Control system design using Supervisory Control Theory: from theory to implementation

by

Matteo Cantarelli

Submitted to the Universita degli Studi di Cagliari
in partial fulfillment of the requirements for the degree of

Laurea Specialistica in Ingegneria Elettronica

Dec 2006

Advisors:

Prof. Alessandro Giua, University of Cagliari

Prof. Jean-Marc Roussel, Ecole Normale Superieure de Cachan

Dipartimento di Ingegneria Elettrica ed Elettronica

Control system design using Supervisory Control Theory: from theory to implementation

by

Matteo Cantarelli

Submitted to the Universita degli Studi di Cagliari
on Dec 12, 2006, in partial fulfillment of the
requirements for the degree of
Laurea Specialistica in Ingegneria Elettronica

Abstract

This Master thesis deals with the design of the program for a control unit using Supervisory Control Theory (SCT) introducing a systematic methodology for development of event-driven supervisory control systems. Different difficulties arises in physical implementation of SCT and unfortunately there is still a gap between the theoretical development and the number of applications of SCT in the industry. The approach presented in this thesis is based on the introduction of different behavioural models among the entries of the SCT. Both input and output electric signals of the PLC have been modelled to take into account the gap between an automaton event and a logic signal. A model of the interactions between the control unit and the process is included in the Plant model to facilitate the extraction of the controller behaviour from the supervisor. An approach to obtain from the Supervisor a *Controller* straightforwardly implementable is presented. The Controller is implemented as a Mealy machine. The obtained Controller is exploited as input for the automatic generation of the PLC control program. Two Use Case Studies have been treated to experiment the presented methodology. The first one deals with the control of an automatic gate for a car park while the second one controls a pick and place station. Both the use case studies are high detailed with built automata models and generated PLC control code.

Control system design using Supervisory Control Theory: from theory to implementation

by

Matteo Cantarelli

Submitted to the Universita degli Studi di Cagliari
on Dec 12, 2006, in partial fulfillment of the
requirements for the degree of
Laurea Specialistica in Ingegneria Elettronica

Abstract

Questa tesi ha per oggetto il progetto di programmi per controllori a logica programmata (PLC) utilizzando la teoria del controllo supervisivo (SCT), un approccio sistematico per lo sviluppo di sistemi di controllo orientati agli eventi. L'impiego della SCT per il controllo di sistemi reali presenta diversi problemi e tra il suo sviluppo teorico e le sue applicazioni pratiche esiste a tutt'oggi ancora un enorme divario. L'approccio presentato in questa tesi basato sull'introduzione di diversi modelli comportamentali tra i modelli di ingresso della SCT. Sia i segnali di input che i segnali di output del PLC sono stati modellizzati per tenere conto della differente natura di un automa ad eventi e di un segnale elettrico. E' stato introdotto inoltre un modello delle interazioni tra il PLC e il processo da controllare per rendere possibile l'estrazione del comportamento del controllore dal supervisore ottenuto con la SCT. Il comportamento del controllore, sottoinsieme del comportamento permesso dal supervisore, viene implementato in una macchina di Mealy. Il controllore cos ottenuto viene direttamente utilizzato per produrre il codice di controllo per il PLC. Due casi di studio sono stati trattati per sperimentare e validare la metodologia presentata. Il primo si occupa del controllo di un cancello automatico in un parcheggio, mentre il secondo controlla una stazione di presa e rilascio all'interno di una catena di montaggio. Nella tesi entrambi i casi di studio sono trattati in dettaglio, e si presentano sia i modelli impiegati sia il codice generato e testato sul controllore PLC.

Acknowledgments

I would like to express my gratitude to all those who gave me the possibility to complete this Master thesis.

The presented work has been born at the Ecole Normale Superieure de Cachan which retained me for an International Scholarship, allowing me to perform this research within LURPA laboratory. LURPA is an international fame laboratory in automated production which grants to its students first choice equipment and structures. I want to thank LURPA for giving me the possibility to commence this Master thesis doing the necessary research work. I am deeply indebted to my thesis co-supervisor Monsieur Jean-Marc Roussel whose help, stimulating suggestions and encouragement helped me in all the time of research and writing of this thesis.

I would like to thank the Department of Electrical and Electronic Engineering and especially my thesis supervisor Prof. Alessandro Giua for his precious support and his trust on me, without him I could not have begun this work.

My Master colleagues and all the LURPA staff who gave me a warm welcome and ensured me a pleasant stay. I want to thank them for all their friendship, help, interest and valuable hints. A special thank to my dear friend Enrico Murru with whom I have shared my whole 5 years Master and who taught me how to represent an electromagnetic field with two arms.

Especially, I would like to give my special thanks to my whole family whose support, trust and love enabled me to complete this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The rapidly growing demand on process efficiency and reliability of industrial processes has resulted in wide application of automatic control systems. Thus, research and development within different fields of automatic control systems has gained considerable attention since the first Programmable Logic Controllers appeared on the market. An important portion of todays manufacturing systems can be classified as event driven systems or discrete event systems (DESs). The notion event-driven refers to those processes where the future of the process is determined by the occurrence of process events and conditions, rather than just classical dynamics. The event-driven behaviour is found in almost all practical systems. The source of this behaviour can be different. A number of discrete sensors and actuators such as valves, motors with constant speed are usually involved in the process. Also the way processes operate can be the source of events. Plant start-up and shut-down, production changeover during otherwise continuous operation, and batch processing are examples of discrete event dynamic systems. Although it is possible to analyse and control this class of industrial systems formally, general methods applied in the industry are intuitive rather than being formal. For small sized DES control problems, intuitive methods may yield practical solutions, but as the controlled system gets larger and more complex, formal methods need to be applied. Development of practically feasible methodologies utilizing organized procedures for control and supervision of event-driven dynamic processes is therefore of great interest. This thesis attempts to contribute to this development.

1.2 Aims

The aim of the thesis is to introduce a systematic methodology for development of event-driven supervisory control systems. Programmable Logic Controllers (PLCs) have been used in industrial applications for more than 25 years and today most of the automated manufacturing systems use PLCs as control units. Therefore the potential physical platform to realise a supervisor in industry is the PLC.

Supervisory control theory (SCT) introduced by Ramadge and Wonham provides a powerful framework for control of discrete event systems. The theory enables synthesis of closed loop control systems for DESs by making some assumptions on the system that is to be controlled, and on the supervisor that is to control the system. Although SCT has received a wide acceptance in academy and some applications of SCT have been reported in the literature it has not been accepted in the industry yet. This is mainly due to the difficulties arising in physical implementation of SCT. Finite state automata are used for representing the plant model and the supervisor in SCT. Implementation of SCT necessitates an appropriate method for developing a PLC program corresponding to the automaton that represents the theoretical supervisor. Therefore implementing SCT is a matter of developing an appropriate PLC program, which will force the PLC to behave as an automaton. The methods for developing PLC codes for this purpose and the problems that may arise in doing so are discussed in. One of the main difficulties arising in SCT implementations is state-space explosion. System models are generally built as combination of subsystems and the number of states of the global system grows exponentially with the number of subsystems. Computational complexity caused by large number of states can make it impossible to design and implement a supervisor for a given DES control problem. Therefore some informal techniques are used to reduce computational complexity either by reducing sizes of the automata or using different control structures comprising more than one supervisor. Informal techniques generally depend on the designers experience and preferences such as ignoring some of the details (i.e., some of the states of the physical system) that are not necessary to appear for the given control problem. In [12] the authors show indeed how nowadays the majority of industrial developments for manufacturing system are made manually, relying on the reuse of hand written logic from previous projects, due to the similarity of both the equipments to control and the behavioural requirements. In spite of the currently competitiveness of this strategy, it will not be compatible with the incoming safety requirements for installations. The main limit of this practice is moreover that the quality of the resulting code depends totally on the expertise of the developers.

1.3 Thesis approach

As SCT is a mathematical framework capable to automatically calculate the maximal permissive behaviour of a system guaranteeing the respect of the specification, this theory seems to be ideal to obtain safe programs. The behaviour to implement in the control unit is only a part of the behaviour expressed by the supervisor, so it is necessary to develop a way to extract it. In this work a validated systematic approach to exploit the supervisor and to obtain the PLC code is presented. This approach is based on high detailed entry models, proposed to take in account the technology used in the process and to guarantee so a suitable supervisor. The possible behaviour of the system is given by a set of Deterministic Finite Automata (DFA) named *Plant model*, while the allowed behaviour is given by a set of DFA named *Specification*. From these two data inputs SCT calculates the *Supervisor*, the DFA which implements the maximal permissive behaviour for the system. Even if SCT provides a useful framework for the achievement of the supervisor, it should appear evident that as the supervisor is the result of a mathematic calculus, its quality stands totally upon the quality of its data inputs. The proposed method is described in Figure 1. The data inputs of SCT synthesis are the Plant model and the Specification, the output is the Supervisor. From the Supervisor is extracted a Controller suitable for the code generation.

In approaches found in literature the stage for the extraction of the behaviour to implement is muddled with the one for the code generation. In the presented approach, the two steps are kept distinct in order to clearly face out all the difficulties to solve in the extraction stage. These difficulties are due to both the different interpretation and the targeted objectives between a supervisor and a controller. The control of a system from a supervisor is done by the only interdiction of the controllable events emitted by the Plant model [15]. The Plant model is a "wide" event generator and the supervisor disables the controllable events which can put the system in an unsafe or undesired state, in according with the specification. The problem arises when controlling a real system as it is often required to force some events to occur and not only to enable and disable some of them. In order to do this an approach proposed by [4], used in different works, supposes that the Plant model only produces the uncontrollable events and that the controllable events are produced by the supervisor on an

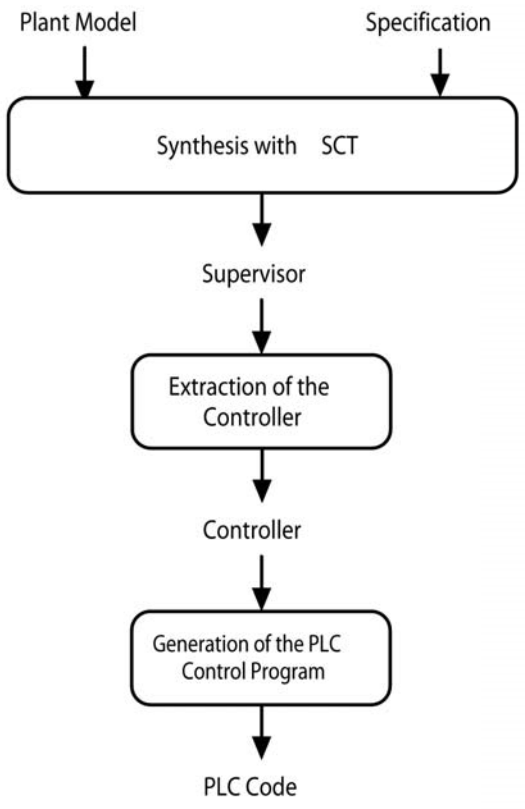


Figure 1: Proposed approach based on SCT

Input/Output perspective (Figure 2).

The approach we propose in this thesis is more general, as it keeps the original structure of controllable and uncontrollable events, permitting however to automatically determine which are the controllable events that need to be forced in order to respect the specification. This possibility is due to the used Plant model. The second difference is due to the different required objectives between a supervisor and a controller. The behaviour of the supervisor in Figure 3 is considered.

From a supervisor point of view, once that uncontrollable event `ue_a` has occurred, the only authorized behaviour is the occurrence of `ue_B` or the occurrence of one of controllable events `ce_C` or `ce_D`. From a control point of view, once that uncontrollable event `ue_a` has occurred, in the `S2` state three different behaviors are possible:

- to wait the occurrence of uncontrollable event `ue_B`,

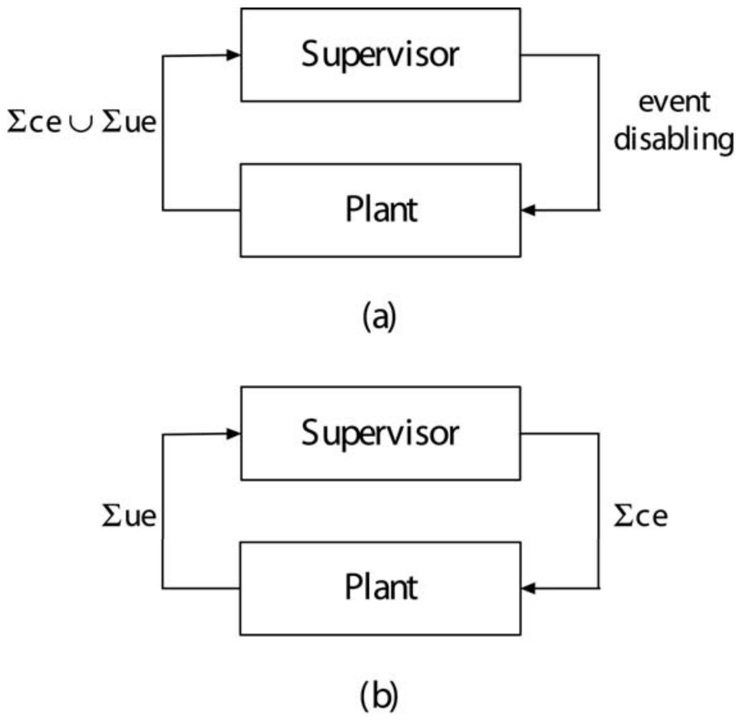


Figure 2: (a) Original interpretation (b) Input/Output interpretation (Balemi, 1992)

- to produce controllable event `ce_C`,
- to produce controllable event `ce_D`.

This behaviour is non deterministic for a controller. Does the system have to wait for the next uncontrollable event (`ue_B`) before producing the controllable one or this last is produced as first? And which one of the two different controllable events (`ce_C`, `ce_D`) has to be produced? While the supervisor represents what the system is authorized to do, the controller represents what the system has to do. This problem has been dealt in different ways in literature. For the choice among uncontrollable and controllable events a solution often proposed is to give the priority to the uncontrollable events [13, 9], while in [16] the supervisor is rejected if it is not controller deterministic too. SCT interpretation is purposely warped in [5] to let the supervisor act like an events trigger. For the choice among controllable events [11] proposes the association of different priority levels or the manual intervention of the designer. In this work it is proposed to extract the deterministic behaviour from the supervisor to obtain a *Controller* in the form of a Mealy machine. The obtained controller

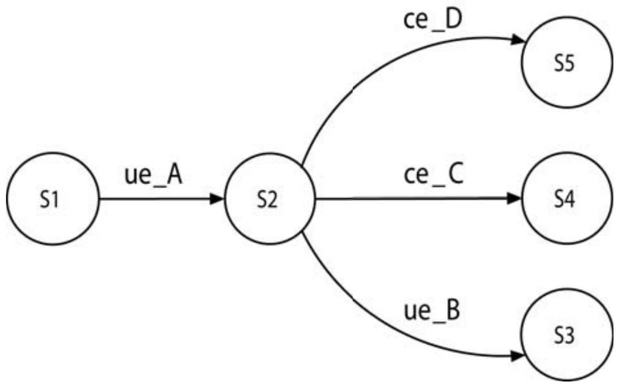


Figure 3: A deterministic behaviour for a supervisor but an indeterministic behaviour for a controller

it will be very suitable in order to obtain the PLC control program.

1.4 Outline

Chapter 2 introduces the Programmable Logic Controllers as devices to perform digital control. The basics of the supervisory control and a focus in the Supervisory Control Theory are presented in Chapter 3. The presented approach is detailed in section 3.3. A model of the control unit will introduce the concept of the PLC scan cycle in the Plant model, this will permit to define the behaviour of the logic controller solving part of the concurrence problems shown in figure 3. A way to write specifications that might force some events to appear is presented in 3.3.2. The section 3.4 shows how to obtain the controller from the supervisor, while in section 3.5 the Mealy machine is exploited to obtain the PLC code. Two case studies are presented. The first one, in Chapter 4, deals with the control of an automatic car gate. The entry models of SCT are presented, as well as the obtained supervisor and the controller. The second case study, in Chapter 5 presents the control problem for a pick and place station and the results of the achieved experiments. The case studies treated with the proposed approach have been successfully tested on a PLC. The last chapter deals with the conclusion and the perspectives of this thesis work.

Chapter 2

Programmable Logic Controllers

2.1 Overview

Computers are everywhere. As a matter of fact the routine of our daily lives is more and more supported by, and at the same time dependent on, computer driven hardware. Most of the electrical appliances we use at home, e.g., the TV set, the refrigerator, and the telephone are nowadays driven by micro-controllers or computers in general. And even computers take over the task to stabilize cars during driving or to ignite air-bags; they restrict access to certain areas or secure bank transfers, and so on. But a vast part of these systems we never see. All the things natural to us like water, food, and electricity rely on computers as far as their distribution and manufacturing process is concerned. Wherever we get directly or indirectly in touch with computers and computer driven hardware, their main purpose is: control. The control of the refrigerator temperature, the traction of the car wheels, the scheduling and transport along the assembly lines that can food, and the control of many safety-critical applications such as fission control in nuclear power plants. A question that arises is, what kind of systems are used for all these controlling processes? Mostly not general-purpose PCs but specific industrial computers. The reasons are simple and stem from the requirements of such control systems: they have to be robust, reliable and cheap. Robustness to ensure that these systems work also under unexpected conditions, which might be heat, dust, and electro-magnetic noise. Reliability in order to have them running 24 hours a day over 5 or even 10 years. And they have to be cheap to enter a mass market and to reduce the costs of the manufacturing process. Next to micro controllers, specific purpose computers on a single chip, one prominent class of industrial controllers are so called programmable logic controllers (PLC). First developed during the early 1970s, PLCs started as simple devices to replace electro-mechanical relays. Using integrated circuit technology, they performed simple sequential control tasks, in isolation from other control and monitoring equipment. These simple devices have grown into complex systems capable of almost any type of control application, including motion control, data manipulation, and advanced computing functions. Nowadays, PLCs are extensively used in the field of automation and they are integrated into much larger environments, requiring communication with other controllers or computer equipment performing plant management functions. Control systems driven by PLCs are often complex, safety critical, and involve a lot of money. Any failure of these systems might not only result in a significant financial loss but lead to casualties as well. Hence, next to robustness and reliability, their actual programming and the correctness of the programs plays a vital role. Since PLCs have been for a long time subject to the engineering community only, their programming languages evolved out of historical needs. This means, that they are particular suitable to implement controlling tasks as the solution to control theoretic problems. Therefore, the used programming languages are often close to relay ladder logic or assembly language. In contrast to computer science, where a number of new programming languages evolved, driven by the latest theoretical results, industrial programs often follow the well-known and established concepts of the first programming languages. This means, that high-level languages including object-orientation, abstract data-types, and graphical programming environments, which support a more abstract and problem-oriented approach, have not or to a much lesser extent found their way into industrial and PLC programming. In this sense industrial programs are much more exposed to design errors and more likely to fail than state-of-the-art programming languages which demand a strict programming regime and foster a more abstract development process. There are at least two ways to improve the current situation.

- One way is to design new programming languages and environments for PLC programming. These languages should be high-level to allow a development on an abstract, problem-oriented level and at the same time they should be simple and rigorous enough to enable a clear understanding and to support the development of safe programs right from the beginning.

- Another approach is to keep the existing programming languages and to develop analysis and verification methods for PLC programs. These methods should then aim at proving the correctness of PLC programs and detecting even subtle bugs.

In this thesis we follow the second approach standing up on the SCT framework. The reasons for this are simple: today there are many PLC programs in existence which will not be rewritten; however, there is a chance that they might be re-analyzed. Moreover, it is not likely that people change domain specific languages easily since there is a long tradition in using them and it takes time to build confidence in new languages. On the other hand, a clear understanding of the semantics of the current languages and tools to analyze the developed programs is much more likely to make an impact on PLC programming.

2.2 PLCs and PLC Programming Languages

The hardware of a PLC consists of a microprocessor based CPU, a memory, input and output ports where signals can be received, e.g., from switches and sensors, and sent to actuators, e.g., to motors or valves. A PLC is equipped with an operating system that allows to load and run programs and perform self-checks. Traditionally, the PLC operating system must respond to interrupts and must be real-time. Programs for PLCs are developed and compiled on external devices and downloaded to them afterwards. The main difference to conventional systems such as PCs is the cyclic operation mode: PLC programs are executed in a permanent loop. Although PCs have also input/output functionality to exchange information with, e.g., a keyboard or a mouse, this I/O functionality is not their main reason of existence. From a computer science perspective, PLCs have the characteristics of real-time, reactive systems. If their environment is taken into account they resemble hybrid systems.

2.2.1 The Central Processing Unit

The central processing unit (CPU) is the part of a programmable controller that retrieves, decodes, stores, and processes information. It also executes the control program stored in the PLCs memory. In essence, the CPU is the brains of a programmable controller. It functions much the same way the CPU of a regular computer does, except that it uses special instructions and coding to perform its functions. The CPU has three parts:

- the processor,
- the memory system,
- the power supply

The processor is the section of the CPU that codes, decodes, and computes data. The memory system is the section of the CPU that stores both the control program and data from the equipment connected to the PLC. The power supply is the section that provides the PLC with the voltage and current it needs to operate.

2.2.2 The Input/Output System

The input/output (I/O) system is the section of a PLC to which all of the field devices are connected. If the CPU can be thought of as the brains of a PLC, then the I/O system can be thought of as the arms and legs.

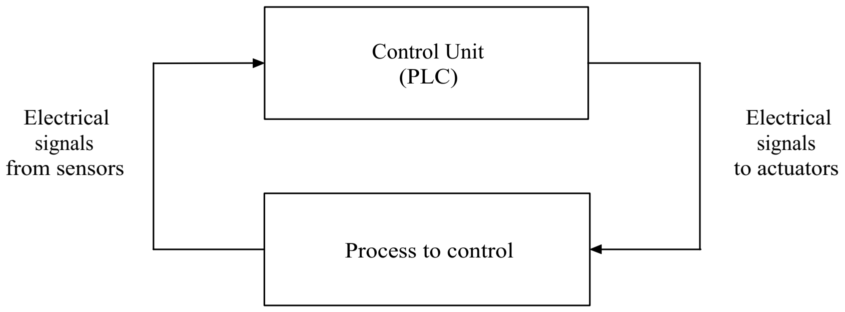

Control Unit
(PLC)
Electrical signals from sensors
Electrical signals to actuators
Process to control

Figure 1: I/O system in a PLC

The I/O system is what actually physically carries out the control commands from the program stored in the PLCs memory. The I/O system consists of two main parts:

- the rack
- I/O modules

The rack is an enclosure with slots in it that is connected to the CPU. I/O modules are devices with connection terminals to which the field devices are wired. Together, the rack and the I/O modules form the interface between the field devices and the PLC. When set up properly, each I/O module is both securely wired to its corresponding field devices and securely installed in a slot in the rack. This creates the physical connection between the field equipment and the PLC. In some small PLCs, the rack and the I/O modules come pre-packaged as one unit. All of the field devices connected to a PLC can be classified in one of two categories:

- inputs

- outputs

Inputs are devices that supply a signal/data to a PLC. Typical examples of inputs are push buttons, switches, and measurement devices. Outputs are devices that await a signal/data from the PLC to perform their control functions. Lights, horns, motors, and valves are all good examples of output devices.

There are two basic types of input and output devices:

- discrete
- analogue

Discrete devices are inputs and outputs that have only two states: on and off. As a result, they send/receive simple signals to/from a PLC. These signals consist of only 1s and 0s. A 1 means that the device is on and a 0 means that the device is off. Analogue devices are inputs and outputs that can have an infinite number of states. These devices can not only be on and off, but they can also be barely on, almost totally on, not quite off, etc. These devices send/receive complex signals to/from a PLC. Their communications consist of a variety of signals, not just 1s and 0s.

In a traditional industrial control system, all control devices are wired directly to each other according to how the system is supposed to operate. In a PLC system, however, the PLC replaces the wiring between the devices. Thus, instead of being wired directly to each other, all equipment is wired to the PLC. Then, the control program inside the PLC provides the wiring connection between the devices. The control program is the computer program stored in the PLCs memory that tells the PLC whats supposed to be going on in the system. The use of a PLC to provide the wiring connections between system devices is called soft wiring. There exist different programming languages for PLCs. They have been designed with an emphasis to implement control tasks, each intended for a specific application domain, and based on the background of the control engineers who use them. In order to achieve more conformity of the different PLC programming languages the standard IEC 61131-3 was developed [1].

- **Instruction List (IL)** is a low level assembly language with one register called current result. IL is still one of the main programming languages for PLCs used in Europe, since it allows compact code and is very close to hardware programming.

- **Structured Text (ST)** is a PASCAL-like language. It is a higher level language than IL and provides more convenient structuring and organizing constructs such as while-loops, if-then-else-conditionals and so on.

Moreover, three graphical languages are defined:

- **Ladder Diagrams (LD)** sometimes also called Relay Ladder Logic. Indeed, LD programs resemble relay logic. The diagrams are drawn in a graphical editor, e.g., on a PC, and then compiled and moved to the PLC. Like IL the LD programming language is very low level, but serves its purpose whenever relay ladder logics have to be compiled into software. Until today LD programs are the major programming language for PLCs used in the U.S.

- **Function Blocks (FB)** are essentially a data-flow language for describing a network of functions connected by signals. Pre-defined functions, generally drawn as boxes, are combined by connecting inputs and outputs in a desired manner. They are interpreted along their flow of control, i.e.,, function blocks are interpreted along their connecting paths and provide a better overview and understanding of how functions relate than, e.g., LD programs.

- **Sequential Function Charts (SFC)** is a graphical, high-level programming language for PLCs which aims at providing a clear understanding of the possibly interwoven program parts. SFCs allow to decompose and structure program parts and include interesting concepts such as parallelism, activity manipulation and hierarchy. Historically, they take over ideas from Petri nets and Grafcet.

Most of these programming languages have additional PLC specific features and allow the use of timers and to access system time. It is to be expected that in several application areas PLCs will be replaced by PCs in the future. However, the main principles such as a cyclic operation mode and the use of timers will be maintained. Also it is unlikely that there will be a change of programming languages in the near future. One indication for this is that although higher-level languages such as ST are standardized, still the lower-level languages such as IL and LD are most commonly used. One exception to this are SFCs.

2.3 Structured Text Language

Structured Text or ST is a high level language which has a syntax that on first appearance is very similar to PASCAL. Although there are some minor similarities to PASCAL, ST is a distinct language that has been specifically developed for industrial control applications. The ST language has a comprehensive range of constructs for assigning values to variables, calling functions and function blocks, for creating expressions, for conditional evaluation of selected statements and for iteration, i.e., repeating selected sections of code. The language statements can be written in a fairly free style where tabs, line feed characters and comments can be inserted anywhere between keywords and identifiers, i.e., wherever a space character is required. For the software developer, ST is fairly straightforward to learn and use. It is a language that is easy to read and understand, particularly when written using meaningful identifiers and well annotated with comments. Structured Text is a general purpose high-level language for expressing different types of behaviour involving a variety of different types of data. It is particularly useful for complex arithmetic calculations. An example program is shown in Figure 2.

One important difference between ST and traditional programming languages is the nature of program flow control. A ST program will be run from beginning to end many times each second. A traditional program should not reach the end until it is completely finished. In an ST program as in any other language a particular situation might into an infinite loop. If this were to happen during a control application the controller would stop responding, the process might become dangerous, and the controller watchdog timer would force a fault. The language is composed of written statements separated by semicolons. The statements use predefined statements and program subroutines to change variables. The variables can be explicitly defined values, internally stored variables, or inputs and outputs. Spaces can be used to separate statements and variables, although they are not often necessary. Structured text is not case sensitive, but it can be useful to make variables lower case, and make

```
PROGRAM

IF (STATE=0) THEN
        IF (Start) THEN
                SET GoDown;
                NEXTSTATE:=25;
        END_IF;
END_IF;

IF (STATE=25) THEN
        IF (NOT Up) THEN
                NEXTSTATE:=28;
        ELSIF (NOT Start) THEN
                NEXTSTATE:=30;
        END_IF;
END_IF;

[...]
[...]
[...]

STATE:=NEXTSTATE;

END_PROGRAM

[EOF]
```

Figure 2: An extract of the generated PLC code in ST language

statements upper case. Indenting and comments should also be used to increase readability and documents the program. ST programs allow named variables to be defined. This is similar to the use of symbols when programming in ladder logic. The ST language has been retained in this work for both its structural nature and the easy way with whom an ST program can be automatically generated.

Chapter 3

Supervisory Control Problem

3.1 Introduction

The application of automatic control techniques have demonstrated a significant improvement in the efficiency of industrial systems. However, the transition from manual to automatic control has given rise to a problem of system complexity that in turn may reduce the reliability of the overall system. This problem is mostly relevant for large scale systems such as industrial plants, where process safety is a very important aspect of plant operation. In order to satisfy the safety requirements, the demands on the plant operator capabilities also increase in form of experience, skills, operation overview, resolution, quickness etc. as the plant complexity increase. Since training of such skills is a rather difficult task, alternative solutions are sought to overcome the problems of plant complexity and to support the plant operators. The rapid development of computer technology have opened new possibilities for this challenging problem. Discrete event systems are systems that are characterized by the sequences of events that they can accept or execute. From an abstract point of view it is not important whether the system actually generates events or whether it restricts the possible events that are generated somewhere else. The important point is that the behavior of the system can be described by sequences of events. Control of discrete event systems was first introduced by P.J. Ramadge and W.M. Wonham [15]. In the framework proposed by Ramadge and Wonham, a system is described by the sequences of events that it can generate, i.e., the language. Also the sequences that represent completed tasks are taken into account. The entries of SCT are composed by two class of automata models: the plant model and the specifications. While the plant model represents what the system is phisically capable to do, the specifications states the allowed behavior. SCT output is the supervisor which describes the maximal permissive behavior of the plant model with respect to the specifications. This thesis presents an approach to obtain a suitable supervisor in order to be able to produce a straightforward implementable control program.

3.2 Languages and Automata

In the literature several models are proposed to describe the behavior of discrete event systems. To name a few: languages, automata, transition systems, Petri nets, boolean expressions, process algebras, etc. In this thesis we will use representations based on languages and automata.

Let Σ denote the finite set of all possible events or event labels. A set of events is usually called an *alphabet*. A trace or string is a finite sequence of events, $\sigma_1\sigma_2...\sigma_n$ with $\sigma_i \in \Sigma$ for all $i \in 1...n$. The *length* of a trace is the number of events in the trace. Let ϵ be the empty trace, i.e., the sequence of events with length 0. Note that $\epsilon \not\in \Sigma$. Σ^n denotes the set of traces with length n. Let $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^n$. It denotes the set of all finite traces with events in Σ. Let $\Sigma^+ = \bigcup_{i=1}^{\infty} \mid \Sigma^n = \Sigma^* - \epsilon$. A language is a set of traces, i.e., a subset of Σ^*. The language of discrete event system A is denoted by $L(A)$.

An automaton is a representation of a discrete event system based on states and transitions between states. In the literature also the terms labeled transition system, finite state machine, and generator are used. The differences between these models are relatively small. In this thesis we will use the term automaton to denote all these forms.

A *Finite State Machine* or *automaton* in its basic form is a five tuple

$$X = (\Sigma(X), Q(X), \delta(X), Q_0(X), Q_m(X)),$$

where
$\Sigma(X) \subseteq \Sigma$ is the set of events,
$Q(X)$ is the set of states,
$\delta(X) : Q(X) \times \Sigma(X) \rightarrow 2^{Q(X)}$ is the transition function,
$Q_0(X) \subseteq Q(X)$ is the set of initial states.
$Q_m(X) \subseteq Q(X)$ is the set of final or marked states.

When needed, extra elements will be added to this basic form. In the sequel the notation $\delta(X, q, \sigma)$ will be used instead of $\delta(X)(q, \sigma)$.

Automata can be seen as machines that generate sequences of events. Initially an automaton is in one of its initial states. From state q automaton X can make a transition to state $q^{'}$ while generating event σ if $q^{'} \in \delta(X, q, \sigma)$. If $\delta(X, q, \sigma) = \emptyset$ then X cannot make a transition from state q labeled σ. The set of traces that automaton X can generate in this way is called the language of automaton X, and denoted $L(X)$. The term "language of an automaton X" can be regarded as short for "the language of a discrete event system represented by automaton $X^{'}$". The transition function can be extended naturally to traces, languages, and sets of states. Let $q \in Q(X), \vartheta \subseteq Q(X), \sigma \in \Sigma(X), s \in \Sigma(X)^*$, and $K \subseteq \Sigma(X)^*$.

Define
$\delta(X, \vartheta, \epsilon) = \vartheta$,
$\delta(X, \vartheta, \sigma) = \bigcup_{q' \in \vartheta} \delta(X, q^{'}, \sigma)$,
$\delta(X, \vartheta, s\sigma) = \delta(X, \delta(X, \vartheta, s), \sigma)$,
$\delta(X, \vartheta, K) = \bigcup_{s \in K} \delta(X, \vartheta, s)$

The two language of automaton X are formally defined by
the generated language: $L(X) = \{s \in \Sigma(X)^* : \delta(X, s) \neq \emptyset\}$
and the accepted language: $L_m(X) = \{s \in \Sigma(X)^* : \delta(X, s) \in Q_m(X)\}$.
The definition of automaton given above is sometimes in the literature referred to as *non-deterministic automaton.* These automata are called nondeterministic because from the observation of a generated trace it cannot be uniquely determined which state the automaton has reached after execution of this trace. An automaton, X, is called deterministic if the initial state set is a singleton set, and if at each state $q \in Q(X)$, and for each event $\sigma \in \Sigma(X), \delta(X, q, \sigma)$ contains at most one element. After observation of a trace $s \in L(X)$, with X a deterministic automaton, it is always clear which state X has reached. Namely, the only element of the singleton set $\delta(X, s)$. (So, we will write $q = \delta(X, s)$ instead of $q = \delta(X, s)$.)

3.3 Supervisory Control Theory: Presented approach

SCT is the theory of analysing discrete-event control systems in a formal manner. This theory describes the fundamental issues and methodologies for performing such analysis. In SCT the process is modelled as a state transition structure, typically a Finite State Machine (FSM), in which transitions are labelled as controllable (those that can be disabled by external intervention) and uncontrollable (those that cannot be prevented from occurring). Control actions are exerted upon the process in feedback mode by another FSM termed supervisor that disables controllable transitions in order to satisfy given behaviour specifications. In other words the supervisor disables process controllable transitions in such a way that it restricts the process behaviour to a subset of all the possible trajectories. In general a design problem can be defined as: given a specification, find an implementation that satisfies the specification. SCT uses the synchronous product (or composition) $||$ in the supervisor syntehsis. For a formal definition of this operator see [7]. A design problem can be considered a supervisory control problem if the implementation consists of an already existing uncontrolled process G and a still to be designed supervisor process S. In this section we will define the control problem of finding a supervisor S such that $G \,||\, S$ can replace a given specification process E. The basic supervisory control problem can be formulated as follows. Given an uncontrolled system G and a specification E, find a supervisor S such that $G \,||\, S \subset E$. In some applications the supervisor does not have the ability to block all events. For instance if an alarm event is executed when some water level exceeds a threshold, then this event can be observed by the supervisor but it cannot be blocked. If this event has to be prevented from occurring then somewhere else in the system some other events have to be blocked (for instance the event corresponding with the closing of a waste gate) such that the alarm event cannot be executed. Usually the presence of uncontrollable events is modeled by splitting up the event set into controllable and uncontrollable events, Σ_c and Σ_{uc} respectively. In SCT the possible behavior of the system is given by a set of Deterministic Finite Automata (DFA) named *Plant model*, while the allowed behavior is given by a set of DFA named *Specification*. From these two entries SCT calculates the *Supervisor*, the DFA which implements the maximal permissive behavior for the system.

3.3.1 Design of the Plant Model

This step is the most difficult one and it deserves the maximum care. The Plant model must represent all the possible behavior of the system, i.e., what the system is physically capable to do. In this approach the Plant model describes the behavior of the process controlled by a control unit with a "random" program. While in literature the Plant model describes only the behavior of the process, in this work several generic models are added to the Plant model to explain how the process and the control unit interact. This strategy is moreover competitive in order to precisely define the right level of abstraction which will be suitable to produce the system control laws. The separation of each basic process model is essential to minimize the errors and to favour a reuse orientation.

The Input and Output models of the Control Unit

As the real system exchanges electrical signals, the first step is to identify the models in order to represent the behavior of inputs and outputs of the control unit. Two events are associated to each electric signal [17]: one for the rising edge (change from 0 to 1) and one for the falling edge (change from 1 to 0). These events will be uncontrollable for the inputs (the values are imposed by the process) and controllable for the outputs (the values are imposed by the control unit). The proposed models for inputs and outputs (Figure 1) are not the same in order to take in account the problem of the initial value.

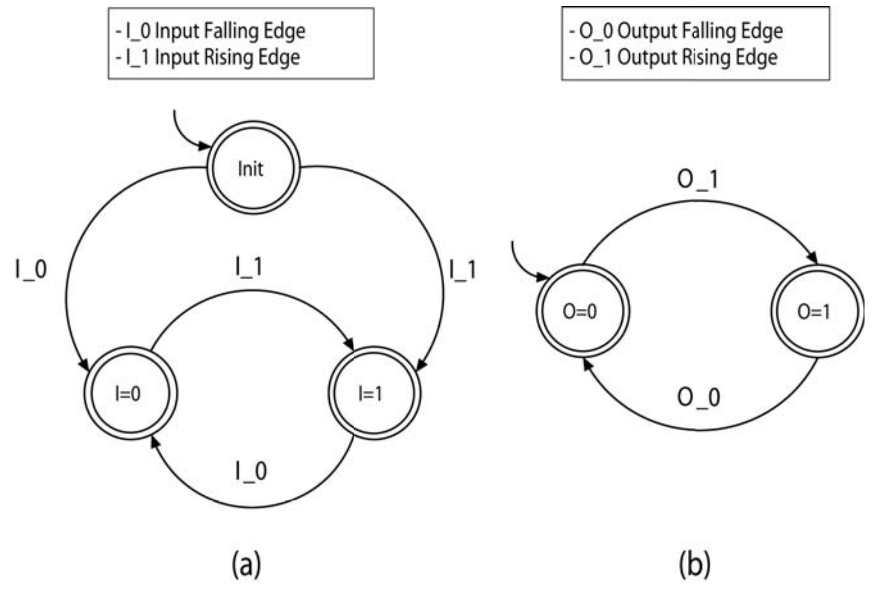


Figure 1: (a) Input model (b) Output Model

As the initial state of inputs is unknown the proposed model has three states. For the outputs a model with two states is sufficient as the initial state can be considered, without lack of generality, as zero. These models are generic.

The Control Unit model

The interactions between the process and the control unit represent a key point in the definition of the Plant model. An automaton model which represents these interactions permits to express the desired behavior of the PLC. If this behavior is explicitly modelled as a part of the Plant model there will be no room left for any kind of ambiguity or handmade choice while extracting the controller's behavior. The synchronization of the control unit model with the process will bring as result the separation between events belonging to different PLC cycles. In this paper two models are proposed for the control unit. In both of them the control unit implemented in the PLC is considered as an infinitely reactive system i.e., it can distinguish input changes and can always calculate the consequent output changes. This is possible if the scan cycle time of the control unit is short enough in comparison with the time constants of the process, which is a common and appropriate assumption.

Single Input Multiple Output (SIMO)

The SIMO model permits to obtain from the control unit a behavior where in each PLC scan cycle a single input is read and multiple outputs can be processed. The proposed model (Figure 2) has two states: `Wait` and `Exe`. The `Exe` state represents the execution of the user program, during which the control unit changes its outputs. In the `Wait` state the outputs are held fixed while the process is allowed to change only one input. Controllable event `End` declares the end of the PLC cycle. After the synchronization each cycle will be so separated by the `End` event. This controllable event will permit besides the writing of specifications which requires the emission of a particular event (see Section 3.3.2). The SIMO model gives implicitly the priority to the controllable events i.e., the outputs of the system.

Multiple Input Multiple Output (MIMO)

As in real system several inputs may change during the same PLC scan cycle a MIMO model is proposed too. The automaton (Figure 3) has two states: `Wait` and `Exe`. The uncontrollable event `Read` has been added as to indicate the conclusion of the reading operations. Many

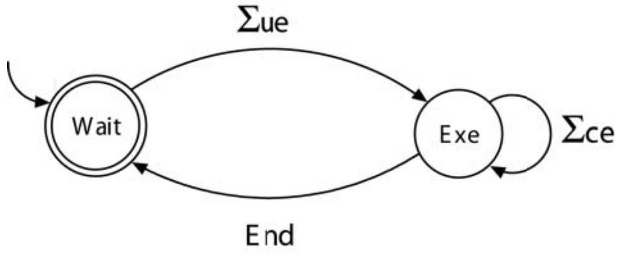


Figure 2: SIMO automaton model

outputs can be processed, the `End` event declares the end of the PLC cycle. As for the SIMO model the `End` event will separate different PLC cycle, while the `Read` event will separate, within a cycle, the reading from the execution step.

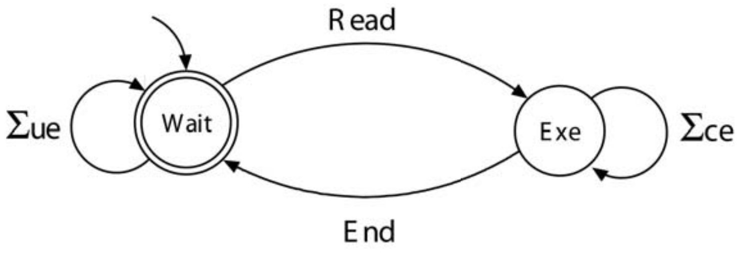


Figure 3: MIMO automaton model

The choice between SIMO and MIMO, in agreement with the system requirements, is up to the designer. The experiments we have performed have shown that the chosen model affects the process models to take in account and the obtained supervisor.

The Process models

Once determined the inputs and the outputs of the system, the automata models, in order to define the behavior of the process, have to be provided. The first common error in this step is a wrong choice of the level of abstraction. An high level abstraction which doesn't permit to distinguish the technology will be a certain source of error (in literature, for example, the same model is often used for both a double and a single acting cylinder). A model for each elementary component should be provided and more complex models should be obtained by these once that the synchronization events are added. The second mistake usually made is to not clearly separate the process behavior from the control logic. The process model has to provide all the degrees of freedom which are offered by the real component; if the process models don't describe also the dangerous behavior there is no guarantee that the supervisor can steer away from it. Often this is ignored from designers who are already thinking about what to obtain rather than thinking about what is really possible given the component at hand.

3.3.2 Design of the Specification

The writing of the specifications has always represented a common problem in the community, whichever modelling class is used. The main deal is to translate something usually expressed in natural language into a formal language used as control model. A further problem is introduced by the use of SCT: as SCT works only upon the disabling of controllable events when the specification aim is to force some controllable event to appear no possibilities are provided. To solve this problem event `End`, purposely introduced in the control unit model, is used. When a controllable event has to be forced to appear it is sufficient to prevent the end of the cycle if that event is not previously emitted. This shows, from a different point of view, the utility of the introduced control unit model: if the concept of the scan cycle is introduced and its behavior modelled in the Plant model the engineer will have at disposal a tool to make up different lacks of the theory, being so capable to completely exploit SCT in order to control a real system.

During the synthesis SCT disables the events that can not lead the system to a marked state. In order to exploit the marked states in this approach they are used to represent the states where the system has finished the task for which it had been started. As immediate result, in the supervisor, all the paths which don't allow the system to reach the end of the task will be forbidden. In Figure 4 an example of a specification shows the presented points.

The aim of this specification is to force `Output` event when `Input` event is read. As the `End` event is forbidden in the state `S2` the only way to end the current PLC cycle is to first produce the `Output` event. The state `S1` finally is the only marked one and it represents both the arrival and the starting point of this basic task. As stated by SCT, the set of marked states of the Supervisor will be the intersection between all the marked states of the single models.

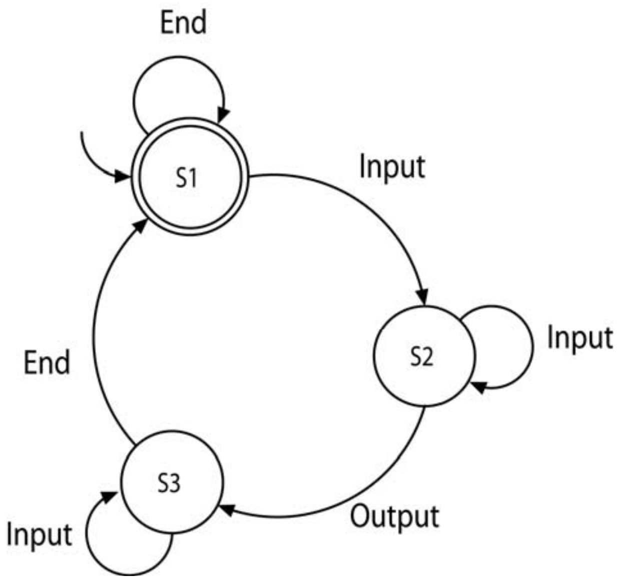
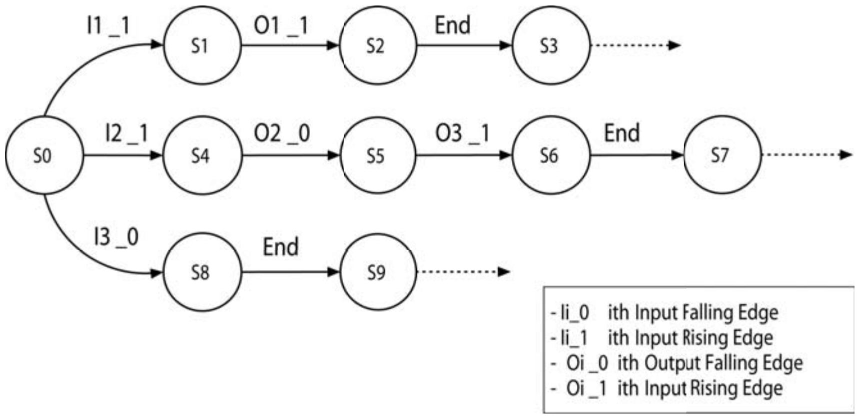


Figure 4: Example of specification

3.4 From Supervisor to Controller

The data output of SCT synthesis is the Supervisor. By first the Supervisor is minimised by language equivalence in order to reduce the number of states. As the Supervisor represents the maximal possible behavior of the system it is necessary to reduce this behavior in order to obtain the desired one, i.e., what the system has to do. The problem to solve is the concurrence among events in order to choose, among all the safe behaviors expressed by the supervisor, the one which has to be implemented. The proposed strategy is to minimise the number of PLC cycles needed to realize the desired task. In section 3.3.2 the states that represent the end of the task have been proposed to be marked. The proposed strategy can so be easily implemented minimising the number of PLC cycles needed to reach a marked state. In order to do this a weight is associated to each state. The weight expresses the distance, in number of PLC cycles, from the current state to the closest marked one. An algorithm, when a choice in terms of controllable events is present, will choose then the event that brings the system in the state with the less weight associated. If this strategy was not sufficient in order to remove all the choices between controllable events some order relations between events or further specifications can be added. The figure 5 shows an example of Controller obtained with a SIMO control unit model.



I1 _1
O1 _1
End
I2 _1
O2 _0
O3 _1
End
I3 _0
End
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
- Ii_0 ith Input Falling Edge
- Ii_1 ith Input Rising Edge
- Oi _0 ith Output Falling Edge
- Oi _1 ith Input Rising Edge

Figure 5: Example of DFA Controller

The behavior obtained with the SIMO model is made up of sequences of events composed by an uncontrollable event, a list of controllable events and closed by event `End`. Each sequence corresponds to an execution of the PLC program during a PLC cycle. In state `S0` the Controller is at the beginning of a PLC cycle. If event `I1_1` is read then the Controller will have to put to one the output 1 emitting event `O1_1`. Event `End` will declare the end of the current cycle and the next PLC cycle will begin from the state `S3`. This same behavior can be represented more compactly by a Mealy machine due to the following properties:

- only the final PLC state is important for the code,
- each sequence can be represented by a single transition,
- if several outputs are emitted their order is not important as they will be processed during the same PLC cycle.

During the conversion to the Mealy machine the `End` event will be ejected as from now each cycle will be represented by a single transition. The Mealy machine obtained from the DFA Controller in Figure 5 is presented in Figure 6.

3.5 From Controller to Code

The obtained Mealy machine is now really suitable to be processed in order to produce the code for the PLC. As the Mealy machine is a generic model that does not put constraints

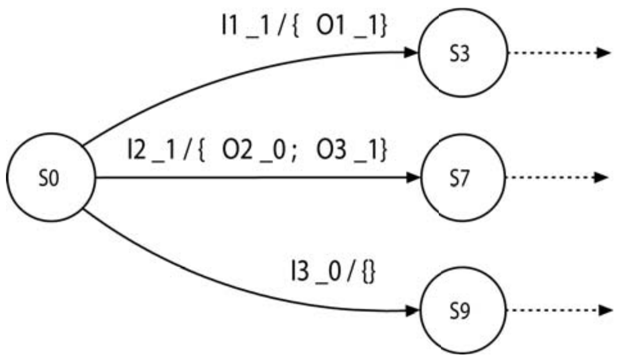


Figure 6: The obtained Mealy machine controller

on the implementation different programing languages could be used. In this work the implementation is made, in conformance with the IEC 61131-3 standard , using the Structured Text (ST) language. The ST language has been retained because of its control flow instructions and its purely textual nature which facilitates the stage of code generation and the exportation towards the programming frameworks. The extract of code, corresponding to the Mealy machine in Figure 6, is presented in Figure 7.

Two integer variables are required to code the Mealy machine: `STATE` for the value of the current state at the beginning of the PLC cycle and `NEXTSTATE` for the value of the state at the end of the PLC cycle. The use of two different variables will permit to avoid several changes of state during the same PLC cycle. The value of the next state is determined from the current state and from the observation of the input variation. The proposed code is only composed of `IF` statements: the extern `IF` selects the current state (variable `STATE`), while the second level select the changed inputs. The test of the input, to determine the right outputs and the next state, is made simple comparing its value with 0 or 1. The rising and falling input edges, represented by the events in the Mealy machine, doesn't need to be take into account as the information of the previous value of the variable is embedded in the state. The output changes are produced using the functions `SET` and `RESET`. At the end of the cycle the `STATE` value is updated with the one present in calculated `NEXTSTATE` variable.

```
PROGRAM

IF (STATE=0) THEN
   IF (I1) THEN
      SET O1;
      NEXTSTATE:=3;
   ELSIF (I2) THEN
      RESET O2;
      SET O3;
      NEXTSTATE:=7;
   ELSIF (NOT I3) THEN
      NEXTSTATE:=9;
   ELSE
      NEXTSTATE:=STATE;
   END_IF
END_IF;

IF (STATE=3) THEN
   [...]
END_IF;

IF (STATE=7) THEN
   [...]
END_IF;

IF (STATE=9) THEN
   [...]
END_IF;

[...]
STATE:=NEXTSTATE;

END_PROGRAM
```

Figure 7: An extract of the generated PLC code in ST language

3.6 Implementation

To set up the case studies and to generate the PLC control program a computer software framework has been developed within this thesis. The built software is completely integrated with Supremica (http://www.supremica.org), a tool for verification and synthesis of discrete event supervisors, which has been used to perform automata synchronization and supervisor synthesis. Supremica has been developed as part of a research effort at Chalmers University of Technology, Sweden. The developed software, coded in Python, permits to automatically create described generic models and to perform supervisor treatment as presented in this thesis. To increase the framework flexility, the data exchange with Supremica is performed automatically using the XML-RPC protocol. Developed software gives as input to

Supremica a part of Plant model and Specification and retrieves as output the synthesized Supervisor. The prototype uses the obtained Supervisor to automatically generate the Controller and the PLC code in ST language. Python sources and documentation are available at http://www.diee.unica.it/ giua/TESI/Cantarelli/.

Chapter 4

Use Case Study: The car gate

The automatic gate of a car park is often held up as an example for an automated systems. As this system is very small and its behavior is well-known by every body, this system is often used as teaching exercise. In this document, we present all steps to obtain, with the Supervisory Control Theory (SCT), the PLC code to control this equipment.

4.1 Description of the process

4.1.1 System structure

The overall system could be partitioned into two parts: the process to control and its control unit (Fig. 1). The boundary between the two parts is imposed by the technology. From a technical point of view, the process of an automatic gate for a car park is composed of the following elements:

- one gate with 2 limit switches to indicate when the gate is fully open or fully close
- one electrical motor with 2 contactors to control the movement direction (one per direction)
- one sensor to detect the presence of a car in front of the gate
- one receiver to catch radio signal emitted by remote controls of users

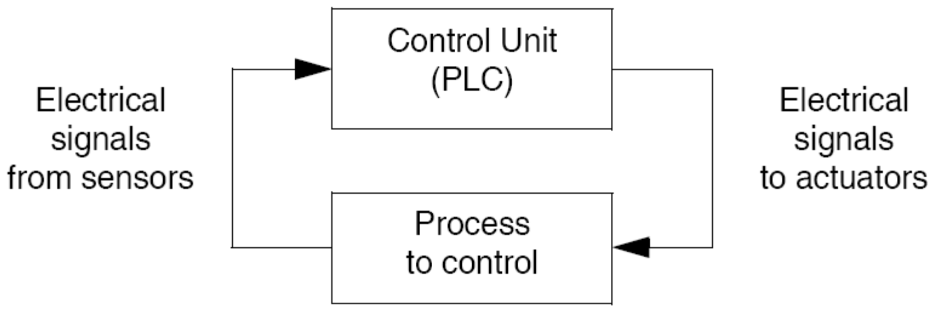


Figure 1: Decomposition of an automated system

The technology used for the process imposes the inputs and outputs of the industrial control unit (Fig. 2). In our case, inputs and outputs have only Boolean values.

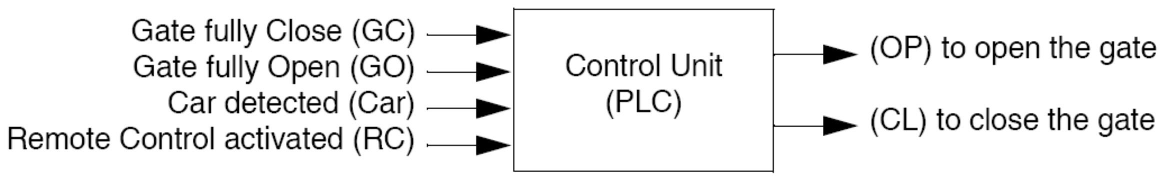


Figure 2: Inputs and outputs of the control unit

4.1.2 Expected behavior

The expected behavior is as follows. Users could open the gate by activating the remote control. When the gate is fully open, the gate is automatically close. When the gate is not fully close, the detection of a car opens the gate.

4.2 SCT: Events definition

The SCT requires the definition of events, a model of the physical process to control and a specification for the closed-loop behavior. As we have a real process to control, events are imposed by the technology and more precisely, by the choice of inputs and outputs of the control unit. To model the behavior of a Boolean variable with events, it is necessary to associate two events to each Boolean: one for the change from 0 to 1 (rising edge) and one for the change from 1 to 0 (falling edge). As the control unit imposes only the values of its outputs, outputs changes are controllable. Inputs changes are uncontrollable. Events used in this case study are presented in table 4.1.

4.3 SCT: Plant Model

The SCT requires a model of the physical process to control, and the quality of that model determines the quality of the generated supervisor and the quality of PLC code obtained from it. If the model of the process is too simple, the real process could generate uncontrollable events for which the occurrence is not expected in the plant model and consequently will not be taken into account into the PLC code. The plant model is obtained by synchronous composition of the following models:

Table 4.1: Used events

Name	**Nature**	**Comment**
GC_1	Uncontrollable	GC Input change from 0 to 1
GC_0	Uncontrollable	GC Input change from 1 to 0
GO_1	Uncontrollable	GO Input change from 0 to 1
GO_0	Uncontrollable	GO Input change from 1 to 0
Car_1	Uncontrollable	Car Input change from 0 to 1
Car_0	Uncontrollable	Car Input change from 1 to 0
RC_1	Uncontrollable	RC Input change from 0 to 1
RC_0	Uncontrollable	RC Input change from 1 to 0
OP_1	Controllable	OP Input change from 0 to 1
OP_0	Controllable	OP Input change from 1 to 0
CL_1	Controllable	CL Input change from 0 to 1
CL_0	Controllable	CL Input change from 1 to 0

- the model of the process (to express what is physically possible by the process)
- the model of interactions between the process and the control unit (to express what is physically possible by the control unit)
- the model of electrical signals exchanged between the process and the control unit (to express what is physically possible by a Boolean value)

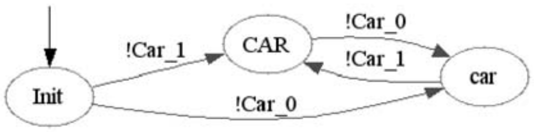
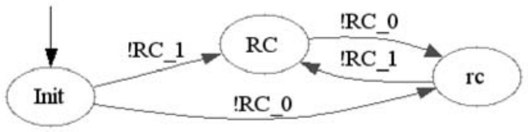
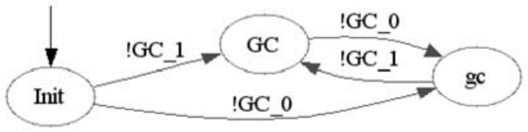
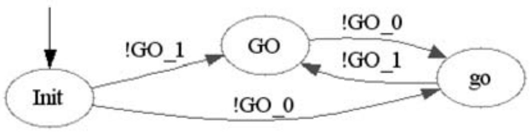
4.3.1 Models of inputs and outputs of the control unit

As two events were associated for each Boolean variable, it is necessary to precise which sequences of events are possible. Theses sequences must respect this following condition: The two events are emitted alternatively. If the initial value of the Boolean variable is known, this behavior could be modelled with a two-state automaton (one by value). If the initial value of the Boolean variable is unknown, this behavior must be modelled with a three-state automaton (one by value and one for the initial state). We propose two distinct models, one for inputs, and one for outputs, to take into account the problem of the initial value. The initial value of CU outputs can without loss of generality be considered to be false. However, the initial value of inputs is unknown (it could be imposed in the process model, if it is necessary.

Models of inputs (4 automata)

As each electrical signal sent by sensors represents a Boolean variable for which the initial value is unknown, the behavior of each input must be modelled with a 3-state automaton. All states are marked.

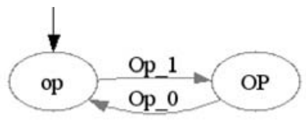
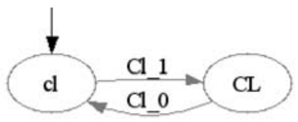
Table 4.2: Models of inputs (4 automata)

Input	**Automaton Model**
`Car`	
`Remote` Control	
`Gate` Closed	
`Gate` Opened	

Models of outputs(2 automata)

As each electrical signal sent to actuators represents a Boolean variable for which the initial value is known, the behavior of each output could be modelled with a 2-state automaton. The initial state of each automaton is "No signal delivered". All states are marked.

Table 4.3: Models of outputs (2 automata)

Output	**Automaton Model**
`Open`	
`Close`	

4.3.2 Model of the process

To convince an engineer to use the SCT to obtain control laws, it is necessary to develop models of the process including all relevant technical characteristics of the process. For this case study, this model is composed of for automata.

Model of the gate

The model given in Fig. 3 presents the different states of the gate. As only two logical sensors are fixed on the gate, the model is composed on the 3 following states:

- **State GC** (initial state): The gate is fully close (sensor `GC` is activated)
- **State GhO** The gate is half-open (no sensor is activated)
- **State GO** The gate is fully open (sensor "GO" is activated)

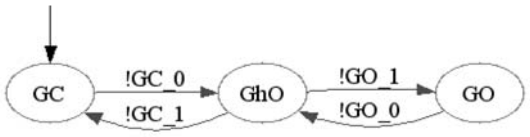

$$\begin{tikzcd} {} \arrow[d] & & \\ GC \arrow[r, "!GC\_0"] & GhO \arrow[l, bend left, "!GC\_1"] \arrow[r, "!GO\_1"] & GO \arrow[l, bend left, "!GO\_0"] \end{tikzcd}$$

Figure 3: Model of the gate

Model of the motor

This model given Fig. 6 presents the different states of an electrical motor. The interaction between the gate model (Fig. 4) and the motor model are expressed by given, in each state of the motor model, which events of the gate model could be occur. This model is composed on the 5 following states:

- State Stop (initial state): The motor is Off (no gate movement is possible).
- State Close_A: Only output `Close` is activated. The motor is On. The selected rotation direction authorizes only the closing of the gate (only events `GO_0` and `GC_1` could be occur).
- State Close_B: Outputs `Close` and `Open` are simultaneously activated. The motor is On. As output `Close` was activated in first, the selected rotation direction authorizes only the closing of the gate (only events `GO_0` and `GC_1` could be occur).
- State Open_A: Only output `Open` is activated. The motor is On. The selected rotation direction authorizes only the opening of the gate (only events `GC_0` and `GO_1` could be occur).
- State Close_B: Outputs "Close" and "Open" are simultaneously activated. The motor is On. As output `Open` was activated in first, the selected rotation direction authorizes only the opening of the gate (only events `GC_0` and `GO_1` could be occur).

All states of this model are marked. All changes of states are controllable by the control unit.

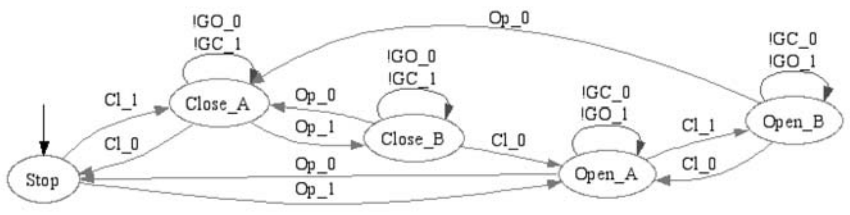
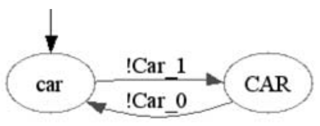
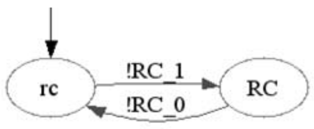


Figure 4: Model of the motor

Model of sensors

For this case study, we have chosen to fix the initial value of sensors "Car" and "RC" to value "False". This choice is modelled with two new automata (one by variable). All states of these automata are marked.

Table 4.4: Models of outputs (2 automata)

Description	**Automaton Model**
`Car Sensor`	
`RC sensor`	

Automaton Car Sensor is included into automaton Car Input. Automaton RC Sensor is included into automaton RC Input RC.

Model of interactions between the process and the control unit

The model presented in Fig.5 expresses the possible behaviour of the control unit. We consider that the control unit is an infinitely reactive system. It can distinguish all input events and can always calculate all the consequences of each event.

The model is composed of 2 states:

- State Wait (initial state): The control unit waits an uncontrollable event to occur.

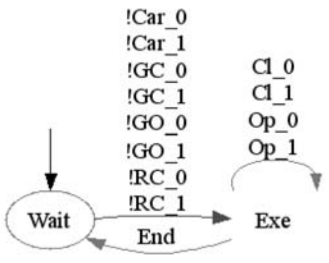


Figure 5: Model of the control unit (PLC Scan)

- State Exe: The control unit executes the user code.

The change from `Wait` to `Exe` depends on an uncontrollable event. In state `Exe`, one or more controllable events could be sent. The change from `Exe` to `Wait` is conditioned by a specific controllable event `End`. This event corresponds to the end of the code execution. Event `End` was introduced to permit the control unit to react to an occurrence of an uncontrollable event by forcing some controllable events to occur. This model is generic. In this model, only state `Wait` is marked.

4.4 SCT: Specification

4.4.1 Specification parts to take into account the technology used

These specification parts are generics.

Spec A: Stop motor at the end of the opening movement

This specification is compulsory to protect the motor. At the end of the user code execution, it is forbidden to have output `Open` activated when input `GO` is true.

The model presented Fig. 6 is composed on the 4 following states:

- State goop (initial state): Input `GO` is false and output `OP` is not activated.
- State goOP: Input `GO` is false and output `OP` is activated.
- State GOop: Input `GO` is true and output `OP` is not activated.

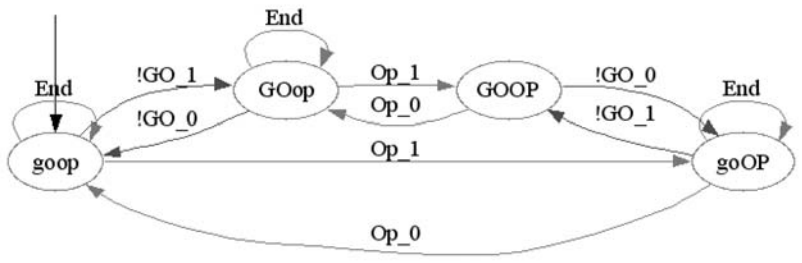


Figure 6: Spec A: Stop motor at the end of the opening movement

- State GOOP: Input `GO` is true and output `OP` is activated.

As controllable event `End` is not authorized from state `GOOP`, it is impossible to finish the PLC scan if this state is true. All States are marked.

Spec B: Stop motor at the end of the closing movement

This specification is compulsory to protect the motor. At the end of the user code execution, it is forbidden to have output `Close` activated when input `GC` is true.

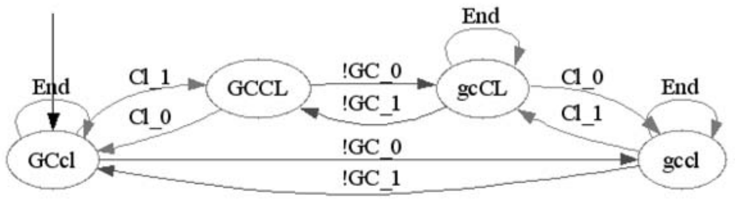


Figure 7: Spec B: Stop motor at the end of the closing movement

The model presented Fig. 7 is composed on the 4 following states:

- State GCcl (initial state): Intput `GC` is true and output `CL` is not activated.
- State GCCL: Input `CL` is true and output `CL` is activated.
- State gcCL: Input `GC` is false and output `CL` is activated.
- State gccl: Input `GC` is false and output `CL` is not activated.

As controllable event `End` is not authorized from state `GCCL`, it is impossible to finish the PLC scan if this state is true. All States are marked.

Spec C: Only one direction activated

This specification is compulsory to protect the motor. At the end of the user code execution, it is forbidden to have simultaneously outputs `Open` and `Close` activated.

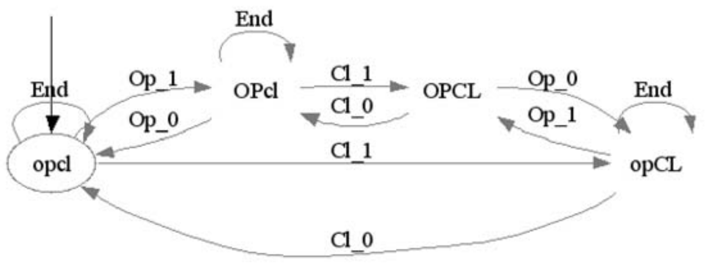


Figure 8: Spec C: Only one direction activated

The model presented Fig. 8 is composed on the 4 following states:

- State opcl (initial state): No output is activated.
- State OPcl: Only output `OP` is activated.
- State opCL: Only output `CL` is activated.
- State OPCL: Outputs `CL` and `OP` are activated.

As controllable event `End` is not authorized from state `CLOP`, it is impossible to finish the PLC scan if this state is true. Only state `opcl` is marked.

4.4.2 Specification parts introduced to obtain the expected behaviour

Spec D: No closing movement when a car is detected

This specification is compulsory to protect the users. It is forbidden to have a closing movement when a car is detected. The closing movement notion is expressed by the changes of input values. The model presented Fig. 9 is composed on the 2 following states:

- State car (initial state): No car detected.

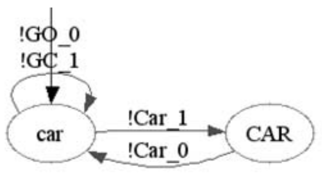


Figure 9: Spec D: No closing movement when a car is detected

- State CAR: A car is detected by the sensor.

Uncontrollable events `GO_0` and `GC_1` are not authorized from state `CAR`. All States are marked.

Spec E: No closing movement when the remote control is activated

This specification is compulsory to protect the users. It is forbidden to have an closing movement when the RC is activated. The closing movement notion is expressed by the changes of input values.

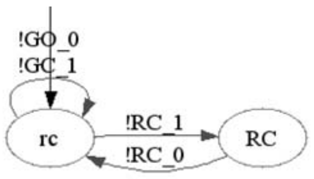


Figure 10: Spec E: No closing movement when the remote control is activated

The model presented Fig. 10 is composed on the 2 following states:

- State rc (initial state): No activation of the remote control.
- State RC: The remote control is activated.

Uncontrollable events `GO_0` and `GC_1` are not authorized from state `CAR`. All States are marked.

Spec F: No opening movement without a previous activation of the remote control

This specification is compulsory to avoid the opening of the gate when the gate is fully close without an activation of the remote control. Only users with a remote control could open the gate.

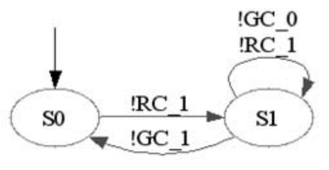


Figure 11: Spec F: No opening movement without a previous activation of the remote control

The model presented Fig. 11 is composed on the 2 following states:

- State S0 (initial state): The gate is close and the remote control is not activated.
- State S1: The remote control has been activated after the last end of closing movement.

Uncontrollable event `GC_0` is not authorized from state `S0`. All States are marked.

Spec G: An activation of the remote control opens the gate

This specification is introduced to obtain the opening of the gate when the remote control is activated. When the gate is fully open, the activation of the remote control has no effect. The model presented Fig. 12 is composed on the 3 following states:

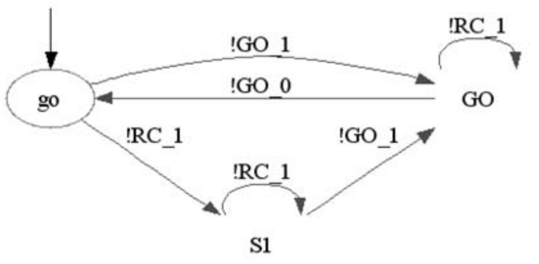


Figure 12: Spec G: An activation of the remote control opens the gate

- State go (initial state): The gate is not open and the remote control is not activated.

- State GO: The gate is fully open.
- State S1: The remote control has been activated and the end of the opening movement is expected.

Only initial state is marked.

Spec H: The detection of a car opens the gate

This specification is introduced to obtain the opening of the gate when a car is detected. When the gate is fully open or fully close, the detection of the car has no effect. The model

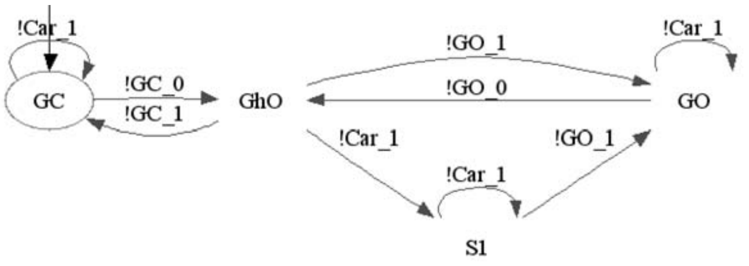


Figure 13: Spec H: The detection of a car opens the gate

presented Fig. 13 is composed on the 4 following states:

- State GC (initial state): The gate is fully close.
- State GhO: The gate is half-open and no car is detected.
- State GO: The gate is fully open.
- State S1: A car has been detected when the gate was half-open. The end of the opening movement is expected.

Only initial state is marked.

4.4.3 Specification parts introduced to obtain the stability of outputs

These specification parts are generics and are introduced to suppress some circuits in the supervisor model. A model is built for each output. This model avoid several changes of the same output during a code execution. Each model is an 2-sate automaton (Fig. 14).

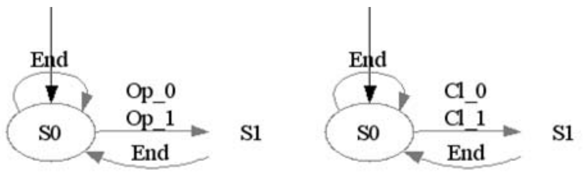


Figure 14: Stability of outputs

- State S0 (initial state): No change for the considered output during the current code execution.
- State S1: The considered output has changed of value during the current code execution.

Only initial State is marked.

4.5 SCT: Supervisor synthesis

For this case study, we have 21 automata with 13 events (8 events are uncontrollable and 5 are controllable). The Plant Model is composed by 11 automata. The Specification is composed by 10 automata. The supervisor proposed by Supremica is composed on 368 states (6 states are marked) and 646 transitions. After a minimization by "Language Equivalence", this supervisor can be describe with an automaton with 110 states (2 states are marked) and 194 transitions. This model is presented on Fig. 4. This automaton is non-blocking and deterministic.

4.6 From Supervisor to Controller

This section presents a strategy to define a deterministic controller from the supervisor obtained with the SCT. Fig. 16 is an extract of the supervisor given Fig. 15. This extract presents all possible behaviors permitted by the model of the PLC (see automaton give Fig. 5).

- Occurrence of an input event with the possibility or not to send an output event, For example: from state q0, when input event RC_1 occurs, the Control Unit has the possibility or not to sent output event Op_1 during the current PLC scan.
- Occurrence of an input event with no possibility to send an output event, For example: from state q0, when input event Car_1 occurs, the Control Unit must finish its PLC scan without sending an output event.
- Occurrence of an input event with the obligation to send an output event. For example: from state q108, when input event GC_1 occurs, the Control Unit must sent output event Cl_0 during the current PLC scan.

In two last cases the behavior of the supervisor is determined, in the first case a choice is present. The supervisor is then non-deterministic by rapport to the control function it should achieve. The controller must be deterministic, meaning that a given input sequence should always produce the same sequence of outputs. To avoid a non-deterministic choice during the code generation, it is necessary to suppress this possibility from the supervisor. A strategy to select the "good" event must to be poposed. The "good" event is the one which drives the system towards the right path. A solution is to complete the specification (with new automata) to obtain after the synthesis step, a supervisor with no choice. It is also possible to select the "good" event according to a specific objective (manual selection or systematic approach). In this case study a strategy which minimizes the number of PLC scan is used. To do this, a weight which represents the minimum number of End events to reach a marked state is assigned to each state. When a choice among more controllable events is present, the transitions which do not provide a gain in terms of step to reach a marked state are suppressed. As this criterion is not sufficient a priority mechanism between output events which favours the functioning of the machine has been added (Op_1 has priority on End, Cl_1 has priority on End, End has priority on Op_0, End has priority on Cl_0). The resulting automaton, composed of 45 states and 70 transitions, is given Fig. 17.

A Mealy machine is the simplest model to express the behaviour of a Control Unit. This behaviour is composed of reactions of a Control Unit to the occurrence of an input event. The Control Unit reacts to the occurrence of an input event by sending of a set of outputs events and changing of current state. This behavior could be obtained from the obtained controller by regrouping all sequences composed of:

- an input event ,
- Output events (the order of events has now no importance because they will sent from the same PLC scan)
- event `End`.

In our case, this operation presents no difficulty because the controller is deterministic. The resulting Mealy machine in Fig. 18 has 15 states and 40 transitions.

4.7 From Mealy machine to PLC code

The obtained Mealy machine is now really suitable to be processed in order to produce the code for the PLC. The implementation is made in conformance with the International Electrotechnical Commission 61131-3 standard using the ST language. The code must represent the evolution from a state to another when an input event occurs. The change of states could be associated with the change of values of one or more logical outputs. As in the Mealy machine only one state is active at the same time, the use of a Boolean variable for each state is not necessary. An integer variable is sufficient to code all states of the Mealy machine. Each value identifies a different state of the Mealy machine. To simplify the initialization step, it is proposed to code the initial state with the zero value. This solution is only feasible if the default initial value of an integer for the PLC used is zero too. The proposed code is only composed of statements IF. The first level is to select the possible reactions from the current state (variable "State"). The second level is to select the reaction of the input change. The changes of outputs are produced using functions SET and RESET. Variable "NewState" represents the state after the evolution. For this case study, the PLC code obtained is presented in table 1. It was tested with success on to control a real electrical motor. Remarks:

- The test of the input, to determinate the right outputs and the next state, is made simple comparing its value with 0 or 1. The upper or lower front test doesn't need to be take into account as the information of the previous value of the variable is embedded in the state.

- Two distinct integers (`State`, `NextState`) are used in the proposed code to avoid to avoid several states changes within the same PLC scan. By using for states statements `IF ELSIF` or `CASE`, it will be possible to avoid several states changes within the same PLC scan with an only integer. Theses solutions were not retained because not accepted by the PLC used for tests.
- The last instruction permits to update variable `STATE`.
- For PLC scans without evolution on the Mealy machine, the value of variables `STATE` and `NEXTSTATE` are the same.

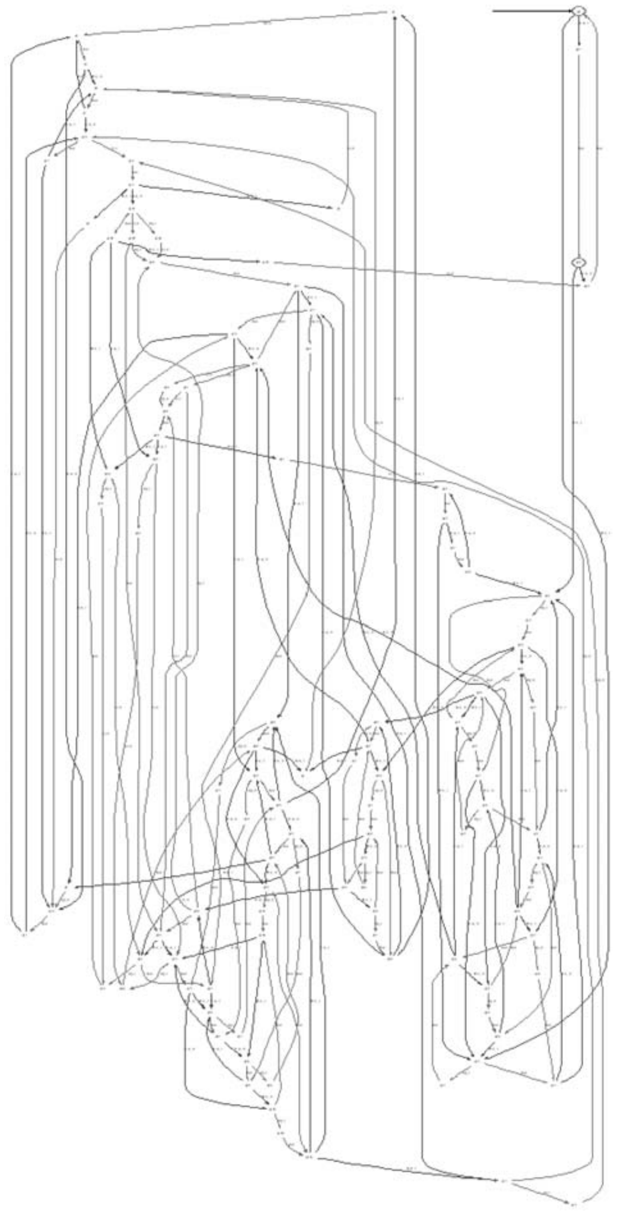


Figure 15: Supervisor after minimization by "Language Equivalence"

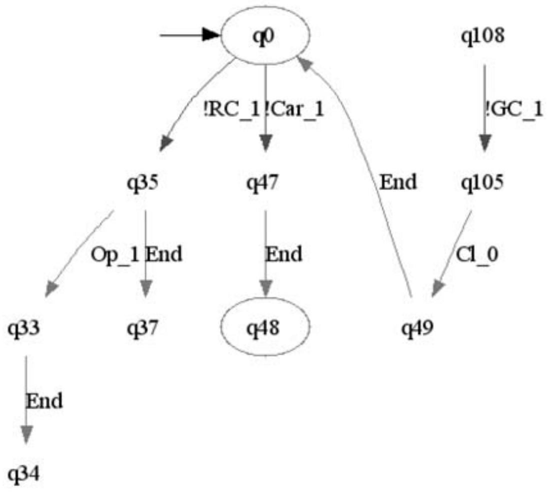


Figure 16: Extract of the supervisor in Figure 15

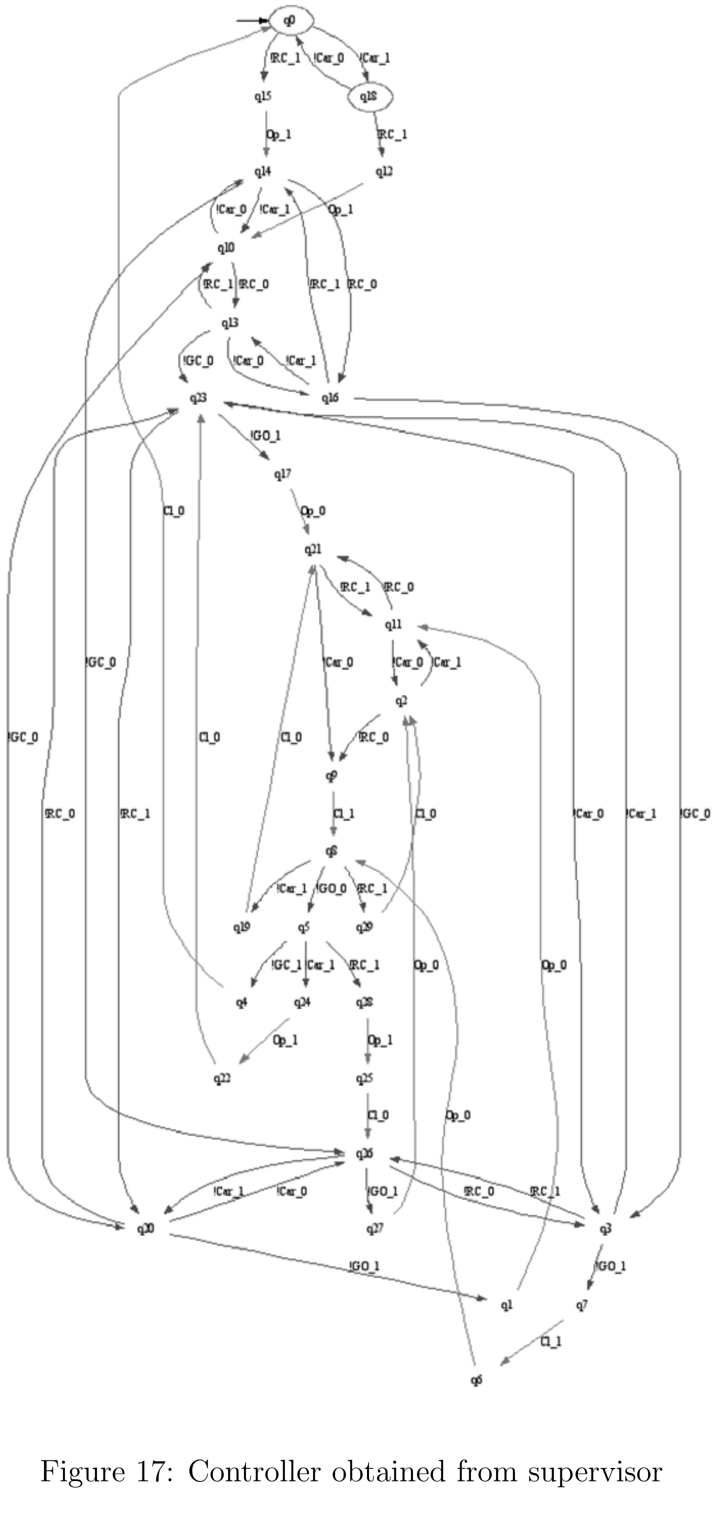


Figure 17: Controller obtained from supervisor

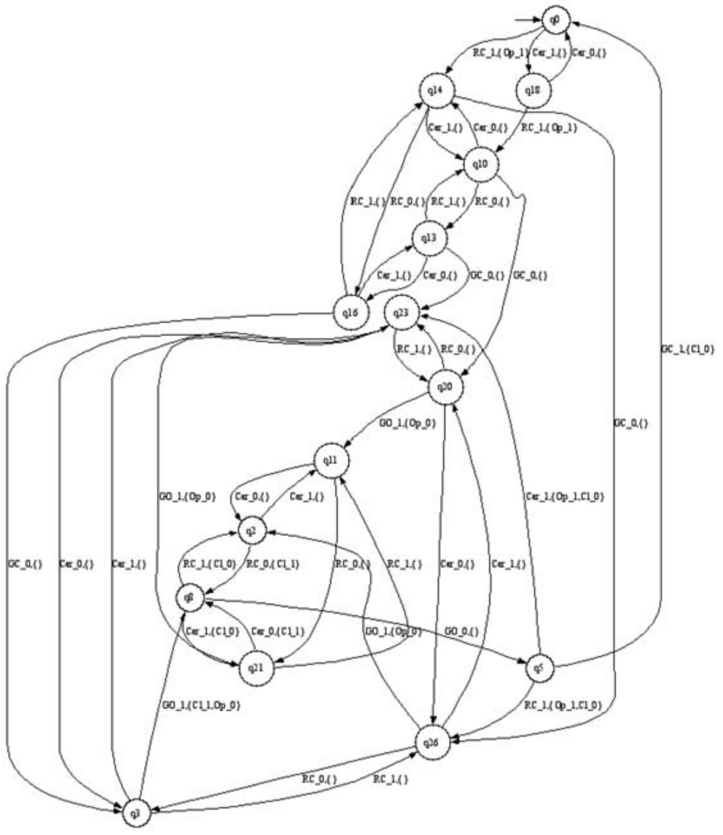


Figure 18: Mealy machine implementation of the controller

Chapter 5

Use Case Study: The pick and place station

This use case deals with the control of a pick and place station. This system has been chosen for its particular sequential nature which can be approached really easily with the SCT framework while it can constitute a difficulty when dealt with other techniques.

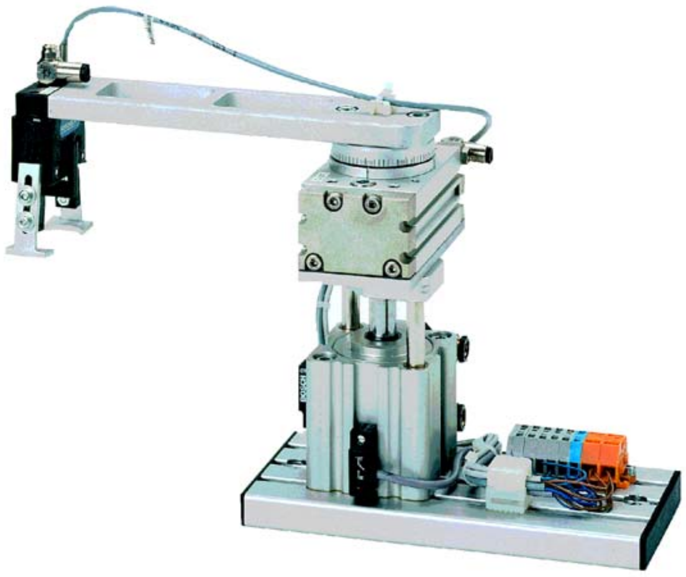


Figure 1: Rotary lift gripper to control

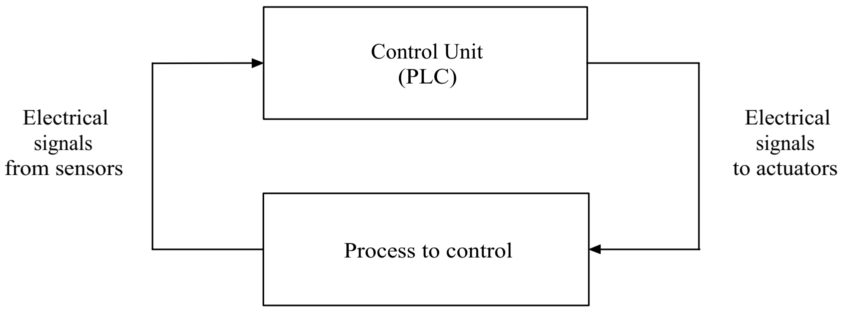
5.1 Description of the process

5.1.1 System structure

The overall subsystem can be partitioned in two parts: the process to control and the control unit (Fig. 2). The boundary between the two parts is imposed by the technology, and more precisely, by the choice of inputs and outputs of the industrial control unit.

The process is composed of three pneumatic components which permit:

- the closing and the opening of the gripper to take and release a piece,
- the descent of the gripper in order to catch up the piece,
- an horizontal displacement in order to move the gripper from the pick station to the place station.



Control Unit (PLC)
Electrical signals from sensors
Electrical signals to actuators
Process to control

Figure 2: Decomposition of an automated system

The pneumatic scheme is presented in Figure 3.

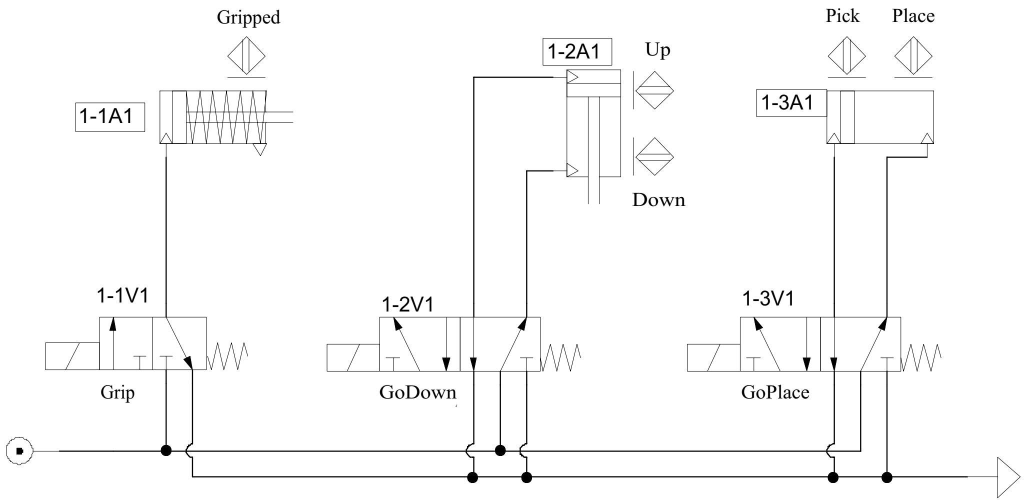


Figure 3: Pneumatic scheme

In the schema it is presented the functional chain of each pneumatic component. The first pneumatic functional chain is composed of:

- One single acting cylinder with one limit sensor to indicate the `Gripped` position
- One monostable valve to control the cylinder (`Grip`)

The second pneumatic functional chain is composed of:

- One double acting cylinder with two limit sensors to indicate when the arm is `Up` or `Down`

- One monostable valve to control the cylinder (`GoDown`)

The third pneumatic functional chain is composed of:

- One double acting cylinder with two limit sensors to indicate when the arm is at the `Pick` or at the `Place` station
- One monostable valve to control the cylinder (`GoPlace`)

The pick and place operation is started by a push button. The technology used for the process imposes the inputs and outputs of the industrial control unit (Fig. 4).

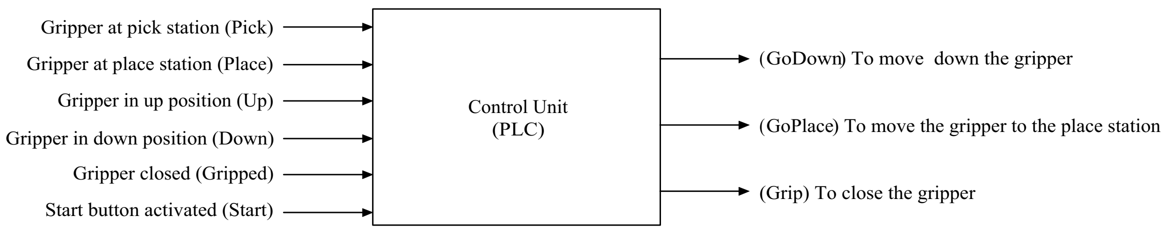

Gripper at pick station (Pick)
Gripper at place station (Place)
Gripper in up position (Up)
Gripper in down position (Down)
Gripper closed (Gripped)
Start button activated (Start)
Control Unit (PLC)
(GoDown) To move down the gripper
(GoPlace) To move the gripper to the place station
(Grip) To close the gripper

Figure 4: Inputs and outputs of the control unit

5.1.2 Expected behaviour

As the Start button is pushed the arm takes a piece, moves to the Place station, depose the piece and return to the Pick station..

5.2 SCT: Events definition

The SCT requires the definition of events, a model of the physical process to control and a specification for the closed-loop behaviour. As this is the control of a real process the events are imposed by the technology and then by the choice of inputs and outputs of the control unit. To model the behaviour of Boolean variable with events, it is necessary to associate two events to each to each variable: one for the change from 0 to 1 (eventname_1 rising edge of the) and one for the change from 1 to 0 (eventname_0 falling edge). For the input models those events are `uncontrollable`, for the output models they are `controllable`.

5.3 SCT: Plant Model

The SCT requires a model of the physical process to control, and the quality of this model determines the quality of the generated supervisor and the quality of the PLC code obtained from it. If the model of the process is too simple, the real process could generate uncontrollable events for which the occurrence is not expected in the plant model and consequently it will no be taken into account in the PLC code. The proposed plant model is obtained by synchronous composition of the following models:

- the model of the process (to express what is physically possible by the process)
- the model of interactions between the process and the control unit (to express what is physically possible by the control unit)
- the model of the electrical signal exchanged between the process and the control unit (to express what is physically possible by a Boolean variable)

5.3.1 Models of inputs and outputs of the control unit

Two different models are utilised for the inputs and the outputs. As for the inputs no assumptions should be done for the initial value of the electric signal the model has three state (one for each value and one for the initial state). Differently the initial value of the outputs can be considered, without loss of generality, false. Thus the model for the outputs has only two state (one for each value). In the used convention for the capital letters states the signals are present (signal value 1), for the minuscule letter states the signals are not present(signal value 0). In the tables 5.1 and 5.2 inputs and outputs models are presented.

5.3.2 Model of the process

As already said it is necessary to develop the models of the process including all relevant technical characteristics. Designing the model it is necessary nevertheless to pay attention to stay in the predictable behaviour of the component. Indeed, studying the model of a pneumatic cylinder for example, the careful engineer would want to take into account the

Table 5.1: Models of inputs (6 automata)

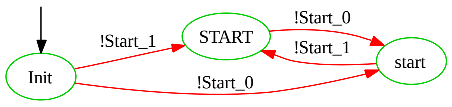
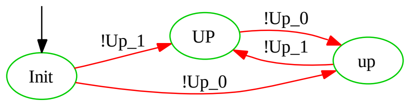
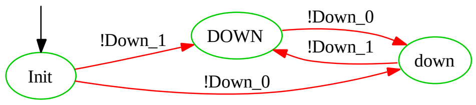
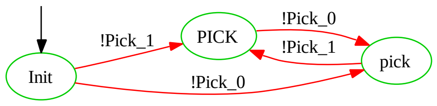
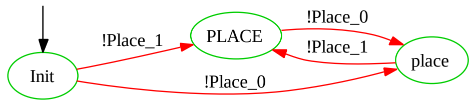
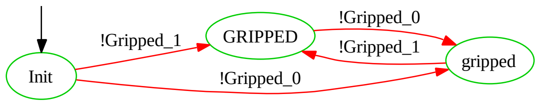
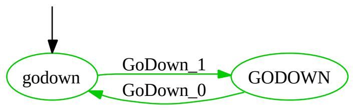
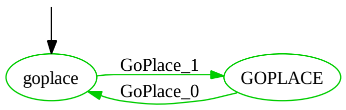
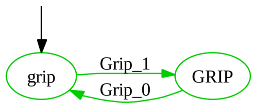
Input	**Automaton Model**
Start	 Init !Start_1 START !Start_0 !Start_1 start !Start_0
Up	
Down	 Init !Down_1 DOWN !Down_0 !Down_1 down !Down_0
Pick	 Init !Pick_1 PICK !Pick_0 !Pick_1 pick !Pick_0
Place	 Init !Place_1 PLACE !Place_0 !Place_1 place !Place_0
Gripped	 Init !Gripped_1 GRIPPED !Gripped_0 !Gripped_1 gripped !Gripped_0

Table 5.2: Models of outputs (3 automata)

Output	**Automaton Model**
GoDown	
GoPlace	
Grip	

nature of the cylindrical chamber. In fact, the coming out of the piston depends on the chamber air pressure. As not differently specified the calculated supervisor thinks that he can always stop the command of the valve which controls the cylinder in order to avoid the coming out of the piston. For the supervisor, if the sensor doesn't states that the piston is completely out from the cylinder, it is always possible to cancel the order putting the output to zero. This is not true in reality as if the valve was commanded to supply the air for enough time (and then there is enough pressure in the chamber) the supervisor cannot nevermore avoid the piston's coming out also if he stops the valve. There is an unpredictable behaviour of the component. To manage this aspect it would be needed to take in account the uncertain nature of some events (or the time). This is not possible as the SCT doesn't provide a framework for stochastic or timed automata. Finally the careful engineer who would like to use the SCT theory to control a process will then be in need to add more specifications in order to instruct the supervisor on aspects that the supervisor cannot discover by himself simply reading the plant models (as they are not, once again, enough detailed). Nevertheless the engineer should still design the plant models with all relevant details. He will just need to pay attention to stay in the predictable domain.

Model of Cylinders

The models in Figures 5 and 6 present the two double effect cylinders used to move the mechanical arm. The `Vertical Cylinder` is the one which permits to reach the `Up` and `Down` positions. The `Horizontal Cylinder` permits the arm rotation in order to reach the `Pick` and `Place` stations. As these two cylinders have two logical sensors the models have three state, one for each sensor plus a state where no sensor is activated. The Figure 7 presents the `Gripper Cylinder` which permits the closing of the gripper. As this last is a simple effect cylinder with one logical sensor the model has just two state (`Gripped` or `Ungripped`). For all the three models the initial state has been chosen in according to the initial desired state of the arm: `Pick` station, `Up` position, `Ungripped`. All the states of these models are marked. All the state changes are moreover observable by the control unit.

Model of Valves

Each cylinder has a valve which controls its movements. The Figures 8, 9 and 10 presents the three models. The valves which control the cylinders are monostable, the designed models

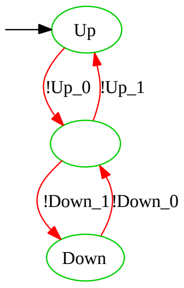


Figure 5: Double effect vertical cylinder

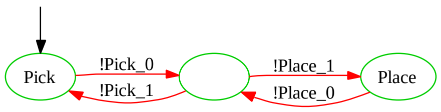


Figure 6: Double effect horizontal cylinder

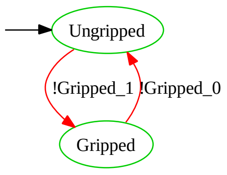


Figure 7: Simple effect gripper cylinder

are then composed by two states. One state represents the valve commanded in order to let the piston out from the cylinder (`GoDown`, `GoPlace`, `Grip`). The other state represents the valve commanded in order to let the piston in. The interactions between the cylinders and the valves are expressed giving for each state of the valves the events of the cylinders which can occur. In the vertical cylinder valve, for example, it is possible for the sensor `Down` to be activated (`Down_1` event) only if the valve state is `GoDown`, which means that the air is entering the cylinder in order to let the piston out. As the vertical and horizontal cylinders have double effect the states in the corresponding valves which permit to reach the two positions (`Up` and `Down` for the vertical cylinder and `Pick` and `Place` for the horizontal cylinder) are both marked.

As the cylinder of the gripper has simple effect the state of its valve which bring the cylinder into the unstable state (`Gripped`) is not marked. For all the models all the state changes are observable by the control unit.

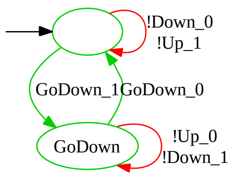


Figure 8: Monostable valve for the vertical cylinder

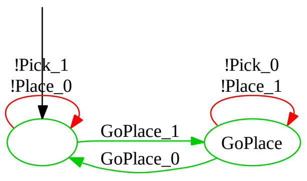


Figure 9: Monostable valve for the horizontal cylinder

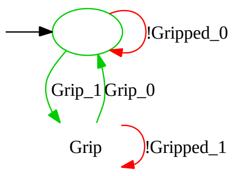


Figure 10: Monostable valve for the gripper cylinder

Model of the Button

To start the pick and place operation is present a monostable button which interacts with the `Start` input. The model is presented in the Figure 11. It is immediate to see that this model is not so far from the input model presented in the Table 5.1; the only difference is that in the model button we know the initial state. It is important to understand the need of these two different models. If the input model was suppressed nothing will have changed on the calculated supervisor. This is the reason why when peoples approaches the SCT to control a system the input models are never presented. But that model is essential to separate the different parts of the system, to provide the systematic approach we are looking for. The electric signal which is "behind" the button have no constraints on its initial state. It is the technology of the used button which states that in its initial position it is released.

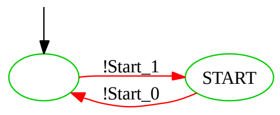


Figure 11: Monostable button to start the operation

5.3.3 Model of interactions between the process and the control unit

The model presented in the Figure 12 expresses the possible behavior of the control unit which is considered as an infinitely reactive system. It can distinguish all the inputs events and can always calculate the consequences for each event. The model is composed of two states. In the `Wait` state the control unit waits for an uncontrollable event to occur, in the `Exe` state the control unit executes the user code. The change from the `Wait` state to the `Exe` state depends on an uncontrollable event. In the `Exe` state one or more controllable events can occur. The `End` event corresponds to the end of the code execution and permits the state change from `Exe` to `Wait`. This event has been introduced to permit to the control unit to react to an occurrence of an uncontrollable event by forcing some controllable event to occur. The `Wait` state is the only marked one. The presented model is generic.

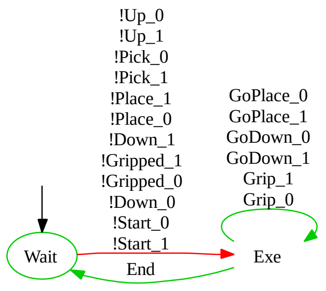


Figure 12: Model of the control unit (PLC scan)

5.4 SCT: Specifications

5.4.1 Specifications to obtain the expected behavior

The Plant Model presented above describe what the different components of the system can do. In order to obtain the expected behavior it is needed to add some specifications which instruct the supervisor on what it can permits. While the plant models can be considered generic for every system (if the models are correct and properly detailed) these specifications are specifically designed for the considered system. The specifications are moreover expressed in terms of uncontrollable events. In fact this means that the system is instructed on what he has to do, not on how to do it. The efforts done in properly designing the Plant Model will permit to the supervisor to discover himself the way to make things up.

Spec A: The activation of the button produces the desired sequence

This specification is introduced to obtain the right sequence of movements when the `Start` button is pressed. As said above in the expected behavior after the activation of the button the arm have to take a piece at the pick station, bring it at the place station, release the piece and then come back to the pick station. The model which describes this sequence is presented in the figure 13. In this specification there are no constraints on the vertical position, so the `Up_` and `Down_` events can be produced in every state once the `Start` button has been activated. The start button can moreover be activated in every state as there are no constraints on this point.

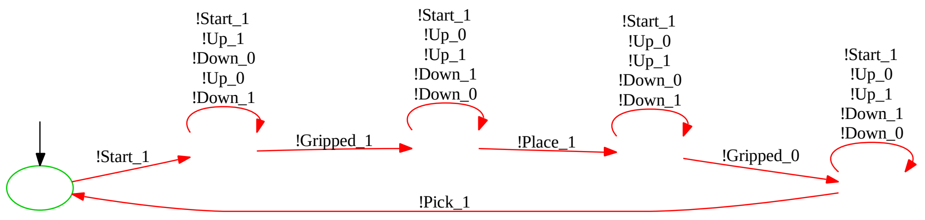


Figure 13: Spec A: The desired sequence

Spec B: The gripper opening and closing are avoided in undesired situations

This specification is compulsory to ensure the right transport of the piece. The `Gripped_` events are possible only if we are in the `Down` position. Furthermore the `Gripped_1` event is possible only at the pick station (where it is needed to take the piece) and the `Gripped_0` event is possible only at the place station (where it is needed to release the piece). These specification avoid the release of the piece during the transport as well. Figures 14 and 15 present the two models.

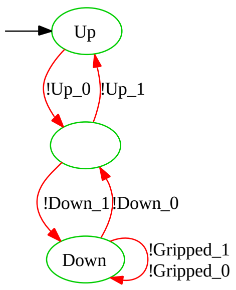


Figure 14: Spec B1: The gripper opening and closing are avoided in `Up` position

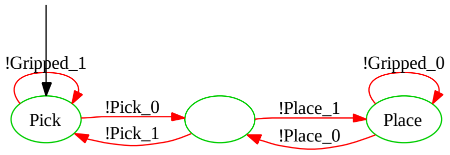


Figure 15: Spec B2: The gripper opening and closing are avoided in undesired situations

Spec C: No pick and place movements if not in up position

This specification is compulsory to protect the user and the environment. The rotatory movement to displace the arm among the pick and the place stations is possible only when in `Up` position. The rotatory movement notion is expressed by the changes of the input values. The Figure 16 presents the two models.

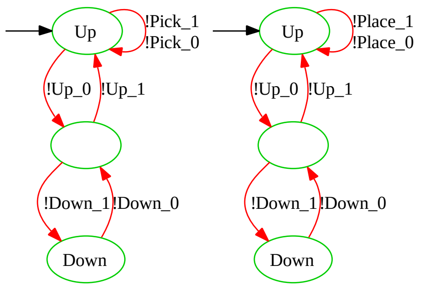


Figure 16: Spec C: No pick and place movements if not in up position

5.4.2 Non determinism avoidance specification

Spec D: The arm have to raise after having taken the piece

This specification is compulsory to guarantee the correct sequence of operations and it has been added to solve a non deterministic behaviour of the controller. This automaton could have been replaced with a priority strategy on outputs events as done with the Gate case study. In this case the solution to the non determinism is embedded in the specifications and the supervisor discovers it during the synthesize operation while in the priority strategy it has obtained with a *post-production* algorithm. The model is presented in Figure 17.

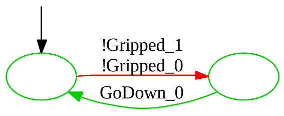


Figure 17: Spec D: The arm have to raise after having taken the piece

5.4.3 Specifications to obtain the stability of the outputs

These specifications are generic and are introduced to suppress some circuits in the supervisor model. A model is built for each output. This model avoids several changes of the same output during a code execution. The models are presented in the Figure 18.

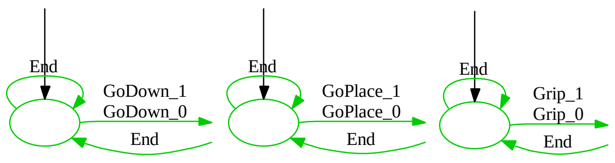


Figure 18: Stability of outputs

5.5 SCT: Supervisor synthesis

In this case study 27 automata with 19 events have been presented. The Plant Model is composed by 15 automata. The Specification is composed by 12 automata. The supervisor proposed by Supremica[1] is composed of 173 states (3 states are marked) and 251 transitions. After a minimization by language equivalence, this supervisor can be described by an automaton with 108 states (2 states are marked) and 160 transitions. The automaton, presented in Figure 19, is non-blocking and deterministic.

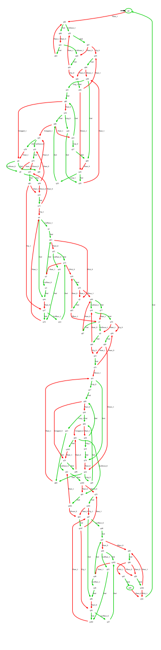


Figure 19: The supervisor after the minimization

[1]Supremica is a tool for verification and synthesis of discrete event supervisors developed as part of a research effort at Chalmers University of Technology, Sweden

5.6 SCT: From Supervisor to Controller

This section presents a strategy to define a deterministic controller from the supervisor obtained with the SCT. The Figure 20 is an extract of the supervisor given in Figure 19.

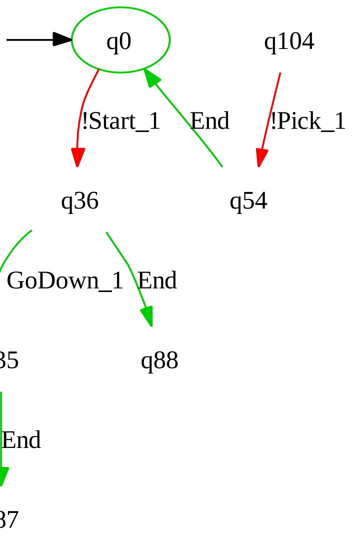


Figure 20: Extract of the supervisor presented in Figure 19

This extract presents some behaviors permitted by the model of the PLC (see automaton Figure 12):

1. Occurrence of an input event with the possibility or not to send an output event. For example: from state `q0`, when input event `Start_1` occurs, the Control Unit has the possibility or not to sent output event `GoDown_1` during the current PLC scan.

2. Occurrence of an input event with no possibility to send an output event. scan without sending an output event.

In the last case the behavior of the supervisor is determined, in the first case a choice is present. The supervisor is then non-deterministic by rapport to the control function it should achieve. The controller must be deterministic, meaning that a given input sequence should always produce the same sequence of outputs. To avoid a non-deterministic choice during the code generation, it is necessary to suppress this possibility from the supervisor. A strategy to select the *good* event must to be proposed. The *good* event is the one which drives the system towards the right path.

A solution is to complete the specification (with new automata) to obtain after the synthesis step, a supervisor with no choice (as partially done with the automaton in
For example: from state `q104`, when input event `Pick_1` occurs, the Control Unit must finish its PLCFigure 17). It is also possible to select the *good* event according to a specific objective (manual selection or systematic approach). In this case study a strategy which minimizes the number of PLC scan is used. To do this, a weight which represents the minimum number of `End` events to reach a marked state is assigned to each state. When a choice among more controllable events is present, the transitions which do not provide a gain in terms of step to reach a marked state are suppressed. The resulting automaton, presented in Figure 21 is composed of 45 states and 73 transitions. The presented strategy, already tested on the Gate use case, has been proven again useful. Nonetheless its test among other different cases is still needed to verify its generality.

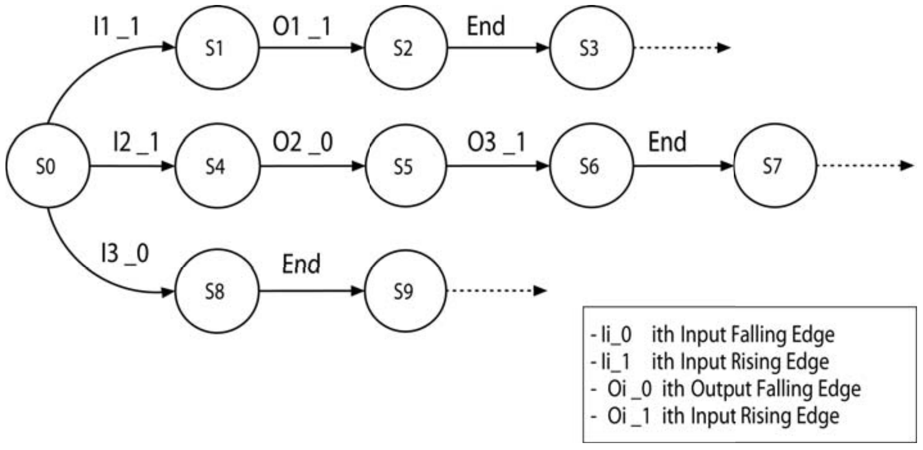

I1 _1
O1 _1
End
I2 _1
O2 _0
O3 _1
End
I3 _0
End
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
- Ii_0 ith Input Falling Edge
- Ii_1 ith Input Rising Edge
- Oi _0 ith Output Falling Edge
- Oi _1 ith Input Rising Edge

Figure 21: The Controller obtained from the Supervisor

5.7 From controller to Mealy machine

A Mealy machine is perhaps the simplest model to express the behavior of a Control Unit. This behavior is composed of reactions of a Control Unit to the occurrence of an input event. The Control Unit reacts to the occurrence of an input event by sending of a set of outputs events and changing of current state. This behavior could be obtained from the obtained controller by regrouping all sequences composed of:

- An Input event ,
- `N` Output events (the order of events has now no importance because they will sent from the same PLC scan) with `N`≥`0`,
- `End` event.

In our case, this operation presents no difficulty as the controller is yet deterministic. The resulting Mealy machine in Figure 22 has 30 states and 58 transitions.

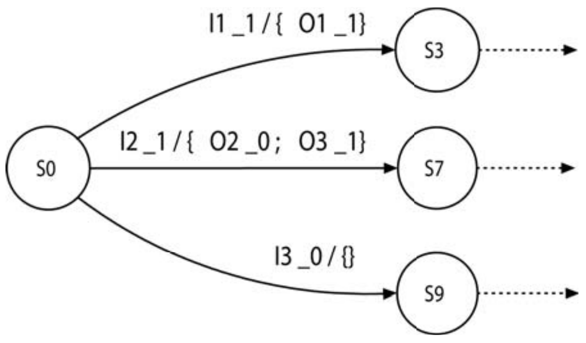


Figure 22: The Mealy machine obtained from the Controller

5.8 From Mealy machine to PLC code

The obtained Mealy machine is now really suitable to be processed in order to produce the code for the PLC. The implementation is made in conformance with the International Electrotechnical Commission 61131-3 standard using the ST language. The code must represent the evolution from a state to another when an input event occurs. The change of states could be associated with the change of values of one or more logical outputs. As in the Mealy machine only one state is active at the same time, the use of a Boolean variable for each state is not necessary. An integer variable is sufficient to code all states of the Mealy machine. Each value identifies a different state of the Mealy machine. To simplify the initialization step, it is proposed to code the initial state with the zero value. This solution is only feasible if the default initial value of an integer for the PLC used is zero too. The proposed code is only composed of statements `IF`. The first level is to select the possible reactions from the current state (variable `State`). The second level is to select the reaction of the input change. The changes of outputs are produced using functions `SET` and `RESET`. Variable `NewState` represents the state after the evolution. For this case study, the PLC code obtained is presented in table 1. It was tested with success on to control a real electrical motor.

Remarks:

- The test of the input, to determinate the right outputs and the next state, is made simple comparing its value with 0 or 1. The upper or lower front test doesn't need to be take into account as the information of the previous value of the variable is embedded in the state.
- Two distinct integers (`State`, `NextState`) are used in the proposed code to avoid several states changes within the same PLC scan. By using in the states the statements `IF ELSIF` or `CASE`, it will be possible to avoid several states changes within the same PLC scan with an only integer. These solutions were not retained because not accepted by the PLC used for tests.
- The last instruction permits to update the variable `State`.
- For PLC scans without evolution on the Mealy machine, the value of variables `State` and `NextState` are the same.

```
PROGRAM

IF (STATE=0) THEN
        IF (Start) THEN
                SET GoDown;
                NEXTSTATE:=25;
        END_IF;
END_IF;

IF (STATE=25) THEN
        IF (NOT Up) THEN
                NEXTSTATE:=28;
        ELSIF (NOT Start) THEN
                NEXTSTATE:=30;
        END_IF;
END_IF;

[...]
[...]
[...]

STATE:=NEXTSTATE;

END_PROGRAM

[EOF]
```

Figure 23: An extract of the generated PLC code in ST language

Chapter 6

Concluding remarks

In this thesis several aspects of discrete event control are discussed. The direction of the research has been motivated by control and design of industrially relevant equipment. The results, however, are generic in nature. They hold for a much more general class of discrete event control problems. In the first part of this thesis programmable logic controllers and the supervisory control theory are presented. The supervisory control problem is to find a supervisor such that the controlled system can replace the specification. The main problem in using the SCT as framework for the design of the control program is the non deterministic nature of the supervisor in relation with the controller function that is required. The key problem is nondeterminism. Literally nondeterminism means: not fully determined. Nondeterminism represents uncertainty. A main concept in this first part of the thesis is that a supervisor requires additional processing to obtain a deterministic controller. In particular it is required that the controller for each input sequence (uncontrollable events) produces always the same output (controllable events). In Chapter 3 a systematic approach is presented to rightly exploit SCT to produce a suitable supervisor. Generic models to represent the controller unit and its inputs and outputs are presented as well as process specific models. For the control unit, a SIMO model which allows the reading of a single input and the producing of multiple outputs has been presented. A MIMO model has been introduced as well in order to take into account more generic situations when the reading of multiple inputs is compulsory in order to produce the right output sequence. The distinction about what it should be included while designing the process model and the specification is presented. If the process models don't describe all possible behavior of the equipment, the supervisor could never prevent the process to reach dangerous situations as these are not completely described. The role to interdict unsafe and unwanted behaviour stands up totally on the specifications. The obtained supervisor is treated to obtain a deterministic controller in the form of a Mealy machine. In literature the final stage when the code is obtained is often muddled with the stage in which determinism is extracted, so that normally a great confusion arises and no clear approach is provided. In this work the two stages are clearly distinct in order to face all the difficulties which are present while extracting deterministic behaviour from the supervisor's maximal permissive behaviour. An algorithm to obtain from the Mealy machine a control program in language ST is presented. In Chapter 4 and 5 two relevant case studies are presented. Two examples show an application of the presented approach. As the proposed approach requires high detailed models the result is a supervisor with a high number of states. State explosion is known as a relevant problem while dealing with supervisory control theory and in this approach this has been confirmed. Anyway, in spite of the huge size of the supervisor constructed following the standard SCT approach, when we pass to the subsequent steps and determinize it, the controller in the form of a Mealy machine has a much smaller number of states. In both case studies the Mealy machine has a remarkably small number of states, showing that this approach is useful in industrial applications. This work leaves open questions on the influence of different control unit models on process models which needs to be provided and so on the obtained supervisor. As the proposed approach is quite modular a study about the generality of the process models would certainly lead to interesting and useful results to build a model database which is suitable to be used in this methodology.

Bibliography

[1] 61131-3: Programmable logic controllers, part 3: Programming languages.

[2] K. Akesson and M. Fabian. Implementing supervisory control for chemical batch processes. *Control Applications, 1999. Proceedings of the 1999 IEEE International Conference on*, 2, 1999.

[3] K. Akesson, M. Fabian, H. Flordal, and A. Vahidi. Supremica: a tool for verification and synthesis of discrete event supervisors. *Proc. of the 11th Mediterranean Conference on Control and Automation*, 2003.

[4] S. Balemi. *Control of Discrete Event Systems: Theory and Application.* PhD thesis, PhD thesis, Swiss Federal Institute of Technology Zurich, 1992.

[5] S. Balemi, GJ Hoffmann, P. Gyugyi, H. Wong-Toi, and GF Franklin. Supervisory control of a rapid thermal multiprocessor. *Automatic Control, IEEE Transactions on*, 38(7):1040–1059, 1993.

[6] BA Brandin. The real-time supervisory control of an experimental manufacturingcell. *Robotics and Automation, IEEE Transactions on*, 12(1):1–14, 1996.

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

[8] MH de Queiroz and JER Cury. Synthesis and implementation of local modular supervisory control for a manufacturing cell. *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, pages 377–382, 2002.

[9] P. Dietrich, R. Malik, WM Wonham, and BA Brandin. Implementation considerations in supervisory control. *Synthesis and Control of Discrete Event Systems*, 185201, 2002.

[10] M. Fabian and A. Hellgren. PLC-based Implementation of Supervisory Control for Discrete Event Systems.

[11] D. Gouyon, J.F. Petin, and A. Gouin. Pragmatic approach for modular control synthesis and implementation. *International Journal of Production Research*, 42(14):2839–2858, 2004.

[12] M.R. Lucas and D.M. Tilbury. A study of current logic design practices in the automotive manufacturing industry. *International Journal of Human-Computer Studies*, 59(5):725–753, 2003.

[13] P. Malik. Generating Controllers from Discrete-Event Models. *Proceedings of MOVEP'2002*, 2002.

[14] SR Mohanty, V. Chandra, and R. Kumar. A computer implementable algorithm for the synthesis of an optimalcontroller for acyclic discrete event processes. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 1, 1999.

[15] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[16] J.-M. Roussel and A. Giua. Designing dependable logic controllers using the supervisory control theory. *16th IFAC World Congress (Prague, Czech Republic)*, 2005.

[17] J. Zaytoon and V. Carre-Menetrier. Synthesis of control implementation for discrete manufacturing systems. *International Journal of Production Research*, 39(2):329–345, 2001.