Chapter 2

Background

This chapter will discuss the main topics needed to understand this work, from discrete event systems to discrete control implementation on Programmable Logic Controllers (PLCs), a more detailed explanation of each topic can be found on the respective cited work.

2.1 Systems

A System as defined by the Cambridge's dictionary is "a set of connected things or devices that operate together". As seen two basic properties of systems are :

- they are formed by grouping smaller parts
- the smaller parts when grouped work together to carry out a specific function

As its definition is so abstract almost anything can be defined as a system, physical or not, beings can be defined as systems and even economic mechanisms can also be considered as systems.

Usually systems are modelled by a Input/Output process. The system is fed with a set of inputs, it process the inputs resulting on the output set, as we can see in Figure 5.3.

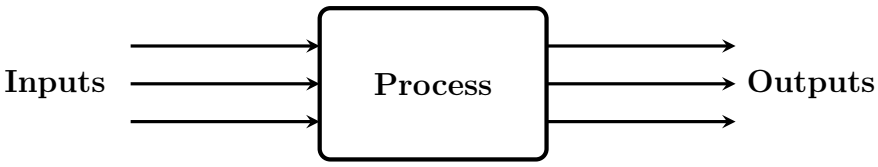


Figure 2.1: Input/Output Process model

In some systems, its inputs and outputs can't represent it's behaviour, so the concept of state is created, and it represents the behaviour of the system in a given instant t.

The states can be continuous or discrete, and the systems which these states represent can be considered as Continuous Systems, Discrete Systems or even Hybrid Systems, which combine both kind of states.

The systems modelled in this work are Discrete Systems, more details about other kinds of systems as well as examples and their analysis can be found on OPPENHEIM *et al.* (1996) and KALOUPTSIDIS (1997).

2.2 Discrete Event Systems

Discrete Systems can be driven by time and by events. It means, the states can be changed continuously by the time or instantaneously by some ensemble of events.

In this thesis we are interested in the event-driven type. Some basic mathematical formalisms, nomenclature and representations can be developed to facilitate the understanding. Some of those will be presented in the following sections based on CASSANDRAS and LAFORTUNE (2009); DAVID and ALLA (1989, 2005).

2.3 Languages

A language can be defined by the Merriam-Webster's dictionary as "a systematic means of communicating ideas or feelings by the use of conventionalized signs, sounds, gestures, or marks having understood meanings" And as it is defined by this dictionary entry we pursue to communicate the complete behaviour of the Discrete Event System (DES). Firstly we need to define a group, or set of marks to characterise the singular behaviour of the system. So, we define a set Σ. This set contains all elements which combined can create a language. Again in analogy with linguistics, each one of these marks, the events can be compared to letters , provided that Σ can be called an "alphabet", and the combination of its events "words". Words are also called "strings " or even "traces". Considering the use of the word "string" as the variable type used on several programming languages used in this work, we prefer the use of the vocables "word" and "trace". We can also define a mark to represent an empty word, ϵ, that is, a word that is not formed by any event.

The combination process to form words is called concatenation. For instance, given two events a and b, the words ab and ba can be created concatenating these two events

and there is no particular reason to suppose that ab is equal to ba, the same way the words "ten" and "net" have different meanings in English.

We can also concatenate two words, to create a different one, we can take the words ab and ba and create words like $abba$ and $baab$.

As we extended the definition of concatenation to words, we define ϵ, the empty word, as the identity element of concatenation: $w\epsilon = \epsilon w = w$ for any word w.

Likewise, we can define the length of a word as the number of events contained by this word, we denote the length with two vertical bars, given a word w its length is equal to $|w|$ and by definition $|\epsilon| = 0$.

As we know, there is a great number of human western languages, as portuguese, english, french, spanish etc, that roughly are formed by the same alphabet, but overall they are formed by different combination of words. Similar things can happen with languages that define the DESs, so we can define as in CASSANDRAS and LAFORTUNE (2009).

Definition 2.1 (Language)
A Language defined over an alphabet Σ is formed from finite-length words generated from the concatenation of the events in Σ and ϵ.

Take for example an alphabet $\Sigma = \{a, b, g\}$, we can define different languages

$$L_1 = \{\epsilon, a, abb\}$$

$$L_2 = \{\text{all possible words of length 3 starting with g}\}$$

$$L_3 = \{\text{all possible words starting with g}\}$$

The cardinality of this sets are $|L_1| = 3$, $|L_2| = 9$, $|L_3| = \infty$ As we can see from the same alphabet very different languages can be created, thus we can define a way to encapsulate all possible languages generated from the same alphabet Σ. Let us denote by Σ^* the set containing all finite words composed with the elements of Σ and ϵ. The * operation is called the *Kleene-closure*. Similarly to L_3 it is countably infinite since it contains arbitrarily long words. For instance the *Kleene-closure* of the alphabet $\Sigma = \{a, b, c\}$ is:

$$\Sigma^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, \dots\}$$

There are a few operations with languages and alphabets that can be defined, but they are outside the scope of this work, they can be found on CASSANDRAS and LAFORTUNE (2009).

2.4 Representation of Languages

Although languages can describe the behaviour of DESs, there are cases, as the one shown by the language L_3 in the last section, in which the language is enormous, in that case countably infinite, what makes them not so simple to communicate the behaviour of the system. For this purpose, there are some other formalisms that aid the comprehension, since they can be a more compact way of expressing the system's behaviour or accompanied by diagrams.

In the following subsections two of the most known representations will be presented: Automata and Petri Nets.

2.4.1 Automata

One of the most known representation of languages are automata. The notion of automaton is basically the definition of DESs, as we saw in the section 2.2: a set of events can change the state of the system. If we know all the events composing the language of the system and its states, we can have its alphabet Σ and we can create a set X composed by all states. From Σ and X we can derive a function that represents the transition from a state to other, this function is called *transition function* of the automaton denoted as $f : X \times \Sigma \to X$. For example if a system have an alphabet $\Sigma = \{a, b\}$ and 2 states, we can name the states x and y, and then create the set $X = \{x, y\}$. Knowing that the system begins at state x and that when event a happens it changes to state z we can create a function $f(x, a)$ and define it as y. Likewise if we know that when the system is at state y and event b happens, a function $f(y, b)$ can be defined as a.

As a visual aid, a representation of these functions can be made through a diagram, called *state transition diagram*. In this kind of diagram the states are represented by circles labeled with their names, and the functions as arcs labeled with the corresponding event,connecting two states, with arrows in one of their extremities indicating the transition from a state to other. The initial state of the automaton has an arc pointing towards it coming from no other state. Figure 2.2 can represent the functions $f(x, a)$ and $f(y, b)$ described in the last paragraph.

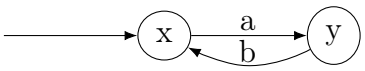


Figure 2.2: State Transition Diagram

Now, for a more complex example, from CASSANDRAS and LAFORTUNE (2009):

Example 2.1 (Simple Automaton)
Given $\Sigma = \{a, b, g\}$, $X = \{x, y, z\}$ and the following transition functions:

$$f(x, a) = x \qquad f(x, g) = z$$
$$f(y, a) = x \qquad f(y, b) = y$$
$$f(z, b) = z \qquad f(z, a) = f(z, g) = y$$

We can represent this automaton with the diagram on Figure 2.3

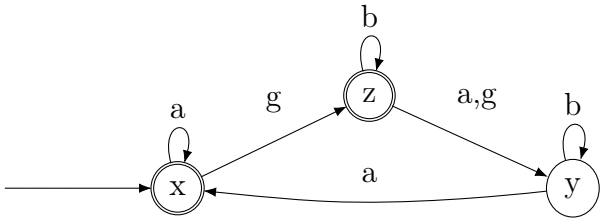


Figure 2.3: Diagram representing the automaton from example 2.1

We can also mark states that have some special meaning, a final state for instance. In this work, as in CASSANDRAS and LAFORTUNE (2009) they are going to be identified by double circles.

Now a deterministic Automaton can be defined:

Definition 2.2 (Deterministic Automaton)
A Deterministic Automaton, denoted by G, is a five-tuple

$$G = (X, \Sigma, f, x_0, X_m)$$

where:

X *is the set of* ***states***

Σ *is the finite set of* ***events*** *associated with* G

f: $X \times \Sigma \rightarrow X$ *is the* ***transition function***

x_0 *is the* ***initial state***

$X_m \subseteq X$ *is the set of* ***marked states***

Other kinds of automata and operations between automata exist but are not going to be used in this work, again CASSANDRAS and LAFORTUNE (2009) present them.

2.4.2 Petri Nets

Another kind of representation of languages are Petri Nets, whose concept was created by C.A.Petri in the early 1960's. Differently from the automata representation that are basically formed from states, Petri nets are bipartite graphs, formed by nodes called *places* and *transitions*. Transitions represent the events that drive the system, and places represent the conditions for these events to happen. The mechanism to represent the fulfilment of the conditions is named marking. A petri net is built over three basic concepts, the petri net graph/structure, its marking and firing transitions. The next subsections will be based on DAVID and ALLA (2005) and CASSANDRAS and LAFORTUNE (2009).

Petri Net Graph

Similarly, arcs are used to connect the nodes and have arrowheads to identify the direction, but differently, all arcs must have exclusively one node at each end, that means no arc is used to identify the initial state of a petri net. As said, a petri net is bipartite graph, that means places can only connect to transitions and vice versa. In this work as in DAVID and ALLA (2005) places will be represented by circles and transitions by bars.

(a) A place. (b) A transition

Figure 2.4: Component nodes of a petri net.

The same way a function was created to define the transitions of states in an automaton, two functions will be created to define the connections between places and transitions. First we need to define the sets of places and transitions. P is the set of places and T the set of transitions. With this two sets we can then define those functions. The first one represents the arcs from places to transitions, and is denoted as $Pre : P \times T \to \{0, 1\}$, the second one the arcs that connects transitions to places, denoted as $Post : P \times T \to \{0, 1\}$. The value 1 is attributed to arcs that exist and 0 to the nonexistent ones.

Example 2.2 (Simple Petri Net structure)
Given $P = \{p_0, p_1\}$, $T = \{t_0, t_1, t_2\}$ *and the following transition functions:*

$$\begin{array}{cccc} Pre(p_0, t_0) = 0 & Post(p_0, t_1) = 0 & Pre(p_1, t_0) = 0 & Post(p_1, t_1) = 1 \\ Post(p_0, t_0) = 0 & Pre(p_0, t_2) = 0 & Post(p_1, t_0) = 0 & Pre(p_1, t_2) = 0 \\ Pre(p_0, t_1) = 0 & Post(p_0, t_2) = 0 & Pre(p_1, t_1) = 0 & Post(p_1, t_2) = 0 \end{array}$$

We can represent this petri net structure with the diagram on Figure 2.5

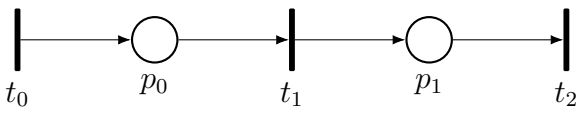


Figure 2.5: Diagram representing the petri net structure from example 2.2

A drawback from the definition of this functions, is that is not possible to have more than an arc linking two nodes, so we can generalize them to any natural number:

$$Pre : P \times T \to \mathbb{N}_0$$
$$Post : P \times T \to \mathbb{N}_0$$

with this new definition we can change the definition of $Post(p_1, t_1)$ from 0 to 2 resulting on the following petri net structure.

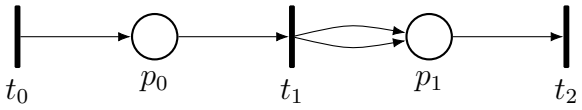


Figure 2.6: Diagram representing the petri net structure from example 2.2, but with $Post(p_1, t_1) = 2$

In order to reduce the number of arcs in a diagram, usually only one arc is drawn and a label is added with the value of its respective function, if it is greater than 1, Figure 2.7 illustrates it:

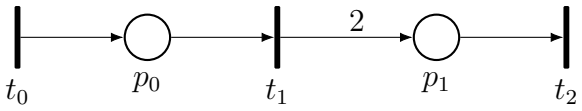
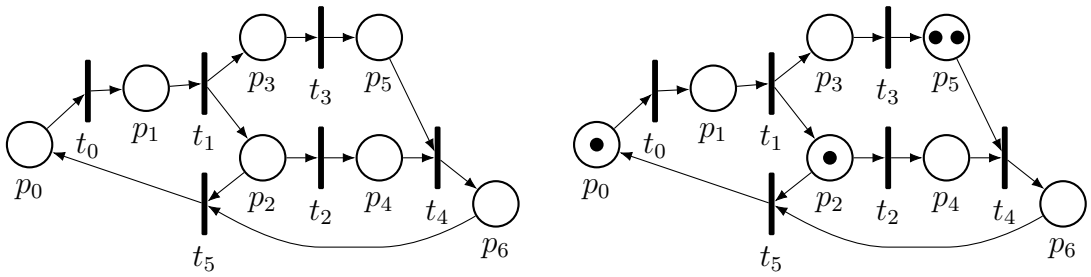


Figure 2.7: Same diagram as Figure 2.6 but with labeled arcs.

Marking

As said in the beginning of this subsection marking is used as the mechanism to represent if the condition of occurrence of a determined event is met or not. But also it can be used to represent the state of the system. The mechanism works as follows. Tokens can be designated to places and the way the tokens are distributed among places is called the marking of a petri net graph. We can define a marking function $x : P \to \mathbb{N}$ that denotes the number of tokens in a determined place. In this work, as in the majority of articles and books, the tokens will be represented as black dots inside the places.

The Figures 2.8a and 2.8b show an unmarked and a marked petri net graph.



(a) Unmarked (b) Marked

Figure 2.8: Example of unmarked and marked petri net graphs.

The marking of a petri net, can be represented as a vector of the function x applied on all places, for example the marking of the Figure 2.8b is the following vector $\mathbf{x}$

$$\mathbf{x} = \begin{bmatrix} x(p_0) \\ x(p_1) \\ x(p_2) \\ x(p_3) \\ x(p_4) \\ x(p_5) \\ x(p_6) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 2 \\ 0 \end{bmatrix}$$

This vector $\mathbf{x}$, the marking of the petri net, can be identified as the state of the petri net. So, different configurations of tokens mean different states of the system, now we only need a way to change from a state to other, in other words, move the tokens.

Firing Transitions

The mean of move the tokens is firing transitions. When an event happens and the corresponding transition is enabled, this transition is fired and tokens are moved between places. We can define the functions $I : T \to 2^P$ and $O : T \to 2^P$ that describe the set of places considered as inputs and outputs of a transition:

$$I(t_j) = \{p \in P : Pre(p, t) > 0\}$$
$$O(t_j) = \{p \in P : Post(p, t) > 0\}$$

Definition 2.3 (Enabled transition)
A transition is enabled if

$$x(p_i) \geq Pre(p_i, t_j) \text{ for all } p_i \in I(t_j)$$

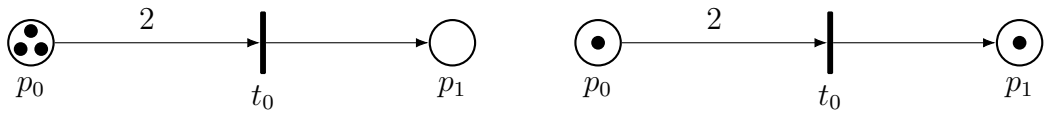
if $I(t_j) = \emptyset$, t_j *is always enabled.*

And we can define the dynamic of the petri net, how the tokens move:

Definition 2.4 (Petri net dynamics)
It is possible to define a state transition *function,* $f : \mathbb{N}^n \times T \to \mathbb{N}^n$ *, where* n *is the length of the state vector* $\mathbf{x}$*. This function* f *is defined for a transition* $t_j \in T$ *if and only if this transition is enabled. If* $f(\mathbf{x}, t_j)$ *is defined, then we create a new state vector* $\mathbf{x}'$*:*

$$x'(p_i) = x(p_i) - Pre(p_i, t_j) + Post(p_i, t_j), i = 1, \ldots n$$

As an example we can take Figure 2.9:



(a) Before firing. (b) After firing.

Figure 2.9: Example of petri net Dynamic.

The state before firing transition t_0 is $\mathbf{x} = \begin{bmatrix} 3 & 0 \end{bmatrix}^T$ and as we see $Pre(p_0, t_0) = 2$ and $Post(p_1, t_0) = 1$ so applying the petri net dynamic we can find the next state $\mathbf{x}' = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$

Once all these fundamentals are presented we can finally define a Petri Net

Definition 2.5 (Petri net)
A Petri net is defined as a five-tuple

$$PN = (P, T, Pre, Post, \mathbf{x}_0)$$

where:

P *is the set of* ***places***

T *is the set of* ***transitions***

Pre *is the* ***input incidence*** *function*

Post is the ***output incidence*** *function*
$\mathbf{x}_0$ *is the initial marking of the net*
And its dynamic is ruled by the state transition *function* f *defined at 2.9.*

To make the connection between the Petri net and the events of the system, called Σ, the alphabet, we can define a labeling function, $l : T^* \to \Sigma^*$ that makes the link between a sequence of firing transitions and a sequence of events. But each transition can only have one respective event.

Definition 2.6 (Labeled Petri net)
A Labeled Petri net is defined as a seven-tuple

$$PN = (P, T, Pre, Post, \mathbf{x}_0, \Sigma, l)$$

where:
$(P, T, Pre, Post, \mathbf{x}_0)$ *is a Petri Net*
Σ *is the set of* ***events***
l *is the* ***labeling*** *function*

Usually the events are represented in the petri net graph over its respective transition as shown in the Figure 2.10. This system has an alphabet $\Sigma = \{a, b\}$ and labeling functions $l(t_0) = a$ and $l(t_1) = b$.

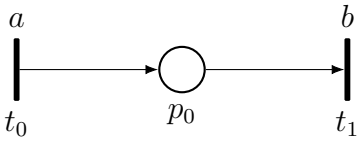


Figure 2.10: Labeled Petri net.

2.5 Control Interpreted Petri Nets

One of the greatest uses of Petri Nets, besides modeling a system, is its ability to model the control of a system. For this intent we use Control Interpreted Petri nets. It is an extension from the labeled Petri net, we add actions for places, so it is possible to change the outputs of the system, conditions to the transitions, so it is possible to change the state of the control based on the inputs of the system, and even the ability

to delay the firing transitions based on time. Another artifice used is an inhibitor arc, that prevents the firing of a transition based on the marking of the corresponding place.

Definition 2.7 (Control Interpreted Petri net)
A Labeled Petri net is defined as a seven-tuple

$$PN = (P, T, Pre, Post, \mathbf{x}_0, In, \Sigma, C, l_C, D, L_D, A, I_A)$$

where:

$(P, T, Pre, Post, \mathbf{x}_0)$ *is a Petri Net*

In *is the* ***inhibitor arc*** *function that prevents the enablement of transitions*

Σ *is the set of* ***events*** *associated to transitions*

C *is the set of* ***conditions*** *associated to transitions*

l_C *is the* ***labeling*** *function that associates a transition with events and conditions from* Σ *and* C

D *is the set of* ***delays*** *associated to transitions*

l_D *is the* ***labeling*** *function that associates a transition with a delay from* D

A *is the set of* ***actions*** *associated to places*

l_A *is the* ***labeling*** *function that assigns actions from* A *to a place*

The definition of $In : (P \times T) \to \mathbb{N}$ is that a transition t_j is inhibited if $x(p_i) \geq In(p_i, t_j)$. Inhibitor arcs are not used in this work but usually they are represented with an arc with a circle in one of its ends, as shown in Figure 2.11.

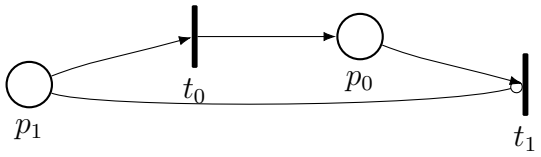


Figure 2.11: Example of Petri net with inhibitor arc.

As we can see from the definition there are two labeling functions to connect transitions, l_C and l_D. The l_c is defined for transitions with no firing delay and l_D for transitions with firing delay.

The labeling function $l_C : T^0 \to (\Sigma \times C)$ defines a pair of event and boolean condition from Σ and C respectively. A transition t_i belonging to T^0 (a subset of T that represents the transitions with no time delay) has a corresponding event, condition tuple (σ_i, c_i) For example, take a transition t_0, $\Sigma = \{\sigma_0\}$ and $C = \{c_0\}$. If a function $l_c(t_0) = (\sigma_0, c_0)$

is defined, this transition will be fired when the condition c_0 is true and the event σ_0 happens, but obviously, if and only if this transitions is enabled and not inhibited. The transition t_0 is represented graphically as shown in Figure 2.12

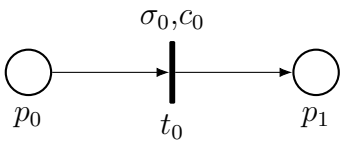


Figure 2.12: Representation of new labeling function

If the event is missing from the representation of the transition, it is equal to the λ , the always occurring event. And if the condition is missing, that means it is equal to 1, it is always *true*. If both are missing, that means the transition will be automatically executed if it is enabled.

By the other hand, the labeling function $l_D : T^D \rightarrow D$, defines a delay for the transition to be fired. A timed transition t_i, a transition in T^D (a subset of T that represents the transitions with a time delay), has a corresponding delay d_i. As an example, take a timed transition t_1 and $D = \{d_1\}$, after the enablement of the transition, it takes d_1 time units in order to be fired. In this work timed transitions are represented as bars slightly larger than normal conditions. An example of this representation we can see Figure 2.13

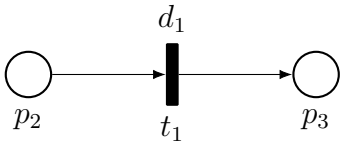


Figure 2.13: Representation of a timed transition.

Another labeling function that was created is $l_A : P \rightarrow 2^A$, assigning a set of actions belonging to A to a place. Actions can be impulse actions or continuous. A continuous action happens always that the marking of a place is greater than 0, $x(p_i) > 0$, an impulse action, on the other hand, happens only when the marking of the place changes from 0 to a value greater than 0. Actions are represented graphically as labels in places. Impulse actions are differed by a star (*) at its end. So an action F is continuous and B^* is an impulse action. Figure 2.14 show a representation of a Place with both kinds of actions.

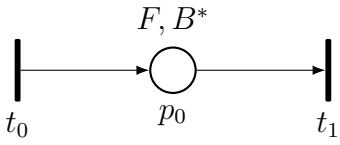


Figure 2.14: Representation of labeling of Actions.

Although these representations exist, in this work events, conditions and actions labels are suppressed from the diagrams and tables accompany the drawings showing the meaning of the transitions (firing events and conditions) and places (Actions). This choice was made because when the controlled interpreted Petri nets are very large as the ones shown in the next chapters, if the events, conditions and actions labels are long they can increase the size of the diagram.

To illustrate this better we give an example based on one example from DAVID and ALLA (1989).

Example 2.3 (Loading of a wagon)

We consider the system represented by the scheme in Figure 2.15. A wagon can be moved between the points a and b, using the inputs L and R (moving it to the left or right, respectively). At point a there is a button m that can be pressed by an operator and a limit switch called a that is activated when the wagon is on the left. At point b, an homonym limit switch is placed and activated when the wagon is on the right. There is a hopper that can be opened when the input Open is turned on and closed when not.If it is opened its content is poured. There is also a button p that is activated when the weight applied over the plate is equal or greater to the weight of a full wagon.

The objective of the control is, when the wagon is in its leftmost position and the button m is pressed, it moves to the right, stops at b, the hopper is opened and the wagon is loaded, when it is completely full it moves to the left and it stops at a waiting to be unloaded and for a next press of m to recommence the loop.

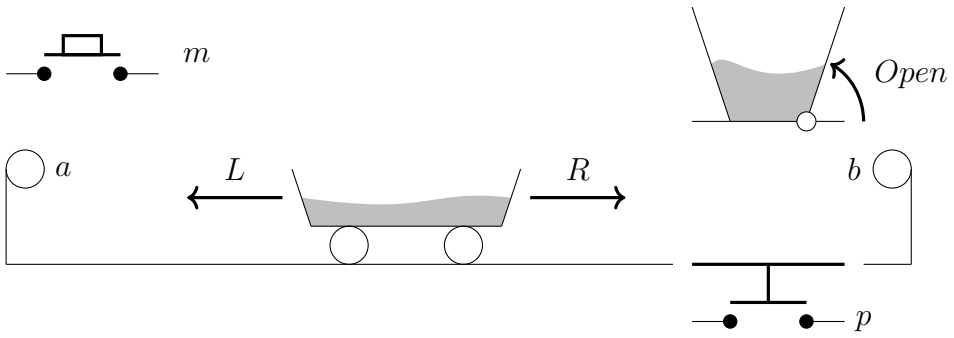


Figure 2.15: Example of System to be controlled by the Petri Net

From the description of the control it is possible to create a Control Interpret Petri Net to describe it, as the one in Figure 2.16

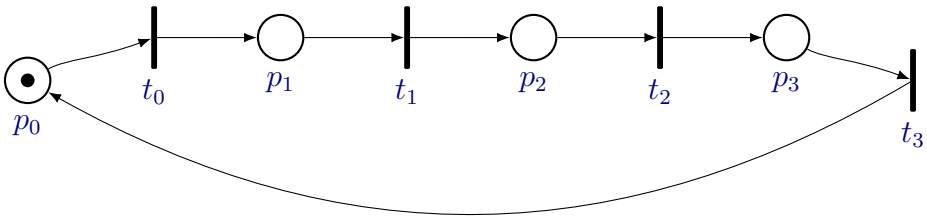


Figure 2.16: Example of Control Interpreted Petri Net to control system in Figure 2.15

The meaning/description of each place and transition is given by the following tables:

Table 2.1: Control Interpreted Petri Net Example Places.

Places	Meaning
p_0	System Stopped
p_1	R (Car Moving to the Right)
p_2	Open (Container Opened)
p_3	L (Car Moving to the Left)

Table 2.2: Control Interpreted Petri Net Example Transitions.

Transitions	Meaning
t_0	$\uparrow$ m (filling request)
t_1	$\uparrow$ b (Right Limit Switch)
t_2	$\uparrow$ p (Car is Full)
t_3	$\uparrow$ a (Left Limit Switch)

In this work as in the usual boolean notation, when just the name of a variable is given in a table it means the variable is equal to true, and when there is a bar in its top it is equal to false, so they determine conditions. E.g.: b and $\overline{b}$. And when a variable is preceded by $\uparrow$ and $\downarrow$, they determine events corresponding to its raising and falling edge.

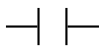
2.6 Implementation of Control Interpreted Petri Nets

Once the control of a system is modeled by a Control Interpreted Petri Net (CIPN), it is needed to implement the control in a real controller. The most used controllers in the

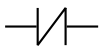
industry are PLCs. The international standard IEC 61131, defines all the standards for PLCs, and its third part (IEC 61131-3) defines five languages to program PLCs: Ladder Diagram (LD), Function Block Diagram (FBD), Structured Text (ST), Instruction List (IL) and Sequential Function Chart (SFC). One of the most used in the industry is LD, because of its resemblance with electric connections. So we are going to use LD to implement the control designed with the CIPN.

2.6.1 Ladder Logic

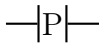
The ladder logic is based on two components, contacts and coils. Their terminals are interconnected to transmit boolean signals. This connection is similar to the ladder physical implementation, from where came its inspiration, the components were connect to boards and they formed electric circuits, turning on and off motors and other actuators, based on the combination of its inputs. The name Ladder comes from the resemblance between its structure (circuits formed in parallel one above the other) and a ladder, so each circuit is called a Rung by analogy. The logic values in a LD rung are transmitted from the left to the right of the diagram. The components let the logic "current" flow from its left terminal to the right terminal depending on some conditions, and these conditions vary from component to component. The graphical representation of the most used types of contacts and coils can be seen in Figures 2.17 and 2.18



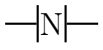
(a) Normally Opened Contact.



(b) Normally Closed Contact.

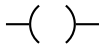


(c) Positive Edge Contact.



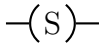
(d) Negative Edge Contact.

Figure 2.17: Types of Contacts.

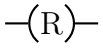


(a) Coil.

(b) Negated Coil.



(c) Set (latch) Coil.



(d) Reset (unlatch) Coil.

Figure 2.18: Types of Coils.

Contacts

Contacts represent the conditions of the ladder logic depending on inputs. These inputs can be any variable in a PLC, an external input (sensors of the system to be controlled), a variable stored in memory or the current value sent to an output from the PLC. A normally opened contact activates its right terminal (set it to $true$) if the logic value in its left terminal is $true$ and its corresponding input is equal to $true$ A normally closed contact activates its right terminal if the logic value in its left terminal is $true$ and its corresponding input is equal to $false$. The Positive Edge contact activates its right terminal only in the instant that its input change from logic value $false$ to $true$, if the logic value in its left terminal is $true$. And the Negative Edge contact activates its right terminal only in the instant that its input change from logic value $true$ to $false$.

As we can see, positive and negative contacts can be used to represent raising ($\uparrow$) and falling edge ($\downarrow$) events and normally opened and closed contacts to represent conditions (and their negation).

Coils

Coils, by the other side represent the actuation in outputs. These outputs can be a variable stored in memory or the outputs of the controller (actuators of the system to be controlled, for instance). A coil sets its output variable to true if the logic value of its left terminal is $true$, and sets the output to $false$ otherwise. A negated coil does the exact opposite, sets the output value to true if the logic value of its left terminal is $false$ and sets it to $true$ if the logic is $true$.

A set coil (or latch) sets its output variable to $true$ if the logic value of its terminal is $true$ and it remains $true$ until the variable is reset. And a reset coil (or unlatch) sets its output variable to $false$ if the logic value of its terminal is $true$ and it remains $false$ until the variable is set.

Combinational Logic

In boolean logic, in order to show functional completeness, it is need to show a complete set of connectives (a set that can create all other logic connectives as a combination of its elements). A well-know complete set is $S = \{AND, NOT\}$, binary conjunction and negation. To show that the ladder logic is functional complete we need only to present how to construct this two connectors in it. The conjunction of two inputs, can be made using two contacts in series, as shown in Figure 2.19

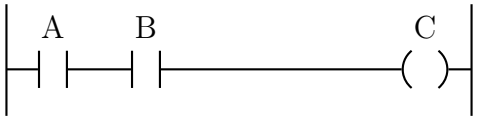


Figure 2.19: And logic in a Ladder rung.

In this case C will only be activated if A and B are equal to *true*. ($C = AB$)

The negation of a variable can be achieved by the use of a normally closed contact (Figure 2.20).

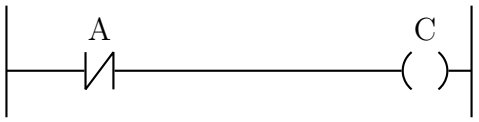


Figure 2.20: Not logic in a Ladder rung.

C will only by activated if A is $false$. ($C = \overline{A}$)

Although all logic connectives can be constructed with this two connectors, the OR connector can be achieved by the use contacts in parallel (Figure 2.21).

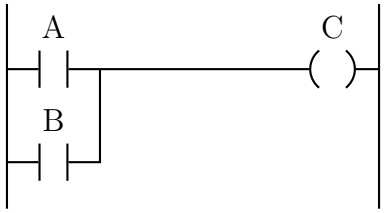
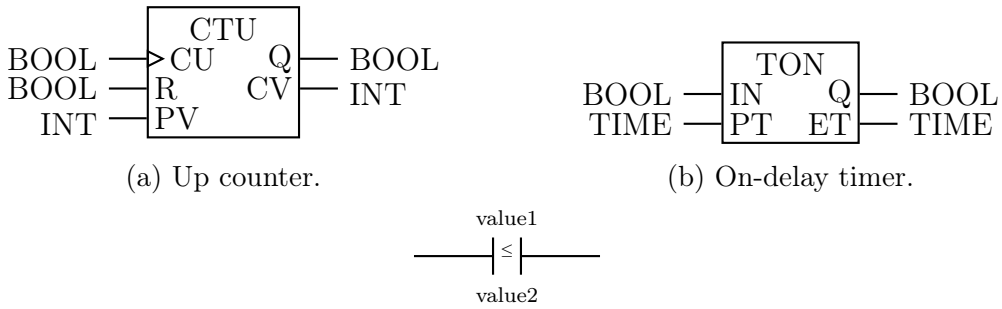


Figure 2.21: Or logic in a Ladder rung.

Function Blocks and extensions

In order to increase functionality some function blocks and extensions to contacts were created. We can see examples of theses blocks and contacts in the next figure:



(a) Up counter.

(b) On-delay timer.

(c) Less or equal comparator.

Figure 2.22: Examples of function blocks.

Up counters (Figure 2.22a) save the value of a counter in a CV variable. Every raising edge on input CU it increments CV value. If $CV = PV$, the logic value of output Q is set to 0. When the input R is true CV value is set to 0 and the output Q set to false.

On-delay timers (Figure 2.22a) set a timer when input IN is *true* and save it to ET. If $ET = PT$, the logic value of output Q is set to *true*. But if, meanwhile the counting, the value of IN returns to $false$, ET is reset to 0.

Comparator contacts as the less or equal comparator in Figure 2.22c, work similarly to contacts, but instead of an input as a condition, there are two inputs ($value1$ and $value2$) and the condition is a comparison between both of them. In this case, the contact is activated once its left terminals' logic value is *true* and $value1 \leq value2$.

Other blocks and functions can be found in the IEC 61131-3, as adders, subtractors, communication blocks etc.

2.6.2 Conversion from CIPN to LD

A simple method of conversion from CIPN to LDis presented in MOREIRA and BASILIO (2013).

It consists in dividing the CIPN in 4 modules:

1. A module of external events
   To create conditions to the firing of transitions based on external events (inputs)

2. A module of firing conditions
   To indicate what condition will be fired using the Pre, and In functions, the conditions found on the last module and time delays (if it is a timed transition)

3. A module of Petri Net dynamics
   Uses the Pre and $Post$ functions to determine the tokens "motion"

4. A module of actions
   Determines the places where each action is performed.

In this work, the external events and firing conditions was combined in order to reduce the size of the program. But every module will be described as in MOREIRA and BASILIO (2013).

External events

As external events are associated with positive and negative edge of the inputs of the system, in this module, positive and negative edge contacts are used to detect the rising and falling edge events, and they are stored in variables using coils, a variable is created for every event. For visibility's and organisation's sake a rung is used for each event, resulting $|\Sigma|$ rungs.

Firing Conditions

As said in 2.5, for a transition t_j to be fired, first it needs to be enabled ($x(p_i) \geq Pre(p_i, t_j)$ for all $p_i \in I(t_j)$), not inhibited ($x(p_i) < In(p_i, t_j)$ for all $p_i \in I(t_j)$) and the conditions and events $\sigma_j c_j$ are met or the delay d_j is elapsed, depending on the kind of transition. As places can have multiple tokens, we can use *int* variables to store the number of tokens, and comparator contacts to determine if the transitions are enabled and not inhibited. When a place can only bear at most one single token for all markings of the Petri net, a *bool* variable can be used to store the number of tokens, in this case a single normally open contact can be used to determine if there is a token in that place. The time delays are implemented using on-delay timers. The state of fulfilment of the conditions is stored in variables, one for each transition. Similarly, for organisation's sake a rung is used for each transition, resulting $|T|$ rungs.

Petri Net Dynamics

In this module the dynamic of the tokens is implemented. If the condition for the firing of a transition is fulfilled (represented by normally open contacts), adders and subtractors can be used to represent the movement of tokens, increasing and decreasing the values from the *int* variables that represent the marking of each place, if their capacity is greater than one, if not they are represented by boolean variables, and instead

Set and Reset coils can be used to represent a token entering a place and a token exiting another, respectively. Again, for organisation's sake a rung is associated for each transition, resulting $|T|$ rungs.

Actions

In the Action module, we use coils to act on the outputs. Depending on the type of action and the logic of the control, set/reset coils or normal coils can be used. The condition to activate/deactivate the output is the presence of a token in the places where the action is performed, this can be achieved by comparing the numbers of tokens in a place. If the tokens of a place is represented by an *int* variable, we use greater or equal comparators, but if it is represented by a *bool* variable, a normally opened contact is enough. If the action is an impulse action we can put a positive edge in series with the places contact or the comparator.

Differently from the original article, we will use one rung per each action in A, resulting in $|A|$ rungs. This is made because in RENAULT (2017) it is recommended to perform an action in only one rung, and group the conditions using OR connectors. In the industry this is important in order to reduce errors and to ease the debugging process.

Example

An example of this conversion can be given using the same CIPN from example 2.3. As said, the external events and firing condition modules are grouped in same module in this work. The converted Ladder Logic can be seen in Figure 2.23.

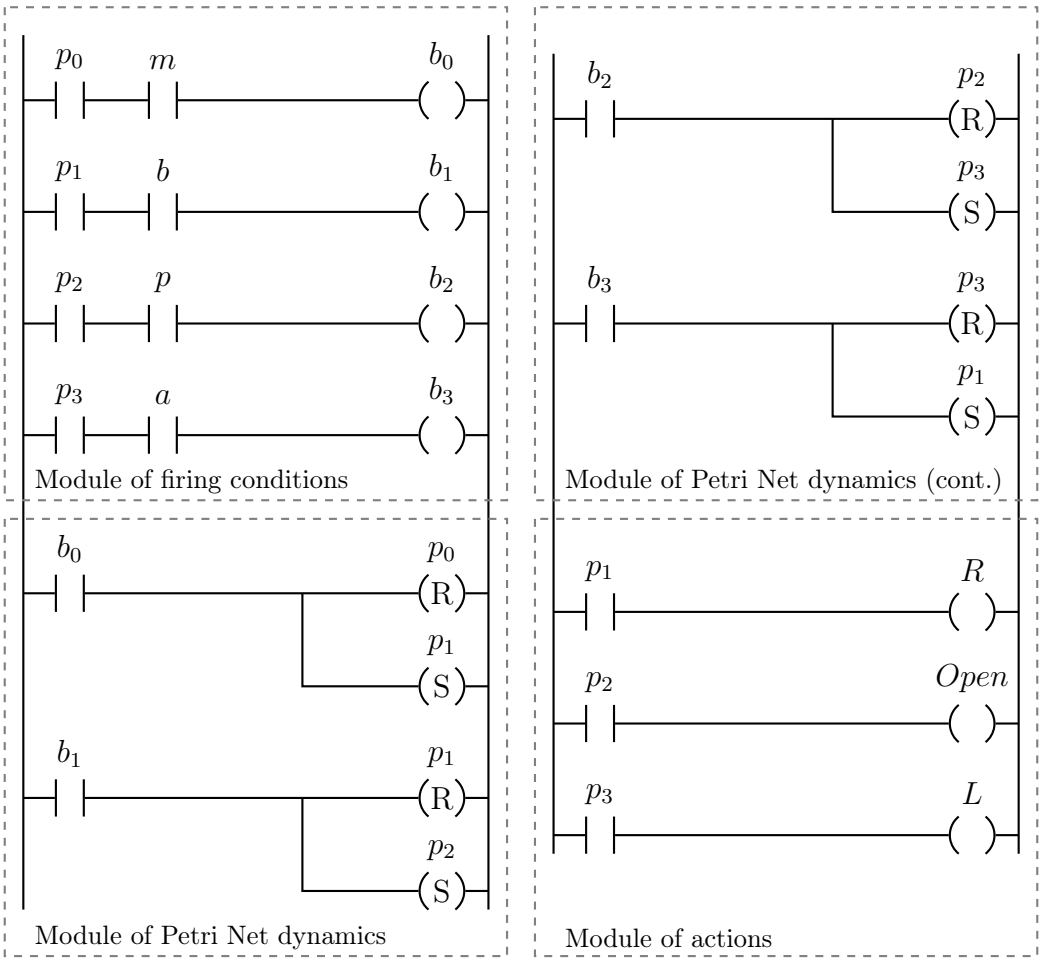
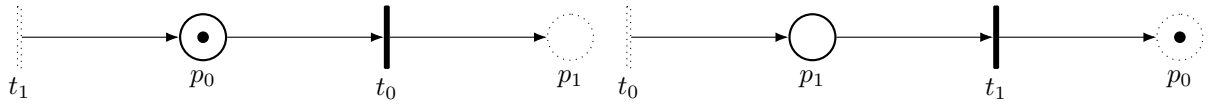


Figure 2.23: Example of Control Interpreted Petri Net converted to Ladder.

2.6.3 Petri Net divided in multiple PLCs

A problem that can occur, is the case where a CIPN must be divided in multiple PLCs, because of how the assembling of the plant was (some inputs/outputs are only connected to a single PLC, and other inputs/outputs to another PLC), this division can be arbitrarily decided by the person who will implement the control.

In Figure 2.25a, it is shown an example of a CIPN divided between 2 PLCs.



(a) Petri Net on PLC 1.

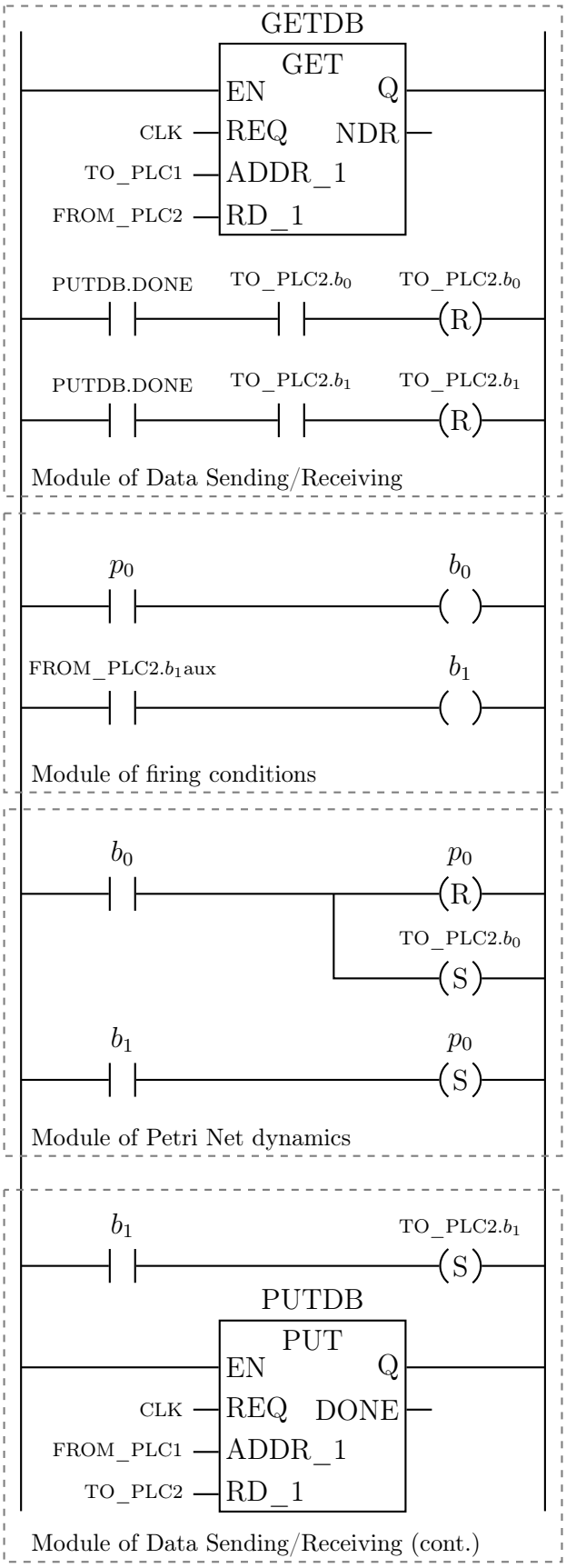
(b) Petri Net on PLC 2.

Figure 2.24: Example of Petri Net divided between 2 PLCs.

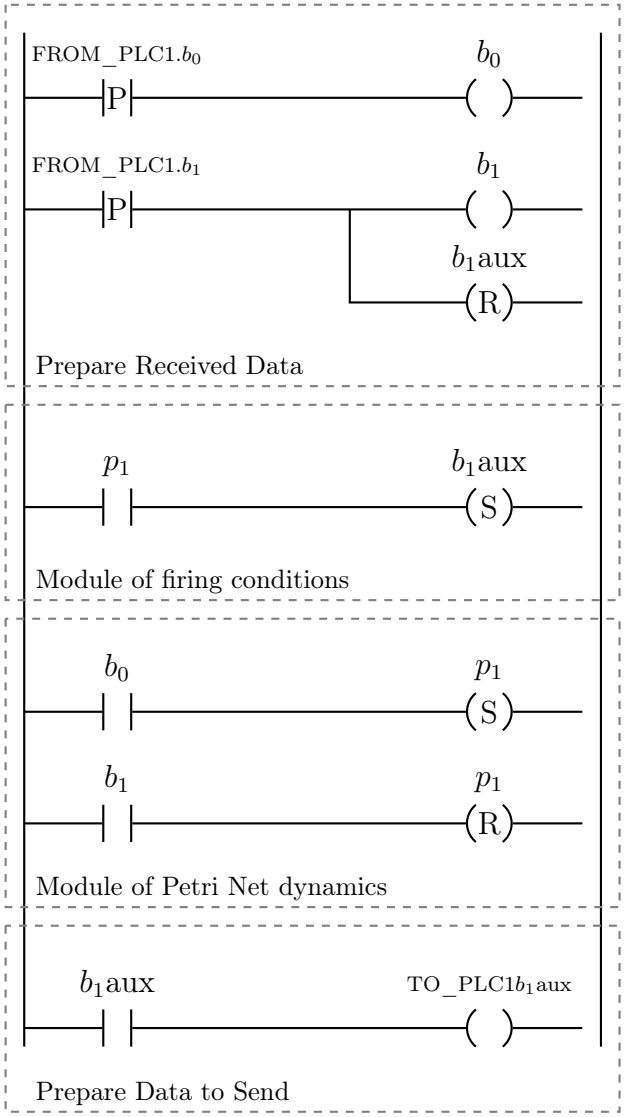
As we can see, in Figures 2.25b and 2.24b there are dotted transitions and places. In this work, we will represent as dotted, transitions and places that are not represented in the same figure, in order to show that the figures are connected forming a complete Petri Net together. In the digital form[1] of this thesis it is possible to travel between figures that are not in the same page just by clicking in the name of the corresponding dotted place/transitions.

In this work, in order to solve this problem.

[1]Available at: https://github.com/Accacio/docsTCC/blob/master/monografia.pdf



(a) Ladder Logic on PLC 1.



(b) Ladder Logic on PLC 2.

Figure 2.25: Example of Petri Net divided between 2 PLCs.

Figure 2.25a

2.7 Identification

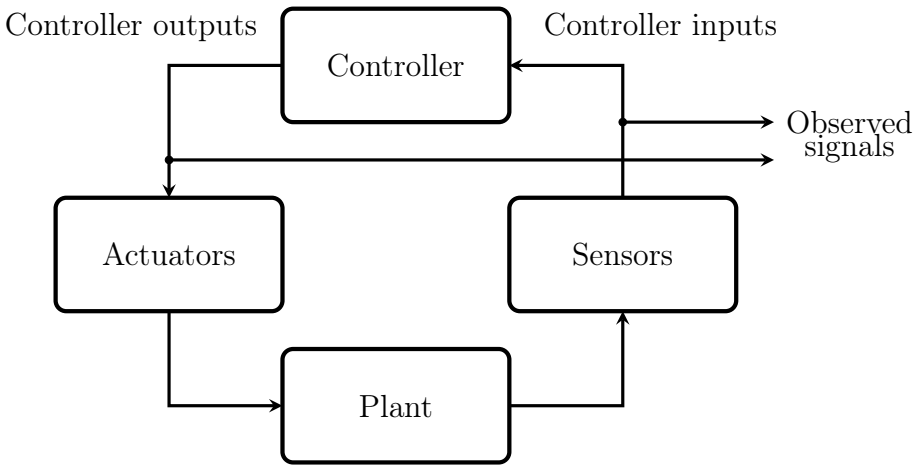


Figure 2.26: Input/Output Process model

2.7.1 DAOCT

Definition 2.8 (Deterministic Automaton With Outputs and Conditional Transitions (DA

A Deterministic Automaton, denoted by DAOCT, is a nine-tuple

$$DAOCT = (X, \Sigma, f, \lambda, R, \theta, x_0, X_f)$$

where:

X *is the set of* ***states***
Σ *is the finite set of* ***events***
$\Omega \subset \mathbb{N}_1^{m_i+m_o}$ *is the set of* ***I/O vectors***
f: $X \times \Sigma^\star \to X$ *is the* ***deterministic transition function***
$\lambda : X \to \Omega$ *is the* ***state output function***
$R = 1, 2, \ldots, r$ *is the set of* ***path indices***
$\theta : X \times \Sigma \to 2^R$ *is the* ***path estimation function***
x_0 *is the* ***initial state***
$X_f \subseteq X$ *is the set of* ***final states***

Identification algorithm adapted from MOREIRA and LESAGE (2018)

Algorithm 1: Identification Algorithm

Input: Modified observed paths p_i^k, for i= 1,. . . ,r
Output: DAOCT = (X,Σ,Ω,f,λ,R,θ,x_0,X_f)

Create an initial state x_0, and define $\lambda(x_0) = \tilde{\lambda}(x_0) = y_{1,1}$
$X = \{x_0\}, X_f = \emptyset, R = \emptyset$
for $i = 1$ **to** r **do**
    $R = R \cup \{i\}$
    for $j = 1$ **to** $l_i - 1$ **do**
        Find the State $x \in X$ such that $\tilde{\lambda}(x) = y_{i,j+1}$
        if $\tilde{\lambda}(s) \neq y_{i,j+1}$ *for all* $s \in X$ **then**
            Create state x' and define $\tilde{\lambda}(x') = y_{i,j+1}$
            $X = X \cup \{x'\}$
            $\lambda(x') = \tilde{\lambda}_l(x')$
        else
            Find $x' \in X$ such that $\tilde{\lambda}(x') = y_{i,j+1}$
        end
        $f(x, \sigma_{i,j}) = x'$
        Add i to $\theta(x, \sigma_{i,j})$
        if $j = l_i - 1$ **then**
            $X_f = X_f \cup \{x'\}$
        end
    end
end

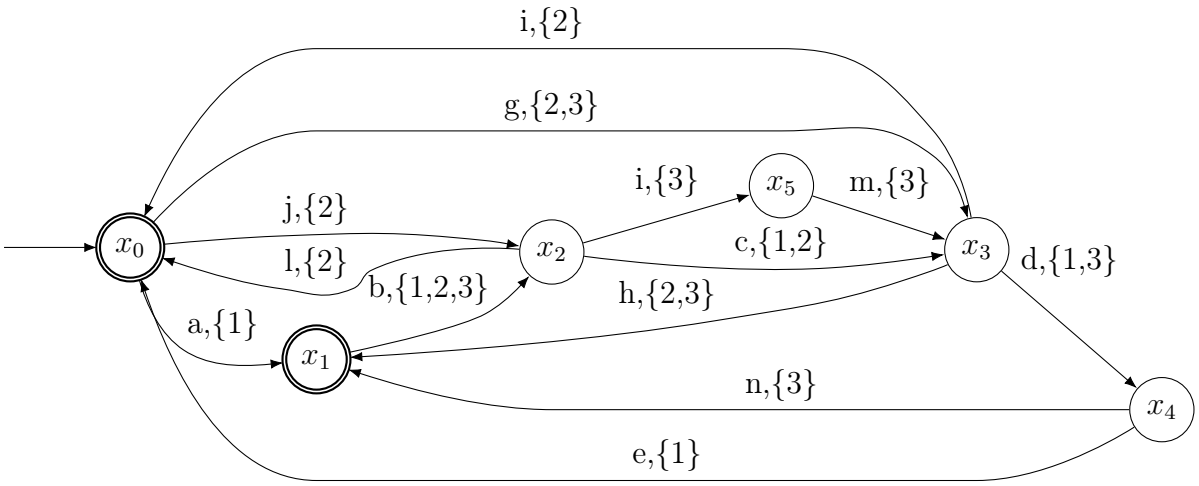


Figure 2.27: Diagram representing the DAOCT from example number

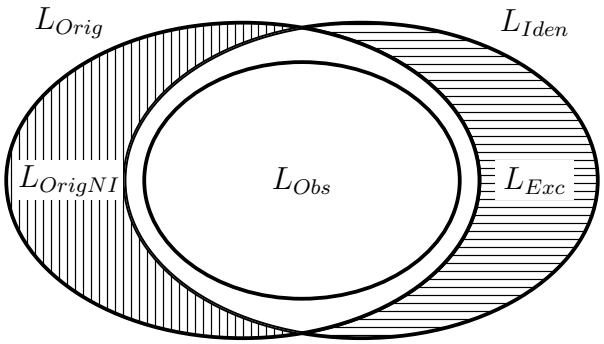


Figure 2.28: Venn diagram showing relations between L_{Orig}, L_{OrigNI}, L_{Obs}, L_{Exc} and L_{Iden}

Bibliography

ANTUNES FLORIANO, L. *Sincronização de Sistemas a Eventos Discretos Modelados por Redes de Petri Usando Lugares Comuns.* Undergraduate Project, Universidade Federal do Rio de Janeiro, 2019.

CABRAL, F. G., MOREIRA, M. V. "Synchronous Codiagnosability of Modular Discrete-Event Systems", *IFAC-PapersOnLine*, v. 50, n. 1, pp. 6831–6836, 2017.

CABRAL, F. G., MOREIRA, M. V., DIENE, O., et al. "Petri Net Diagnoser for Discrete Event Systems Modeled by Finite State Automata", *IEEE Transactions on Automatic Control*, v. 60, n. 1, pp. 59–71, 2015a.

CABRAL, F. G., MOREIRA, M. V., DIENE, O. "Online Fault Diagnosis of Modular Discrete-Event Systems". In: *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*, pp. 4450–4455. IEEE, 2015b.

CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to discrete event systems.* Springer Science & Business Media, 2009.

DAVID, R., ALLA, H. L. *Du Grafcet aux réseaux de Petri.* Hermes, 1989.

DAVID, R., ALLA, H. *Discrete, continuous, and hybrid Petri nets*, v. 1. Springer, 2005.

DAVIS, R., HAMSCHER, W. "Model-based reasoning: Troubleshooting". In: *Exploring artificial intelligence*, Elsevier, pp. 297–346, 1988.

FRANÇA, T. C. *Projeto de um Sistema Supervisório para uma Planta Mecatrônica de Estocagem de Peças.* Undergraduate Project, Universidade Federal do Rio de Janeiro, 2015.

JOURDAN, G., BOCHMANN, G. V. "On testing 1-safe Petri nets". In: *2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*, pp. 275–281. IEEE, 2009.

KALOUPTSIDIS, N. *Signal processing systems: theory and design*, v. 28. Wiley-Interscience, 1997.

KLEIN, S., LITZ, L., LESAGE, J.-J. "Fault detection of discrete event systems using an identification approach", *IFAC Proceedings Volumes*, v. 38, n. 1, pp. 92–97, 2005.

KUMAR, R., TAKAI, S. "Comments on "Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems" versus "Decentralized Failure Diagnosis of Discrete Event Systems": Complexity Clarification", *IEEE Transactions on Automatic Control*, v. 59, n. 5, pp. 1391–1392, 2014.

LANCELLOTE JÚNIOR, F. P. *Automação de uma Planta Mecatrônica Modelada por uma Rede de Petri Interpretada para Controle*. Undergraduate Project, Universidade Federal do Rio de Janeiro, 2014.

LUCIO, M. L. *Diagnóstico de Falhas Sincronizado de uma Planta de Manufatura*. Undergraduate Project, Universidade Federal do Rio de Janeiro, 2015.

MOREIRA, M. V., BASILIO, J. C. "Bridging the gap between design and implementation of discrete-event controllers", *IEEE Transactions on Automation Science and Engineering*, v. 11, n. 1, pp. 48–65, 2013.

MOREIRA, M. V., LESAGE, J.-J. "Enhanced Discrete Event Model for System Identification with the Aim of Fault Detection", *IFAC-PapersOnLine*, v. 51, n. 7, pp. 160–166, 2018.

MOREIRA, M. V., BASILIO, J. C., CABRAL, F. G. ""Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems" Versus "Decentralized Failure Diagnosis of Discrete Event Systems: A Critical Appraisal", *IEEE Transactions on Automatic Control*, v. 61, n. 1, pp. 178–181, 2016.

OLIVEIRA, V. D. S. L. *Protocolo de Comunicação Profinet para Redes de Automação*. Undergraduate Project, Universidade Federal do Rio de Janeiro, 2016.

OPPENHEIM, A. V., WILLSKY, A. S., NAWAB, S. "Signals and Systems (Prentice-Hall signal processing series)", 1996.

PITANGA CLETO DE SOUZA, R. *Um Modelo Temporizado para a Identificação de Sistemas a Eventos Discretos*. Undergraduate Project, Universidade Federal do Rio de Janeiro, 2019.

RENAULT. *Règles de qualité logicielle pour la programmation des automatismes*. Technical repotrt, Département Ingénierie Automatismes et Robotique. Groupe Renault, 2017. Available at: <http://cnomo.com/fichiers/2/15981/Eb03j0010_A_fr%20.pdf?download=true>.

ROCHA PEREIRA, A. P. *Automação de uma Planta Mecatrônica de Montagem e Armazenamento de Cubos Utilizando Comunicação entre Controladores Lógicos Programáveis*. Undergraduate Project, Universidade Federal do Rio de Janeiro, 2019.

ROTH, M., LESAGE, J.-J., LITZ, L. "An FDI method for manufacturing systems based on an identified model", *IFAC Proceedings Volumes*, v. 42, n. 4, pp. 1406–1411, 2009.

SIEMENS. *S7-1500 Web server Function Manual*. SIEMENS, a. Available at: <https://support.industry.siemens.com/cs/attachments/59193560/s71500_webserver_function_manual_en-US_en-US.pdf?download=true>.

SIEMENS. *Creating Userdefined Web Pages on S7-1200 / S7-1500*. SIEMENS, b. Available at: <https://support.industry.siemens.com/cs/attachments/59193560/s71500_webserver_function_manual_en-US_en-US.pdf?download=true>.

SIEMENS. *S7-1500 Structure and Use of the CPU Memory*. SIEMENS, c. Available at: <https://support.industry.siemens.com/cs/attachments/59193101/s71500_structure_and_use_of_the_PLC_memory_function_manual_en-US_en-US.pdf?download=true>.

VERAS, M. Z., CABRAL, F. G., MOREIRA, M. V. "Distributed Synchronous Diagnosability of Discrete-Event Systems", *IFAC-PapersOnLine*, v. 51, n. 7, pp. 88–93, 2018.