations must be treated individually because of the form of the condition (cf. Table VI). The operation SPO (jump in case of Overflow) is carried out if OV = 1.

TABLE VI
EXECUTED JUMP OPERATION ACCORDING TO RESULT DISPALYS

ANZ1	ANZ0	Executed jump operation	
0	0	SPZ	jump when zero
0	1	SPM	jump when negative
		SPN	jump when not zero
1	0	SPN	jump when not zero
		SPP	jump when positive

Examples for the conversion of jump operations are shown below:

```
SPM= M001→IF ANZ1=0 AND ANZ0=1  THEN Jump toM001.
SPN= M001→IF ANZ1 ≠ ANZ0        THEN Jump toM001.
SPO= M001→IF OV=1               THEN Jump toM001.
SPP= M001→IF ANZ1=1 AND ANZ0=0  THEN Jump toM001.
SPZ= M001→IF ANZ1=0 AND ANZ0=0  THEN Jump toM001.
```

H. Optimization of the Abstraction

To reach an optimization concerning the number of states different operations are merged. An algorithm was developed to optimize the digital operations according to the types mentioned above to merge them and extract the information relevant for the output changes. This algorithm can not be exposed in the scope of this paper because of space limitation. Its crucial steps for several operations are elucidated in the following section and through an example in Section VI. During the optimization, AKKU 1 and AKKU 2 are eliminated from the description.

The load operations can be excluded as an important step for this optimization algorithm. This exclusion occurs when the AKKUs are irrelevant to the syntax of the performance of following operations. In case that the succeeding operation is of Type 3 or Type 5 (cf. Table I) or there is a jump in the digital program to the operation assigned (Load Operation) then this exclusion can not take place.

V. IMPLEMENTATION USING JAVA AND SVG

This implementation is an extension to the previous work described in [6]. It is done in Java and it can be illustrated by the following steps:

Step 1: Initializing the Instruction, Address, Type, Operand, and Label. The raw XML mapping the tabular form of the PLC program text (compare [6] [8]) is converted to a core XML using XSL. This XML contains the Address, Label, Instruction, and Operand, together with the attributes of the instruction like the InstructionId, Type, Condition, and the

Denotation. The type of the instruction is the crucial attribute. The formalization is implemented according to the algorithms for the optimization of digital programs (compare IV).

Step 2: Splitting the Type attribute in the XML to obtain the IF-THEN-ELSE statements. The type splitting in the XML becomes very essential for constructing the algorithms.

Step 3: Constructing IF-THEN-ELSE statements (compare the examples above) with the help of the Document Object Model (DOM) [16] which makes it easy to extract information from the XML.

Step 4: Developing the FSM pane using SVG. The states and the transitions of the FSM are automatically generated from the IF-THEN-ELSE statements. SVG is used to draw the Finite state machine.

The SVG format is a new XML grammar for defining vector based 2D graphics for the web and other applications. SVG was created by World Wide Web consortium (W3C) [17] that created HTML and XML. As an XML grammar, SVG offers all the advantages of XML like internationalization (Unicode support), wide tool support, easy manipulation through standard APIs such as DOM, easy transformation through XML style sheet language transformation (XSLT).

SVG has many advantages over other image formats, and particularly over JPEG and other common graphic formats used on the web today. It is a vector format, meaning SVG images can be printed with high quality at any resolution, without the "staircase" effects. Moreover tools available like the SVG Rasterizer [18] which is part of the Batik distribution [19] can convert the SVG to raster format. The Rasterizer can convert individual files or sets of files. The provided formats are JPEG, PNG, and TIF, however the design allows new formats to be added easily.

The SVG Generator uses the DOM API to build the Finite state machines. This generator manages a tree of DOM objects that represent the SVG content corresponding to the SVGgraphics2D instance.

VI. Example

This section explains the formalization of the digital programs using the example code in Fig 3.

```
0001        :L     KB0
0002        :T     PW138
0003        :L     KM0000000010011
0004        :OW
0005        :T     PY128
0006        :L     KB85
0007  M0    :L     KB1
0008        :-F
0009        :SPZ=  M0
000A  M2    :L     PY28
000B        :T     MB225
000C        :UN    M225.7
000D        :SPB=  M2
000E        :BE
```

Fig 3. Program code in IL in STEP 5

The program is transformed to a first (raw) XML using JAVA. The result of this transformation is shown in Fig 4 where this XML is mapping the tabular form of the PLC program code. This raw XML is then transformed to the core XML shown in Fig 5. The program is converted into seven IF-THEN-ELSE statements (cf. Fig 6). Note that the IF-THEN-ELSE algorithm shown here is the one after optimization (compare section IV H), where the load operations followed by transfer operation are excluded from the IF-THEN-ELSE abstraction according to the optimization. The Load operations assigned after the labels can not be excluded since this will affect the logic of the PLC code to be manipulated. These statements are modeled to a mealy automaton of 11 states including one initial and one final state shown in Figure 7.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ILCodeBlock CodeName="digital.example"
xmlns="IL"xmlns:xsi="http://www.w3.org/2001/XMLSche
ma-instance"xsi:schemaLocation="http://www.eit.uni-
kl.de/frey/PLC/ILns.xsd" name="Code">
  <ILRow>
    <Address>0001</Address>
    <Instruction>L</Instruction>
    <Operand>KB0</Operand>
  </ILRow>
  <ILRow>
    <Address>0002</Address>
    <Instruction>T</Instruction>
    <Operand>PW138</Operand>
  </ILRow>
  <ILRow>
    <Address>0003</Address>
    <Instruction>L</Instruction>
    <Operand>KM0000000010011</Operand>
  </ILRow>
    <ILRow>
    <Address>0004</Address>
    <Instruction>OW</Instruction>
  </ILRow>
```

Fig 4. Raw XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ILCodeBlock>
   <ILRow>
    <Address>0001</Address>
    <Instruction InstructionId="Load-
    ,Transferoperation" Type="typ1"
    Condition="no  condition"
    Denotation="no denotation">L</Instruction>
    <Operand>KB0</Operand>
   </ILRow>
   <ILRow>
    <Address>0002</Address>
    <Instruction InstructionId="Load-
    ,Transferoperation" Type="typ2"
    Condition="no condition"
    Denotation="no denotation">T</Instruction>
    <Operand>PW138</Operand>
   </ILRow>
   <ILRow>
    <Address>0003</Address>
    <Instruction InstructionId="Load-
    ,Transferoperation" Type="typ1"
    Condition="no condition"
    Denotation="no denotation">L</Instruction>
    <Operand>KM0000000010011</Operand>
   </ILRow>
```

Fig 5. XML with attributes for instruction identification

```
    PW138 = KB0
  AKKU 1 = KM00000000010011OW KB0
  ANZ0 = 0 AND OV = 0
  IF AKKU 1 = 0 THEN ANZ1 = 0 ELSE ANZ1 = 1
  PY128 = AKKU 1
  AKKU 1 = KB 85
M0  AKKU 2 = AKKU 1
  AKKU 1 = KB1
  AKKU1 = AKKU 2-AKKU 1
  IF AKKU1 <= -1
  THEN IF AKKU1 < -32768
       THEN ANZ1 = 1 AND ANZ0 = 0 AND OV = 1
       ELSE ANZ1 = 0 AND ANZ0 = 1 AND OV = 0
  ELSEIF AKKU1 >= 1
       THEN IF AKKU1 > 32768
            THEN ANZ1 = 0 AND ANZ0 = 1 AND OV = 1
            ELSE ANZ1 = 1 AND ANZ0 = 0 AND OV = 0
       ELSE ANZ1 = 0 AND ANZ0 = 0 AND OV = 0
  IF ANZ1 = 0 AND ANZ0 = 0 THEN Jump to M0
M2  MB225 = PY28
  IF (N M225.7) = 1 THEN Jump To M2
BE
```

Fig 6. IF-THEN-ELSE Algorithm for the example

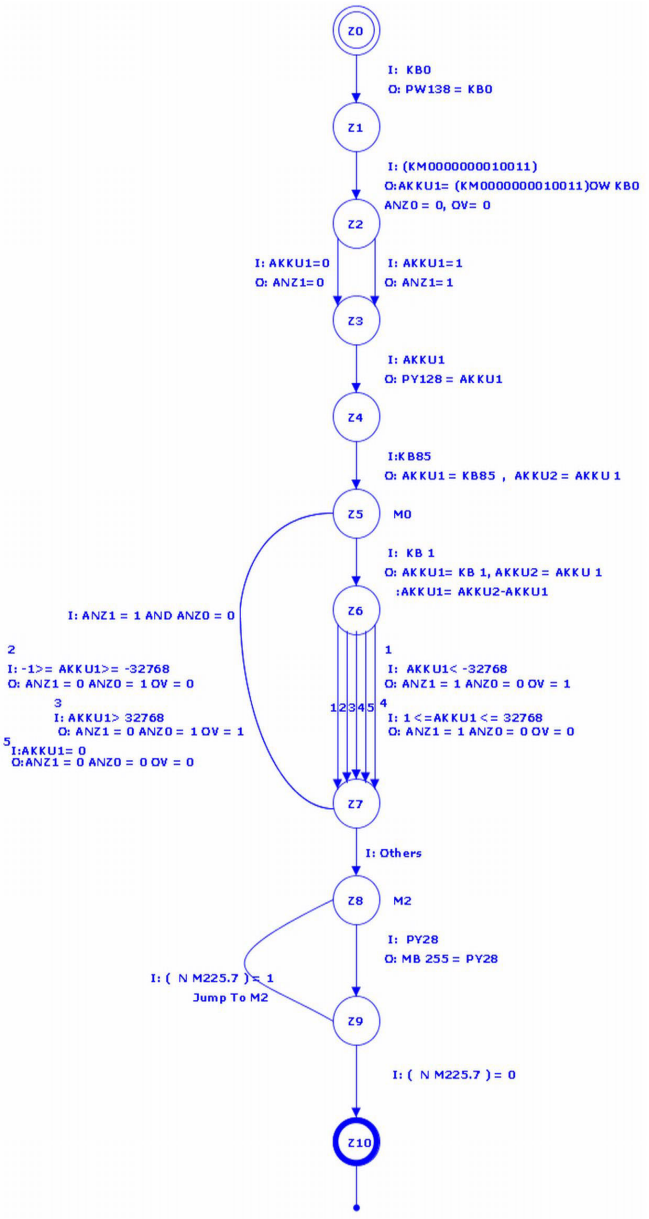

Fig 7. SVG for the FSM of the Example

VII. Conclusion and Outlook

The paper presents an approach for the formalization and visualization of non-binary PLC programs. In this, it is an extension to previous work of the authors on purely binary, i.e. logical, PLC programs. The construction of an abstracted formal model and the graphical visualization thereof allow easier understanding of the given code. The presented work serves as a basis for ongoing research in the direction of re-implementation and re-engineering of PLC programs. The automata or the IF-THEN-ELSE algorithms built from an existing PLC program will be used for the migration to other kinds of hardware or programming environments. In addition to the STEP 7 language, IEC 61131 and IEC 61499 [20] are planned target environments.

References

[1] H. Jack, *Automating Manufacturing system with PLCs*. http://claymore.engineer.gvsu.edu/~jackh/books/plcs/. Unpublished.

[2] S. Lampérière-Couffin, O. Rossi, J.-M. Roussel, J.-J.Lesage, "Formal Validation of PLC Programs: A SURVEY," *Proc. of the European Control Conference (ECC99),* Karlsruhe, Germany, Sept. 1999, paper N° 741.

[3] A. Mader and Hanno Wupper, "What is the method in applying formal methods to PLC applications?." *Proc. of the ADPM 2000 Conference,* Shaker Verlag *2000,* pp. 165-171.

[4] K-H. John, M. Tiegelkamp, *Programming Industrial Automation Systems*. Springer , 2001.

[5] XML Home Page*: http://xml.com/*

[6] M. Bani Younis and G. Frey, "Visualization of PLC Programs Using XML," *Proceedings of the American Control Conference (ACC2004),* Boston, USA, June 30 - July 2, 2004, pp. 3082-3087.

[7] XML Metadata Interchange (XMI) Home Page http://www.omg.org/technology/documents/formal/xmi.htm

[8] G. Frey, M. Bani Younis, "A Re-Engineering Approach for PLC Programs using Finite Automata and UML," *2004 IEEE International Conference on Information Reuse and Integration, IRI-2004*, Las Vegas, USA, pp. 24-29, Nov. 2004.

[9] Scalable Vector Graphics (SVG) 1.1 Specification http://www.w3.org/TR/SVG/

[10] International Electrotechnical Commission, IEC International Standard 1131-3, Programmable Controllers, Part 3, Programming Languages, 1993.

[11] Berger, H.: *Automatisieren mit S5-115U*. Berlin, München: Siemens Aktiengesellschaft, [Abt. Verl.], 1987.

[12] S5-135U/155U-*Tabellenheft*. Karlsruhe: Siemens AG, 1993.

[13] S5-115U-*Operationsliste*. Nürnberg: Siemens Aktiengesellschaft, [Bereich Automatisierungstechnik], 1992.

[14] M. Bani Younis, G. Frey, "Formalization of PLC Programs to Sustain Reliability*," Proceeding of the 2004 IEEE Conference on Robotics Automation and Mechatronics, RAM-2004*, Singapore, pp. 613-618, Dec. 2004.

[15] G. Canet et al., "Towards the automatic verification of PLC programs written in Instruction List" *IEEE Conf. on Systems, Man and Cybernetics (SMC'2000*), Nashville, USA, Oct. 2000, pp. 2449-2454.

[16] DOM website http://www.w3schools.com/dom/

[17] W3C website http://www.w3.org/

[18] SVG rasterizer web site http://xml.apache.org/batik/svgrasterizer.html

[19] Batik website Batik website http://xml.apache.org/batik/

[20] R. Lewis, *Modelling Control Systems using IEC 61499*. The Institution of Electrical Engineers, London, United Kingdom, 2001.