Proceedings of the 13th
Mediterranean Conference on Control and Automation
Limassol, Cyprus, June 27-29, 2005



# IEC 61131-3 Compliant Control Code Generation from Discrete Event Models

Gašper Mušič, Dejan Gradišar, and Drago Matko

*Abstract*— **This paper describes a control logic implementation approach, which is based on discrete event models in form of finite state machines and Petri nets. Such models may be derived during supervisory control synthesis. The approach defines a transformation of the models into an IEC 61131-3 compliant code that can be translated and downloaded into a standard industrial programmable logic controller. This way, the development and implementation phases of industrial automation projects are shortened significantly. A well proven solution libraries may be built by developed and tested models and reused when necessary.**

## I. INTRODUCTION

The development of program code for programmable logic controllers (PLC) is a central design stage in many industrial automation projects. For a long time, PLCs have been programmed in a few specialized programming languages, such as ladder diagram, but with several significant specifics, introduced by different PLC vendors. This hindered the development of general design methods that would facilitate the design of programs for complex and demanding industrial applications.

Recently, major PLC manufacturers accepted the international standard IEC 61131-3 [6] which defines common elements and syntax of several programming languages. Although the standard is not strict enough to achieve the true portability of the code among different PLCs, the structure of the programs is unified. General computer supported design methods can be developed, and adapted to a particular type of PLC by an appropriate code generator.

A PLC may be treated as a discrete event system, which changes its state (and outputs) in response to changes on its inputs and time. The process under control, connected to the controller by means of binary signals only, may be treated similarly. In the control theory there is a well established field - the supervisory control therory (SCT) - covering different aspects of control of the logical discrete event systems, i.e., systems where the ordering of events is of the primary concern [1]. Within the SCT the controller action is interpreted as a mechanism of enabling and disabling events in the system. The theory enables an algorithmic synthesis of a supervisor, given a process model and a specification model [1], [12]. The SCT uses the automata modelling framework, where an automaton is interpreted as a generator of a formal language. The synthesis of supervisors by the use of Petri nets has also been studied, e.g., [5].

G. Mušič, D. Gradišar, and D. Matko are with Faculty of Electrical Engineering, University of Ljubljana, Tržaška 25, 1000 Ljubljana, Slovenia {gasper.music, dejan.gradisar, drago.matko}@fe.uni-lj.si

Despite the sound theoretical foundation the application of the above results to PLC programming is not straightforward. The fundamental issue of investigation within the SCT is the restriction of the system's behaviour. This is well suited for designing interlocks that present a significant part of discrete control logic. For the sequential part, however, this seems less appropriate although some applications are reported (e.g. [2], [8]).

Instead of specifying allowed event sequences the desired operation of controlled system is more naturally described by a kind of flowchart. For this purpose, the IEC 61131-3 defines a specialised graphical programming language used for structuring PLC programs. It is similar to Grafcet [3], which can be interpreted as a special kind of Petri net [7]. This makes Petri net framework a good candidate for specifying event sequences in a more compact notation. A comprehensive survey on the Petri net based methods for discrete event control design can be found in [11].

Recently, a combined approach has been proposed in [9], [10]. The SCT is used to synthesize the interlock part of the control logic. The sequential part is then designed by Petri nets, which are used in a sense of formal specification. The specification is verified against the model of admissible behaviour derived during the interlock synthesis. This paper focuses on the implementation stage, where the derived automata and Petri net models are translated into a program code, that can be downloaded into a standard industrial PLC. This way, the development and implementation phases of industrial automation projects are shortened significantly.

## II. IEC 61131-3 STANDARD

IEC published the first version of the IEC 61131-3 international standard in 1993 and the second version in 2003. The standard defines common elements of programmable controller programs, including program structure, variable declarations, etc., and semantic and syntax of four programming languages. There are two textual languages, Instruction List - IL and Structured Text - ST, and two graphical languages, Ladder Diagram - LD and Function Block Diagram - FBD.

In addition, the standard defines another graphical programming language, Sequential Function chart - SFC. It is intended for structuring the code, and particularly suited for applications involving sequences of actions. It is in general used in conjunction with other programming languages. The SFC defines the general state transition structure of the program, while details on transition conditions and step actions are programmed in other standardized languages.

0-7803-8936-0/05/$20.00 ©2005 IEEE

## III. CONTROL LOGIC SYNTHESIS

As mentioned in Section I the approach described in this paper is based on a combination of the two standard paradigms used in the field of discrete event systems: supervisory control theory, used to synthesize the interlock logic, and Petri nets, used for specification of the sequential logic and verification of non-blocking.

### A. Multi-stage approach

A multi-stage approach to design of the control logic is schematically shown in Fig. 1 [8]. The specifications are split up in two parts. The first part involves prevention of undesired behaviour. It is composed of the so-called interlocks that implement measures to assure safety, coordinate subprocesses, etc. The second part of specifications deals with the sequential behaviour and defines prescribed order of tasks. The sequencing part of the control logic is only synthesized after the interlock part has been designed.

### B. Interlock part of the control logic

The set of interlock supervisors is designed within the SCT framework. Beside state machine models of the supervisors that may be easily implemented in PLC programming software, the result of the synthesis is also a model of admissible behaviour, i.e., the model of all possible event sequences in the controlled system that comply with the interlock specification. One of the key points of the approach is that this model is used as an open-loop process model when designing the sequencing part of the control logic.

### C. Sequencing part of the control logic

The sequencing controller plays a different role than the interlock supervisors. Instead of permitting or disabling the occurrence of events in the system it has to actively trigger events that result in a state change of the actuating elements of the process ($\Sigma_A$ in Fig. 1). The design of the sequencing part of the control logic may also be performed within the supervisory control theory, e.g. [2], [8], but the design approach is not as straightforward as with the interlock part.

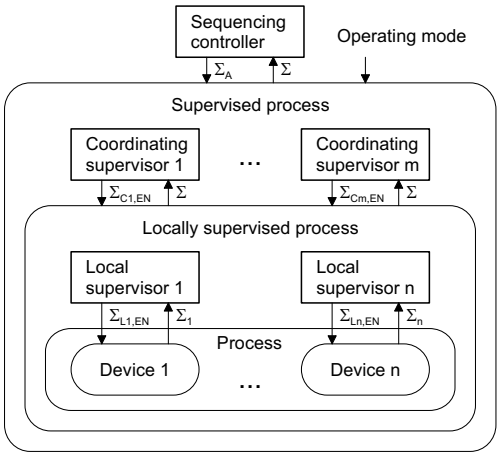


Fig. 1. Proposed control structure

An alternative way is to formalize the specification, e.g. by a Petri net, and to formally verify desired properties. To make the results of such a verification approach useful for the control, an adequate model of the process under control is needed, which is not readily available in many cases. In our approach we solve this by using a model that is developed during the interlock stage of the control logic design. The approach is described in more detail in [9].

An advantage of the Petri net representation is the straightforward path from the developed specification models to the industrial implementation. This is due to the closed relationship between SFC, Grafcet and Petri nets, which enables a SFC to be directly redrawn from a Petri net model and some of the classical analysis techniques of Petri nets can be applied also to SFCs [3].

The synthesis of this part is performed manually through modelling the sequential specification by the Real-time Petri nets (RTPN) extension [13] of Petri nets. Once the RTPN model is obtained, it is algorithmically verified for non-blocking against the admissible behaviour model obtained in the interlock design stage.

## IV. IMPLEMENTATION

The obtained models of the control logic can be implemented by any of the standardized languages for programmable logic controllers [6]. In general, the expressiveness of a particular programming language is not a problem. Special care, however, must be devoted to proper structuring of the code in order for the controller to reflect the functionality of synthesized models of the control logic. The main properties required for the implementation include: determinism, reactiveness, implementation must be deadlock-free.

The problem that has to be solved is how to overcome an inherent difference in interpretation of inputs in the SCT framework and the real-time operation of a PLC [4]. Namely, the input data to the PLC program are states of the corresponding input signals that are scanned in regular time intervals. The inputs of the formally synthesized supervisors are events, occurring spontaneously and asynchronously.

The difference may be partially solved by detecting the state transitions on the input/output (I/O) signals. The delay between an input event detection and the enabling/disabling action of the supervisor is not a problem since only the proper orderings of events are important in the classical DES theory. The time is not taken into account. The proper ordering may be easily achieved for the controllable events ($\Sigma_c$) that are generated by the PLC itself. Other events, e.g. events related to sensor readings, are considered uncontrollable ($\Sigma_u$) and are therefore never blocked by the supervisor, they only need to be observed.

There is, however, another problem. The DES theory assumes the events occur one at a time. Due to the finite sampling frequency of PLC inputs there is a common situation where two or more input signals change state between the two consecutive input scans. This implies several events are detected simultaneously. In our approach

we solve this by memorizing all simultaneously detected events and by processing the events one at a time in the supervisory part of the logic. All events are processed within a single PLC program scan. The basic requirement is that the PLC scan rate is high enough to keep the number of simultaneously detected events as low as possible. More details are given in the following subsections.

The proposed structure of the program is shown in Fig. 2. The goal was to maintain the compliance with the IEC 61131-3 on one hand and on the other hand, to put as little restrictions on the sequencing part as possible. This enables the testing of different design strategies for the sequencing part while maintaining security imposed by the interlock part of the control logic.

### A. Event detection and prioritization

The main goal of the event detection and prioritization part of the control logic (Fig. 2) is to form an event queue

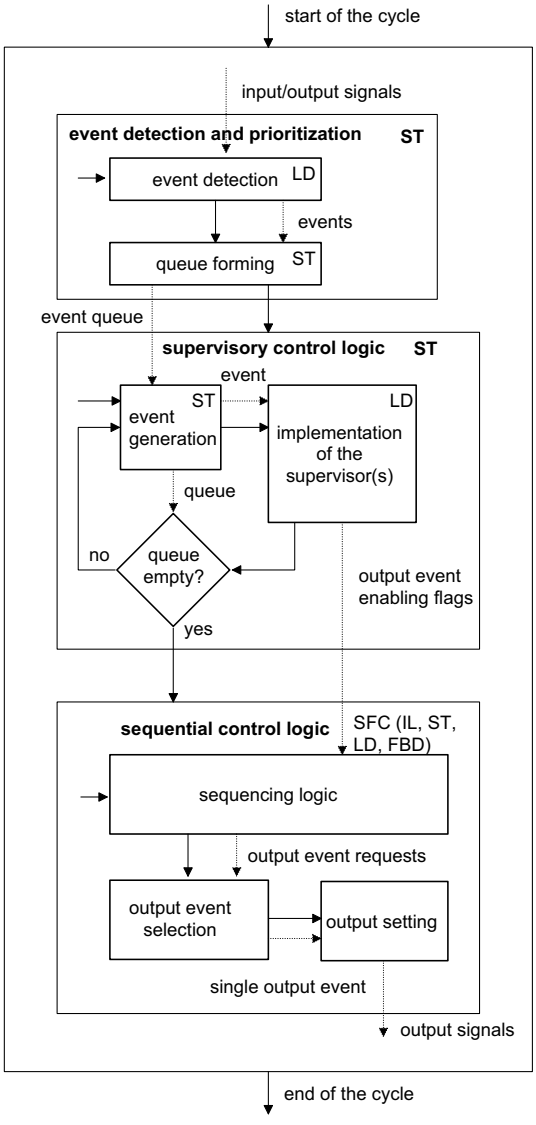


Fig. 2. Proposed PLC program structure

based on the observation of the I/O signals of the PLC. It is important to observe outputs as well, what enables to build the rest of the code modularly, i.e. every supervisory module changes its state based on the same event observation.

The main problem when designing this part is to determine an appropriate event prioritization scheme. As mentioned above, the scan rate of the PLC should be as high as possible and the number of simultaneously detected events is low. There are also several events that may not appear simultaneously due to physical setup of the system. On the other hand, in case of noninteracting devices the event order of events related to different devices is irrelevant. Nevertheless, in some cases the event order may be important. Some further comments on this topic are given in the following subsection.

This part of the code is written as a combination of ladder diagram and structured text. LD is used to build a set of event detection function blocks, a separate block for every device. The rising and falling edges of the related signals are detected and memorized. The blocks are then called from within an event queue function block, written in ST, where every event is assigned an integer code and an event queue in a form of integer array is maintained.

### B. Implementation of the supervisors

The models derived through the supervisory control synthesis are in the form of finite state machines. Every single supervisor is implemented in a separate function block. During the run-time, the blocks are called from within an event-generation loop. At every iteration of the loop an event is taken from the queue and the corresponding state transitions of all supervisors are executed. The loop terminates when there are no more events in the queue (Fig. 2). This guarantees that no event is lost.

By this procedure, correct states of the state machines of the supervisors are reached when events in the queue are related to independently operating devices. The event order may be important when some of the events are related. The formalization of this is beyond the scope of this paper, we can only give some general remarks here:

*(i)* A controllable event $\sigma \in \Sigma_c$ is detected as a result of the output change in the previous program scan. Due to the proposed program structure (Fig. 2) there can not be more than a single controllable event detected at a time and it can not be blocked at this point. Therefore we put an eventual controllable event in the queue first.

*(ii)* The processing of the uncontrollable events related to a single device ($\Sigma_{u,d} \subseteq \Sigma_u$) is correct if we can assure that $\sigma_i, \sigma_j \in \Sigma_{u,d}$ can not appear within the same input scan. This may seem a serious restriction but is generally fulfilled in practice. E.g., two events related to the same signal (rising and falling edge) never appear simultaneously. Signals related to sensors on the opposite sides of the moving parts always change one after another and in general the PLC must be fast enough to perform several program

scans in between. So this restriction is more a problem of a proper system decomposition at the modelling stage.
*(iii)* The problem remains with the interacting devices. The implementation is correct when event sequences given by the defined order of potentially simultaneous events remain within the admissible behaviour. Defining more precise and formalized rules to check this is a matter of further investigation.

The supervisors can be implemented by any standardized language [6]. In our case, ladder diagram was used. The basic idea of coding the state machine into a ladder diagram is shown in Fig. 3. The flag $Ev\_j$ (event) denotes a rising (falling) edge on the related I/O signal. A state transition is then triggered and afterwards, the $Enable$ flag is reset to assure only a single state transition occurs in one call to the function block. This way, the avalanche effect [4] is avoided. The logic could be implemented in a single rung of ladder but some compilers (including the one we use) produce warning messages when the same variable is used in the condition and the action part of the rung. The two rung implementation is used to avoid compiler messages. The additional variable $Trans$ is a temporary variable acting only within a single transition. Therefore it is not necessary to declare a new variable for every transition, a single temporary variable may be shared among all transitions.

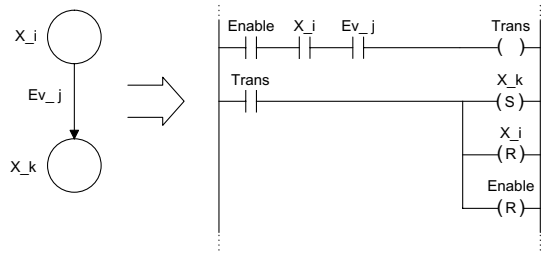


Fig. 3. Translation of a state transition into a rung of the ladder diagram

The presented idea is easily automated and a corresponding code generator was implemented in Matlab. It takes a state machine model and builds a text file representation of a ladder diagram function block compatible with GX IEC Developer, a PLC programming tool compliant with IEC 61131-3. Such a block can be then directly imported and used within any GX IEC Developer project.

### C. Implementation of the sequencing controller

The sequencing controller is implemented as a separate function block. The main difference with regard to the supervisor blocks is the interpretation of the block outputs. Instead of serving as the enabling signals for the events the outputs of the sequencing block are used as event triggers.

To facilitate the implementation of the sequencing controller function block, an automatic SFC generator was implemented in Matlab in a similar way as with the function blocks of the supervisors. Based on the RTPN specification model it builds an ASCII representation of a SFC compatible with GX IEC Developer.

### D. Additional code

Some additional code is required to obtain an operating logic controller. The generated function blocks have to be called with required parameters and a link between the outputs of the blocks and the controller outputs must be established.

To reduce the number of connections among function blocks every supervisor block receives only signals related to events that participate in the transitions between states of the related state machine, i.e. events that only appear in selfloops on the states are not considered. Similarly, only those events are taken into account on the block outputs that are disabled at least at one of the states of the supervisor or triggered by at least one of the steps of the sequencing controller. Obviously, the outputs of any block may only be related to controllable events.

The outputs of the sequencing controllers are linked by the outputs of the supervisors by a logical conjunction. Conditions for setting or resetting the controller outputs are obtained this way. At this point time delays can be inserted when necessary.

## V. EXAMPLE

As an example we show some parts of the code for the control of a modular production line. The problem and the solution are described in more detail in [10]. We focus on the distribution station, consisting of a distribution piston, that takes a workpiece from the input buffer, and a manipulator that transports the workpiece further.

The station is decomposed into three devices, besides the distribution piston, there are also the arm of the manipulator and a gripping device mounted on the arm.

To illustrate the supervisory part of the control logic, we consider the implementation of the supervisor maintaining part of the interlock between the arm and the gripper (Fig. 4). The effect of the supervisor is to prevent start of the arm movement when the vacuum grip is being switched on or off. The control of the grip is performed by a bi-stable electro-pneumatic valve, so there are two control signals - $ag1$ to switch the grip on and $ag0$ to switch the grip off.

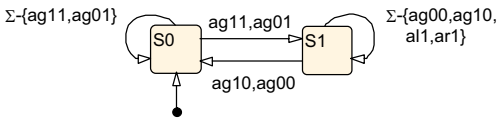


Fig. 4. An example of the interlock supervisor

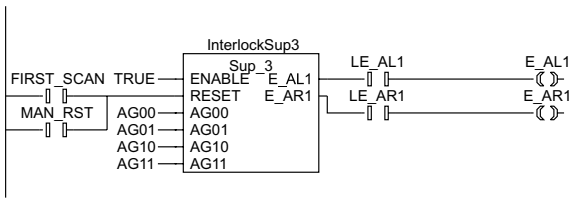


Fig. 5. Function block call of the supervisor

The supervisor is implemented as a function block (Fig. 5). Inputs to the block are $ENABLE$ and $RESET$ signals, which control the internal state transition logic, and signals $AG00$, $AG01$, $AG10$, and $AG11$ denoting possible state transitions of control signals $ag0$ and $ag1$. Related variables are the outputs of the event generation logic.

The outputs of the block are signals $E\_AL1$ and $E\_AR1$, which serve as enable signals for the controller outputs driving the arm left and right, respectively. Note that in Fig. 5 the variables $E\_AL1$ and $E\_AR1$ are set according to the result of logical conjunctions of the block outputs with variables $LE\_AL1$ and $LE\_AR1$. These are the outputs of another, so called arm local supervisor.

The internal logic of the supervisor block is shown in Fig. 6. After the initial two rungs maintaining $ENABLE$ and $RESET$ signals, there starts the state transition section as described previously. In the output section, the movement of the arm is blocked, when the supervisor is in state $S\_1$.

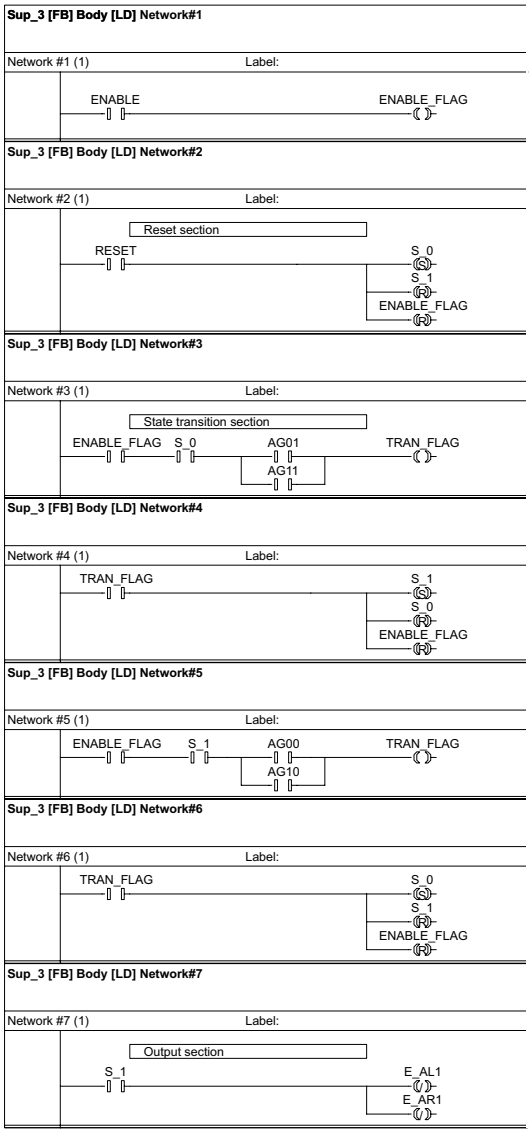


Fig. 6. Ladder diagram implementation of the supervisor

All the supervisors are implemented in a similar manner, their outputs are conjuncted, and a set of enable flags is derived this way. Finally, these flags are used in conjunction to the outputs of the sequencing controller block. This block is implemented as a Sequential function chart, based on the Petri net specification. An example of such a Petri net specification is shown in Fig. 7 and Tables 1 and 2.

After receiving a start signal and moving the arm to the initial position (left - to clear the working area of the neighbouring station), the controller checks another start signal and the presence of a workpiece to start the cyclic operation. A normal working cycle is concluded when the workpiece is carried right to the next working station and the gripper is released. Alternative paths are provided for cases when there is no workpiece but the arm has to be moved to clear the workspace and for the cases when the requested operation does not terminate in the prescribed time. In the later case an error state is entered, which can only be left after acknowledgement of the error.

With the resulting RTPN a code generator is started and the SFC built for the given case is shown in Fig. 8. There can be seen a strict correspondence between places/transitions of the RTPN and steps/transitions of the SFC. This is only possible when the RTPN is a safe and ordinary (all arc weights are 1) Petri net. We therefore require the Petri net specification of a sequencing controller must have these properties. The conflicts present in the RTPN also appear in the SFC. It is a matter of implementation platform how these conflicts are resolved in run-time, e.g.

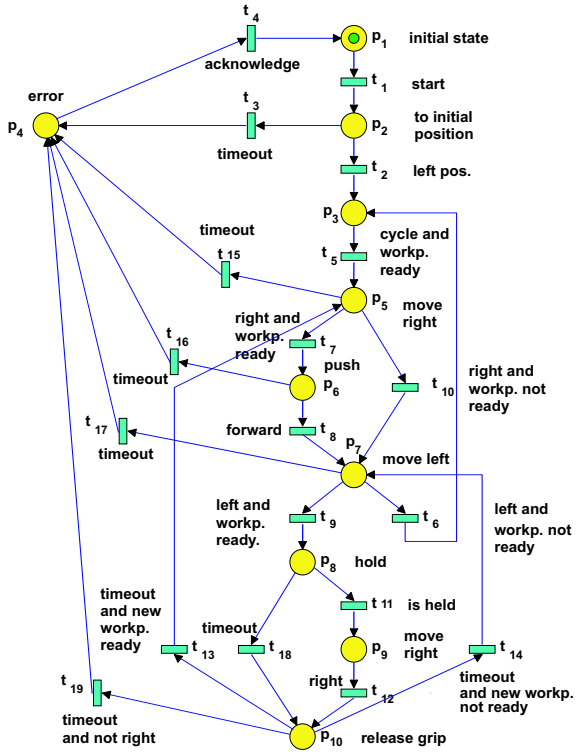


Fig. 7. RTPN sequential specification

TABLE I

RTPN TRANSITION CONDITIONS

| | |
|---|---|
| $Y(t_1)$=start | legend: |
| $Y(t_2)$=sl | ack - error acknowledgement |
| $Y(t_3)$=d4s | cycle - start of the cycle |
| $Y(t_4)$=ack | d05s - 0.5s timeout expired |
| $Y(t_5)$=cycle | (similarly: d$ns$ - $n$s timeout) |
| $Y(t_6)$=sl AND NOT si | sf/b - front/back pos. sensor |
| $Y(t_7)$=sr AND si | sg - workpiece is held |
| $Y(t_8)$=sf | si - workpiece present |
| $Y(t_9)$=sl AND si | sl - left position sensor |
| $Y(t_{10})$=sr AND NOT si | sr - right pos. sensor |
| $Y(t_{11})$=sg | start - start of operation |
| $Y(t_{12})$=sr OR d6s OR NOT sg | |
| $Y(t_{13})$=d05s AND si AND cycle | |
| $Y(t_{14})$=d05s AND (NOT si OR NOT cycle) | |
| $Y(t_{15})$=d6s | |
| $Y(t_{16})$=d1s | |
| $Y(t_{17})$=d4s | |
| $Y(t_{18})$=d2s | |
| $Y(t_{19})$=d05s AND NOT sr | |

TABLE II

RTPN PLACE ACTIONS

| | |
|---|---|
| $Z(p_1)$={(l_start, 1),(l_error, 0)} | |
| $Z(p_2)$={(l_start, 0),(al, 1)} | |
| $Z(p_3)$={(al, 0)} | |
| $Z(p_4)$={(l_error, 1),(al, 0),(ar, 0),(af, 0),(ag0, 0)} | |
| $Z(p_5)$={(ar, 1),(ag0, 0)} | af - piston forward |
| $Z(p_6)$={(ar, 0),(af, 1)} | ag0 - release the grip |
| $Z(p_7)$={(al, 1),(ar, 0),(ag0, 0),(af, 0)} | ag1 - activate the gripper |
| $Z(p_8)$={(al, 0),(ag1, 1)} | al/ar - arm to the left/right |
| $Z(p_9)$={(ar, 1),(ag1, 0)} | l_error - error indicator |
| $Z(p_{10})$={(ar, 0),(ag1, 0),(ag0, 1)} | l_start - initial st. indicator |

in our case the priority is given to the leftmost transition.

## VI. CONCLUSIONS

The presented approach enables a relatively high automation of the control synthesis for the manufacturing systems. Once the model of the plant and the specification models are developed an appropriate computer tool may perform all the necessary calculations and even generate most of the control code. A set of functions was developed in Matlab to perform the supervisory control synthesis, RTPN verification and automatic code generation in a form of IEC 61131-3 compliant function blocks. Only a small amount of additional programming is then needed to obtain an operating logic controller.

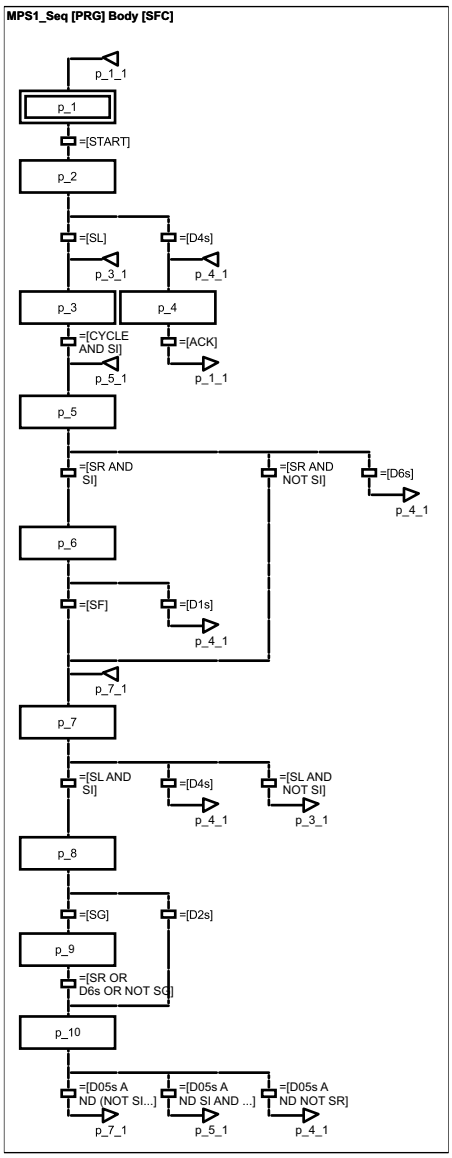


Fig. 8. Automatically generated SFC

## REFERENCES


[1] C.G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, Dordrecht, 1999.

[2] V. Chandra, S.R. Mohanty and R. Kumar, "Automated control synthesis for an assembly line using discrete event system control theory", in *Proceedings of the American Control Conference*, Arlington VA, 2001, pp. 4956–4961.

[3] R. David, Grafcet: A Powerful Tool for Specification of Logic Controllers, *IEEE Trans. on Control Systems Technology*, vol. 3, no. 3, 1995, pp. 253–268.

[4] M. Fabian and A. Hellgren, "PLC-based Implementation of Supervisory Control for Discrete Event Systems", in *Proceedings of CDC'98*, Tampa, Florida, USA, 1998, pp. 3305–3310.

[5] L.E. Holloway, B.H. Krogh, A. Giua, A Survey of Petri Net Methods for Controlled Discrete Event Systems, *Discrete Event Dynamics Systems: Theory and Applications*, vol. 7, 1997, pp. 151–190.

[6] IEC, *Programmable Controllers - Part 3: Programming Languages*. International Electrotechnical Commission, publication 61131.3. Geneva, 1993.

[7] T. Murata, Petri nets: Properties, analysis and applications. *Proc. IEEE*, vol. 77, 1989, pp. 541–580.

[8] G. Mušič, B. Zupančič and D. Matko, "Model based programmable control logic design", in *Preprints of the 15th Triennial IFAC World Congress*. Barcelona, Spain, 2002.

[9] G. Mušič and D. Matko, "Petri net control of systems under discrete-event supervision", in *ECC'03 European Control Conference*. Cambridge, UK, 2003.

[10] G. Mušič and D. Matko, "Combined synthesis/verification approach to programmable logic control of a production line", in *IFAC World Congress 2005*, to appear.

[11] S.S. Peng and M.C. Zhou, Ladder Diagram and Petri-Net-Based Discrete-Event Control Design Methods, *IEEE Trans. on Systems, Man, and Cybernetics - Part C*, vol. 34, 2004, pp. 523–531.

[12] W.M. Wonham, *Notes on Control of Discrete Event Systems: ECE 1636F/1637S 2003-2004*. Systems Control Group, Dept. of ECE, University of Toronto, 2003.

[13] M. Zhou and E. Twiss, Design of industrial automated systems via relay ladder logic programming and petri nets, *IEEE Trans. on Systems, Man, and Cybernetics - Part C*, vol. 28, 1998, pp. 137–150.