Black-Box identification of automated discrete event systems

Ana Paula Estrada Vargas

ENSC- 2013 / 435

THESE DE DOCTORAT

DE L'ECOLE NORMALE SUPERIEURE DE CACHAN - FRANCE

et

DU CINVESTAV, UNITE DE GUADALAJARA - MEXIQUE

Présentée par

Mademoiselle Ana Paula ESTARDA VARGAS

pour obtenir le grade de

DOCTEUR DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Domaine : Électronique – Électrotechnique – Automatique

et de

DOCTEUR ès SCIENCES DU CINVESTAV

Sujet de la thèse :

Black-Box identification of automated discrete event systems

Identification "boîte-noire" des systèmes automatisés à événements discrets

Thèse présentée et soutenue à Guadalajara le : 20 février 2013 devant le jury composé de :

A. Ramírez Treviño	Prof. au Cinvestav, Guadalajara - Mexique	Président
H. Alla	Prof. à l'univ. Joseph Fourier, Grenoble - France	Rapporteur
C. Seatzu	Prof. à l'Univ. de Cagliari, Italie	Rapporteur
J.-J. Lesage	Prof. à l'ENS de Cachan, france	Directeur de Thèse
E. López Mellado	Prof. au Cinvestav, Guadalajara - Mexique	Directeur de Thèse

Nom du Laboratoire : LURPA
ENS CACHAN/EA 1385
61, avenue du Président Wilson, 94235 CACHAN CEDEX (France)

Acknowledgement

The research work in thesis has been the result of collaboration between CINVESTAV Unidad Guadalajara and LURPA of ENS de Cachan.

I want to thank to both institutions for the opportunity of developing my formation in the corresponding Doctoral programs. This research work has been possible thanks to financial support of CONACYT that provided my scholarship, and also the Région Île de France.

I am very grateful with my advisors Dr. Luis Ernesto López Mellado and Prof. Jean-Jacques Lesage for their aid and support during the development of this work. I really appreciate all the knowledge, assistance, and guidance they gave me throughout the thesis project and for their support during this collaboration.

One important component of this research has been provided by the working environment in both Institutions.

I want to thank my Professors and my colleagues from CINVESTAV, specially to F. Ramos, M. Siller, R. González, A. López, A. Lutz, A. Raymundo, B. Gudiño, C. Boyain, E. Salvador, F. Lombera, G. Olascuaga, G. Torres, K. Jaime, K Ríos, L. Gutiérrez, L. Real, M. Trejo, M. Díaz, S. Jáuregui, V. Fernández for their valuable company and their disposition to help every time I needed something.

I am also grateful with to my co-workers from LURPA for their patience and support during my stays at Cachan: A. Guiot, A. Zuquete, D. Aza-Vallina, F. Abecassis, G. Merle, M. Caux, M. Zhang, N. Audfray, P.-A. Brameret, P.-Y. Chaux, S. Benichou, T. Lemattre, V. Lacharnay. I thank specially to B. Denis, J.-M. Roussel, A. Guignard, and M. Danancher, for their help during the experimental work.

At last but not the least, thanks to the reviewers of this thesis and jury members: C. Seatzu, H. Alla, A. Ramírez, F. Ramos for their valuable comments that helped to improve this thesis manuscript.

Esta tesis está dedicada a mis padres
Ma. Graciela Guadalupe Vargas Serrano y Pedro Arturo Estrada Quezada.

Gracias por ser el viento bajo mis alas.

This thesis is dedicated to my parents
Ma. Graciela Guadalupe Vargas Serrano and Pedro Arturo Estrada Quezada.
Thank you for being the wind beneath my wings.

Index

Introduction

Identification allows building systematically a mathematical model that describes the behaviour of an unknown or ill-known system based on the observation of its evolution. In the case of discrete event systems (DES), observations consist of data revealing the system activity: sequences of operations, events, messages, etc., and the models are abstract machines that reproduce the observed behaviour.

DES identification has been first addressed as a problem of grammatical inference. In [Gold, 1967] a finite automaton (FA) is built from positive samples of accepted words. Later several methods for obtaining Mealy [Kella, 1971] [Veelenturf, 1978] and Moore [Biermann and Feldman, 1972] [Veelenturf, 1981] machines have been proposed. Also context free grammars building has been studied [Levy, 1978], [Takada, 1998], [Ishizaka, 1990].

Identification methods yielding Petri net (PN) models have been proposed for coping with more complex systems exhibiting concurrent behaviour. In [Hiraishi, 1992] an algorithm for constructing Petri net models is presented. First, the language of the target system is identified in the form of deterministic FA (DFA). Then, the algorithm obtains from the DFA the structure of a PN that accepts the obtained language.

The problem, seen from the point of view of identification, and not only as a grammatical inference problem has been addressed in literature in various formulations and from diverse approaches. The works summarized in [Cabasino, 2009] obtain a Petri net system from the knowledge of the language it generates, i.e. the set of transition sequences that can be fired from the initial marking. Such works, classified later as synthesis methods in [Cabasino, 2013], differ from the black-box identification approach held in this thesis because considered transitions are unknown; that is, the only available information about the system is the input and output signals evolution. Besides, some of the stated hypotheses on the so called approaches are not well adapted for real complex DES, particularly the assumptions regarding the entire system language observation and the existence of counter examples. In practice, only part of the language is observed, especially when there is a lot of parallelism in the system.

In [Ould El Medhi, 2006] several algorithms are introduced to synthesize a Petri net with regard to an event propagation set. However, distinction between input and output signals is not made and obtained models do not express how inputs and outputs of the system are interrelated to produce the observed behaviour, although it is the core of a reactive system.

In [Meda, 2002a] it is described a method to incrementally construct an IPN model from a single output vectors sequence. The considered DESs to identify must be event-detectable by the outputs. Applying this method to the identification from an I/O sequence would lead to models in which same output changes caused by different input evolutions would not be differentiated and exceeding behaviour could be introduced.

The method presented in [Klein, 2005a] obtains Automata models representing a set of cyclic I/O sequences. This method also considers automated systems. However, in the obtained models, structural information as parallelism cannot be explicitly expressed. An extension of this work has been presented in [Roth, 2010], which allows splitting the system on concurrent parts. Even if modelled subsystems represent parallelism, the method is strongly adapted for fault detection purposes.

In [Dotoli, 2008] an event sequence is observed, as well as the corresponding output symbols of a DES to produce an IPN model, in which the sequence and the observed output vectors are reproducible. This methodology requires the knowledge of an event list, which is not available in the context of black-box identification problem treated in this work. An alternative to this lack of events list could be the consideration of all the observed input changes. In this case, models with several paths describing input changes would be constructed, in which some input-output relations would not be explicitly observed.

Process mining is a research domain that can be considered as similar to system identification: it consists on discovering behavioural models of the processes that capture the structured orderings of activities in a workflow.

The goal of the method presented in [Cook, 2004] is to identify gross patterns of a workflow behaviour that can be useful for understanding the system. Statistical and probabilistic analyses are made, especially to determine when concurrent behaviour is occurring, and the dependence relationships that may exist among observed events.

In [van der Aalst, 2004] the workflow mining problem is also faced. The input of the algorithm is a workflow log in which several workflow instances composed by several tasks, which have been recorded sequentially, even if they may be executed in parallel. Based on the information in the workflow log and by making some assumptions about completeness of the log, a process model in the form of a workflow net is deduced by a so called α-algorithm.

In this thesis, it is addressed the problem of identification of automated DES from a single input-output sequence describing the external observed behaviour of the system. The work focuses on systems composed by a plant and a programmable logic controller (PLC) running in a closed loop. Two identification algorithms are proposed, allowing the creation of IPN models representing approximately the observed behaviour.

Firstly, different approaches adopted in recent publications are reviewed. An overview of recent identification approaches and a comparative study of some techniques is presented.

Afterwards, we describe the problem addressed in this thesis, particularly the PLC and plant compound system operation and the data collection process for identification. We describe technological issues of both aspects which are not considered by previous methods.

Two identification methods are presented. The first one is inspired from the grammatical inference techniques and allows constructing, using an identification parameter κ, an IPN model to represent in detail the behaviour of a system from a single input-output sequence. The proposed method yields an IPN model which represents exactly the same language of length $\kappa+1$ than that generated by the system without taking into account information a-priori about the system other than its input and output signals.

The second one is a statistical method which allows the construction of compact and expressive IPN models representing complex industrial DES behaviour. The method computes the reactive part of the system by means of a statistical analysis of the input-output sequence, yielding a net composed by observable places and labelled transitions. The model is completed with the addition of non-observable places representing the internal behaviour inferred by analysis of the input output sequence.

Both methods are based on polynomial-time algorithms. They have been implemented as software tools and tested with several case studies. The results of the identification of two real systems in operation are illustrated and compared.

This thesis is organized as follows:

- Chapter 1 is devoted to the analysis of existing identification techniques.
- Chapter 2 presents some definitions about DES identification and explains the characteristics of the problem addressed in this work.
- Chapter 3 introduces an algorithm for DES identification from a single input-output sequence, including an analysis of its principal properties and characteristics and some examples to illustrate the application of the method. Limitations of this algorithm are finally analyzed.
- Chapter 4 describes a methodology to find a compact and complete representation of the behaviour of a system.
- Chapter 5 talks about experimental work.
- Conclusions include a summary of the main features of the contributions in this thesis and give perspectives for extending the research work.
- Finally, in Appendix A, the definition of IPN used in the present thesis is included.

Chapter 1

Identification methods of Discrete Event Systems

Abstract. This chapter surveys the identification techniques of discrete event systems found in the literature and analyses recent approaches addressing the identification problem. A comparative study of such approaches is made.

1.1. Methods derived from language theory

Pioneer works on identification have been developed in computer science, where the problem of obtaining a language representation from sets of accepted words has been dealt since a long time. Such methods are generally referred as *languages inference techniques* or *learning techniques*.

Gold´s method [Gold, 1967] processes positive samples: an infinite sequence of examples such that the sequences contain all and only all the strings of the language to learn.

The Probably Approximately Correct (PAC) learning technique proposed in [Valiant, 1984] learns from random examples and studies the effect of noise on learning from queries.

The query learning model proposed in [Angluin, 1988] considers a learning protocol based on a "minimally adequate teacher"; this teacher can answer two types of queries: membership query and equivalence query.

Several works that have adopted state machines as representation model, allow describing the observed behaviour. In [Booth, 1967] a method to model a language as Moore or Mealy machines is presented. The system under investigation is placed within a test bed and connected to a so called experimenter, which generates the input signals and records the output signals of the system. The identification can be started considering a very few number of states. If, at some point of the experiment, it is impossible to find a correct machine with the assumed number of states, the identification is started again considering a machine with one more state.

The method proposed in [Kella, 1971] allows obtaining models representing Mealy machines from a single observed input-output sequence. The algorithm lists all reduced machines which may produce the given sequence. The construction principle is the merging of equivalent states.

In [Biermann and Feldman, 1972] a method for the identification of non deterministic Moore machines based on a set of input output sequences is presented. All the sequences start in the same initial state. The identification principle is the reduction of an initial machine represented as a tree.

The method presented in [Veelenturf, 1978] processes simultaneously a sample of sequences to produce stepwise convergent series of Mealy machines, such that the behaviour of every new machine includes the behaviour of the previous one. At each step, the last obtained machine is analysed and completed by adding transitions and possibly new states.

Later, in [Veelenturf, 1981] an algorithm to identify a unique Moore machine generating the behaviour observed during m sequences starting at the same initial state is proposed. The learning procedure operates in three steps: induction, contradiction, and discrimination. A state can never be deleted and only transitions between states can be modified. This method is improved in [Richetin, 1984], which proposes two algorithms to identify multiple systems as well as systems that may not be initialized between two records.

The identification problem for context free grammars (CFGs) needs, beside given examples, some additional structural information for the inference algorithm [Levy, 1978].

[Ishizaka, 1990] has investigated a subclass of CFGs called simple deterministic grammars. A polynomial time algorithm that allows an exact identification of a simple deterministic language is given.

In [Takada, 1998] it has been shown that the grammatical inference problem for even linear languages can be reduced in polynomial time to the inference of regular languages.

Other works use as description formalism Petri net models. In [Hiraishi, 1992] an algorithm for synthesising Petri net models is presented. The proposed algorithm has two phases. In the first phase, the language of the target system is identified under the form of a DFA. In the second phase, a Petri net that accepts the same language as the DFA is built.

1.2. Recent approaches for DES Identification

In recent years, the scientific community has proposed identification approaches (based on Petri net or automata) for obtaining approximated models of DESs whose behaviour is unknown or ill-known. In the context of automated DESs, identification methods can be complementary to established modelling techniques; identification builds a closed-loop controller-plant model, which is more classically obtained by a composition of models of controller and plant. Several approaches for identifying DESs have been proposed in literature and compared in [Estrada, 2010a]. We present an overview of such approaches; thus for further details please consult the references.

1.2.1. Progressive identification

The problem addressed in this work is to build a model for a DES as it evolves from the observation of its output signals [Meda, 1998], [Meda, 2000a], [Meda, 2000b], [Meda, 2001], [Meda, 2002a], [Meda, 2002b], [Meda, 2003], [Meda, 2005]. A sequence of models is built in such a way that the current model acquires more details than the previous one approaching to the actual model of the system.

The identification approach proposes to compute an Interpreted Petri Net (IPN) model describing the behaviour of the unknown DES. See Appendix A for an IPN definition.

Some assumptions are considered in this work:

- The system to be identified can be described by a live, 1-bounded and cyclic IPN Q.
- Q is event-detectable by the output (the same change of outputs cannot be provoked by different transitions).
- The transitions of Q are not fired simultaneously and Q has not self-loops.

The algorithm receives a sequence of output signal values obtained from observation of the working system. The algorithm returns an IPN in which every observable place represents one of the sensors of the system.

The identification strategy is based on the reconstruction of the cyclic components of the system model, by processing cyclic sequences of transitions (called m-words) computed from the observed output symbols.

During the on-line operation of the identification process, the m-words are computed and then the new model is built adding, removing, or updating dependencies (non-observable places) between the transitions.

The model synthesis procedure performs mainly two tasks: the computation of the observable part of the system and the inference of the non observable part of the system. The first task is made directly from the observation of the output system signals, while the second task, rather difficult, derived a more detailed study about the dependencies formed by a non observable place into a model. The proposed identification algorithm is succinctly described below.

Algorithm 1.1

1. Read the vectors of output symbols o_1, o_2,... generated by the system.
2. Detect an output word when the first and last output symbols are the same.
3. For any two consecutive output symbols compute a transition that represents the output change (if the output was calculated before, take the same transition).
4. Compute an m-word adding each computed transition in the step above.
5. Compute the non observable places.
   a. to constrain the firing order of the transitions
   b. to compute the t-component associated with the m-word
6. Update the computed IPN model with the information provided by the m-word allowing the firing of all computed m-words, inferring t-semiflows of the system.
   a. computing new observable places and transitions
   b. removing or adding dependencies (possibly merging places) updating the computed real t-semiflows.

Example 1.1. In order to illustrate the method, we take from [Meda, 2002a] the following example of a system with 11 output signals. We show the models generated when new m-words are computed from the outputs of the system. For sake of brevity, not all steps of the algorithm are shown.

Step 1. Observe the first output symbols:

$o_1 = [00000000000]^T$, $o_2 = [10000000000]^T$, $o_3 = [00000000000]^T = o_1$, $o_4 = ...$

Step 2. The first cyclic observed sequence is $o_1 o_2 o_1$

Step 3. t_1 will represent the transition from o_1 to o_2 and t_2 the transition from o_2 to o_1

Step 4. The m-word resulting is $m_1 = t_1 t_2$

Step 5. The t-component associated with the m-word $t_1 t_2$ is shown in Figure 1.1.

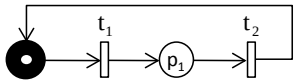


Figure 1.1 t-component associated with $m_1 = t_1 t_2$

Step 6. The first t-semiflow inferred is $W_1 = m_1$.

After the next output word is treated with steps 1-4, it is obtained the m-word $m_2 = t_3 t_4$. Its respective t-component associated is added to infer a new t-semiflow $W_1 = m_1 m_2$ in step 6, as shown in Figure 1.2.

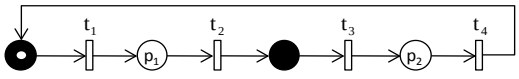


Figure 1.2 t-semiflow inferred $W_1 = m_1 m_2$

After computing of m-words $m_3 = t_6t_7$, $m_4 = t_5t_8$, $m_5 = t_9t_{10}$, $m_6 = t_{11}t_{12}$, $m_7 = t_{13}t_{14}$, it is inferred in step 6 the t-semiflow $W_1 = m_1m_2m_3m_4m_5m_6m_7$ shown in Figure 1.3.

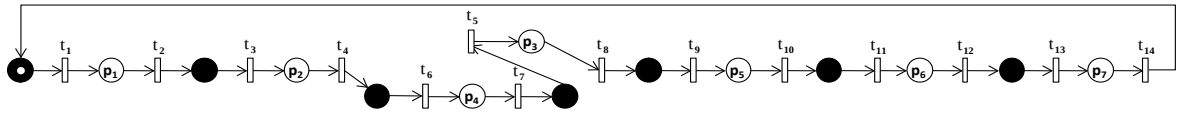


Figure 1.3 t-semiflow inferred $W_1 = m_1m_2m_3m_4m_5m_6m_7$

The arriving m-word $m_1 = t_1t_2$ is the first one of $W_1 = m_1m_2m_3m_4m_5m_6m_7$. Then, it is supposed that W_1 has been completely observed and a new t-semiflow $W_2 = m_1$ is inferred. Observed m-words $m_2 = t_3t_4$, and $m_{3\text{-}4} = t_5t_6t_7t_8$, in step 4 are added to the t-semiflow W_2 and the model is updated in step 6 to allow the firing of all of them, as shown in Figure 1.4.

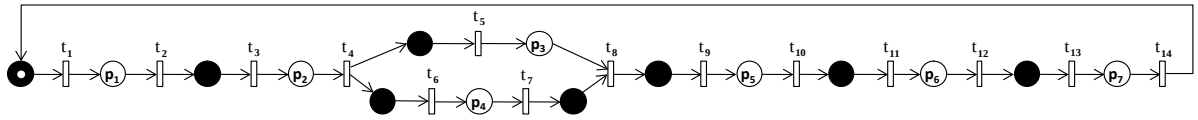


Figure 1.4 Complete t-semiflow $W_1 = m_1m_2m_3m_4m_5m_6m_7$ and inferred t-semiflow $W_2 = m_1m_2m_{3\text{-}4}$

The last m-word $m_7 = t_{13}t_{14}$ is observed. It is made a merging of places to allow the firing of the m-words observed in the order the appeared. A new t-semiflow $W_3 = m_5m_6$ is inferred and t-semiflows $W_1 = m_1m_2m_3m_4m_7$ and $W_2 = m_1m_2m_{3\text{-}4}m_7$ are updated. The final model can be seen in Figure 1.5.

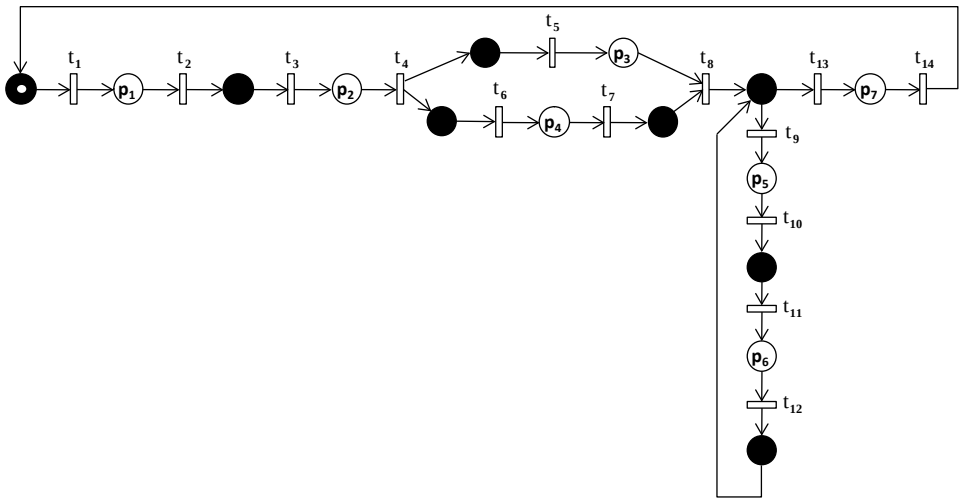


Figure 1.5 t-semiflows $W_1 = m_1m_2m_3m_4m_7$, $W_2 = m_1m_2m_{3\text{-}4}m_7$ and $W_3 = m_5m_6$

The proposed algorithms to update the non observable places have linear complexity on the number of the transitions computed and the m-words detected, that is, in the size of the identified model.

The general algorithm to update a model that includes all the updating procedures of non observable places is also executed in polynomial time.

1.2.2. Parametric automata identification

In [Klein, 2005a] a finite automaton is built form a given set of observed cyclic sequences, containing values of the inputs and outputs values of the system during its normal behaviour. The method was proposed for obtaining models adapted for fault detection in a model-based approach [Roth, 2011][Roth, 2012].

The identification approach proposes to compute a non-deterministic autonomous automaton with output (NDAAO) model describing the behaviour of the unknown DES. The definition of the NDAAO is presented below.

The system to be identified is a compound system controller + plant running in a closed-loop considered as a generator or an information source.

The algorithm receives a set of observed production cycles obtained from the system to be identified. Each observed production cycle or observed sequence is an ordered series of input/output (I/O) binary vectors at different times. The observed sequences do not necessarily have the same length; however, the first and last I/O vectors of different sequences must be identical.

The NDAAO construction principle is to associate each different observed I/O vector with a single state. The transitions between the different states are created after a path between the corresponding I/O vectors has been observed.

First, we present some definitions taken from [Klein, 2005b]. A non-deterministic autonomous automaton with output, denoted NDAAO, is a five-tuple:

NDAAO = (X, Ω, r, λ, x_0)

X	finite set of states,
Ω	output alphabet,
r: $X \rightarrow 2^X$	non deterministic transition relation,
λ: $X \rightarrow \Omega$	output function,
$x_0 \in X$	initial state

Each observed production cycle of the system (also referred to as an observed sequence) is denoted as σ_i and formally defined as $\sigma_i = (u_i(1), u_i(2)... u_i(|\sigma_i|))$ where $u_i(j)$ is an I/O vector and $|\sigma_i|$ represents the length of the considered sequence σ_i.

The cyclic production implies that the first and the last observed I/O vectors of the different sequences are identical. This can be formulated as $\forall(i, j), u_i(1) = u_j(1) = u_i(|\sigma i|) = u_j(|\sigma j|)$. The algorithm proceeds in six steps:

Algorithm 1.2

1. For each observed sequence σ_i, define sequences σ_i^k of k consecutive vectors $u_i(t)$ where k is an a parameter fixed a priori
2. Construct of the NDAAO
3. Rename the output function
4. Reduce the last state
5. Merge equivalent states
6. Closure of the automaton

Example 1.2. Let us consider the example of an elementary plant with a controller having two inputs and one output [Klein, 2005b].

The observed sequences of I/O vectors are:

$$\sigma_1 = \left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \right) \sigma_2 = \left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \right)$$

In order to simplify the notation, each I/O vector is coded as A, B, C, D or E. These letters represent the letters of the observed alphabet. With this coding, the observed sequences are: σ_1 = (A, B, C, D, E, A) and σ_2 = (A, C, B, C, D, A).

Step 1: Construction of vector sequences σ_j^k. Setting k = 2, we obtain for the example:

σ_1^2 = ((A,A),(A,B),(B,C),(C,D),(D,E),(E,A), (A,A))

σ_2^2 = ((A,A),(A,C),(C,B),(B,C),(C,D),(D,A), (A,A))

Step 2: Construction of the NDAAO. The identification principle is to associate each different word with a single state. This step is illustrated by Figure 1.6.

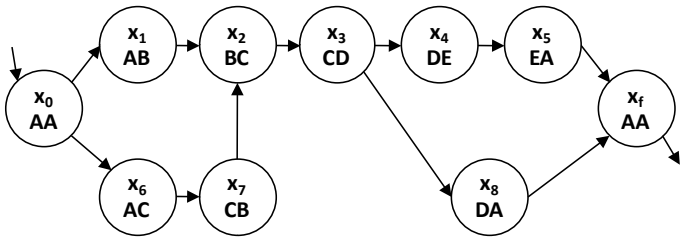


Figure 1.6 Association of words with states of the NDAAO

Step 3: Renaming of the output function. Each state of the NDAAO corresponds to a unique and stable value of the input and output signals. This value is described by the last letter of each sequence of length *k*, as shown in Figure 1.7.

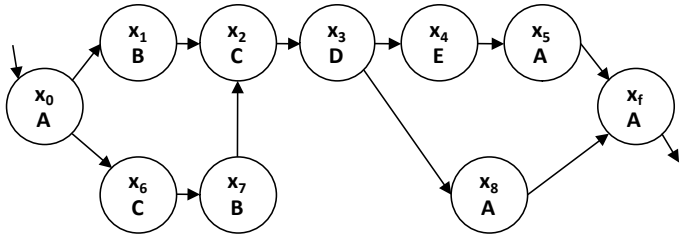


Figure 1.7 Association of states with I/O signals

Step 4: Reduction of the last state. The last *k* states of each branch ending with x_f are associated with the same letter. These states can be reduced with a procedure that has to be iterated *k – 1* times. First, merge the pre-states of x_f. Second, redefine this new state as the final state x_f and delete the former x_f from the set of states. This procedure is illustrated in Figure 1.8.

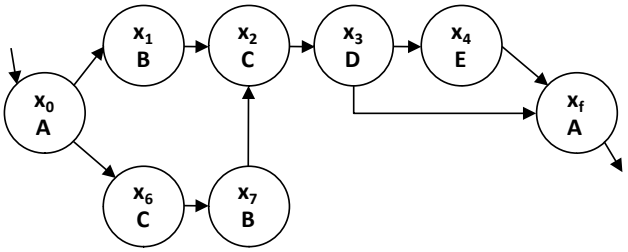


Figure 1.8 Reduction of the last state

Step 5: Merging of equivalent states. Two states are equivalent if and only if:

1. They are associated with the same output
2. They have the same set of post states.

It has been proved that the merging of equivalent states does not affect the languages accepted by the NDAAO. This property can be observed in Figure 1.9.

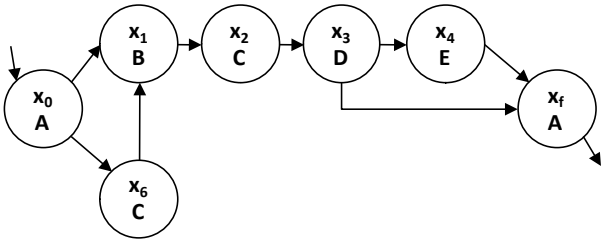


Figure 1.9 Merging of equivalent states

Step 6: Closure of the automaton. With the hypothesis that each observed sequence corresponds to a single production cycle, the states x_0 and x_f of the NDAAO identified are identical. Thus the NDAAO can be closed resulting in a strongly connected NDAAO, as observed in Figure 1.10.

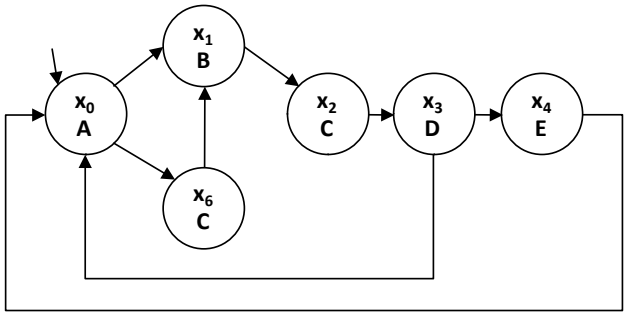


Figure 1.10 Final model

The time required to build different models is very low and does not represent any problems for the application of the identification method. However, the reduction of the NDAAO requires more time than the identification of the model. If new information is available, the time required for the identification of the NDAAO is reduced. However, this gain is not very important since the reduction must be performed again.

1.2.3. Parametric automata distributed identification

As stated before, in many practical applications, a concurrence phenomenon can be observed, which does not allow identifying a suitable model for online fault diagnosis with the [Klein, 2005a] methodology. To overcome this problem, a technique to divide the system in subsystems with converging observed languages has been proposed by [Roth, 2010a], [Roth, 2010b].

The idea is to divide the system into subsystems and to systematically accept a certain amount of unknown combined subsystem behaviour.

The approach is based on the heuristic that a certain amount of unknown global behaviour resulting from a combination of regular subsystem evolutions can often be accepted as fault-free because it is similar to the known fault-free behaviour.

The algorithm receives, similarly to [Klein, 2005a] a set of observed production cycles (cyclic I/O binary vector sequences).

As output, the algorithm returns a set of partial automatons, one for each subsystem in which the observed system has been partitioned. These automatons are later used for fault detection.

The method to automatically perform the partitioning uses an optimization technique to solve the combinatorial problem of assigning controller I/Os to N_{sys} subsystems.

A solution $y(sys_t)$ is a function which assigns a set of controller I/Os to each subsystem. The set $y(sys_t)$ contains the I/Os which are considered in the partial I/O vector of the t-th subsystem. A heuristic optimization approach is used to find an optimal solution. Two optimization criteria related to concurrency are used.

The first one counts for each subsystem the newly observed words in each new system cycle after the first one and multiplies this number with the square root of the according cycle:

$$E_1(y) = \frac{1}{N_{sys}} \sum_{\forall sys_i} \sum_{h=2}^{p} \left(\sqrt{h} \left(\left| W_{Obs,sys_i}^{n,h} \right| - \left| W_{Obs,sys_i}^{n,h-1} \right| \right) \right),$$

where p is the number of observed sequences and $W_{Obs,sys_i}^{n,h}$ is the word set of length n up to the h-th cycle of subsystem sys_i. The term $\sqrt{h}$ was heuristically chosen and increases more weight to new words that occur during last system cycles. By division with the number of subsystems, the optimization criterion is normalized.

The second measure is related to the structure of an automaton identified on the basis of observed system data. Concurrency typically leads to several possible behaviours which are reflected by states with several leaving transitions in the reachability graph of an underlying Petri net. If for a given closed-loop DES an NDAAO is identified with algorithm proposed, the resulting automaton can be seen as an approximation of the reachability graph of a Petri net representing the considered system. Hence, concurrency in the system is represented by states with several leaving transitions.

The branching degree BD of the $NDAAO_{sys_i}$ identified for subsystem sys_i is defined as:

$$BD(NDAAO_{sys_i}) = \sum_{\forall x \in X} \begin{cases} 0 & if\ |r(x)| \le 1 \\ |r(x)| - 1 & if\ |r(x)| > 1 \end{cases}$$

where $r(x)$ is the number of leaving transitions of the state x. Only states with more than one leaving transition contribute to this measure since this represents possible concurrent behaviour.

A measure for the concurrency of a given I/O partitioning y can be calculated by summing up the branching degrees of partial NDAAO identified for the subsystems:

$$E_2(y) = \sum_{\forall sys_i} \left(BD\left(NDAAO_{sys_i} \right) \right)$$

It is possible to have $BD > 0$ and thus $E_2(y) > 0$ although there is no concurrency in the system.

To solve combinatorial problem of partitioning, simulated annealing is used.

Algorithm 1.3: Simulated annealing

Input: Starting temperature T, cooling rate cR and the minimum (stop) temperature $T_{\min}$
Initialize T
Select current solution y_c at random
Repeat
  Select a new solution y_{new}
  If $eval(y_c) > eval(y_{new})$
    Then $y_c \leftarrow y_{new}$
  Else If $random[0,1) < e^{\frac{eval(y_c) - eval(y_{new})}{T}}$
    Then $y_c \leftarrow y_{new}$
  $T \leftarrow T * coolingRate$
Until $T < T_{\min}$

Once the system has been partitioned in subsystems, an NDAAO is constructed as in [Klein, 2005a] for every one of the subsystems, considering only the components of the I/O vector correspondent to each part.

1.2.4. Integer Linear Programming Language identification

In [Giua, 2005] it is presented a technique for identifying a Petri net system, which is able to reproduce a given finite language of transition sequences. Several extensions to this work have been made in [Cabasino, 2006a] [Cabasino, 2006b] [Cabasino, 2006c] [Cabasino, 2009]; such methodology has been proposed to deal with systematic approaches for diagnosis.

This algorithm processes the set of transition sequences that can be fired starting from the initial marking of a Petri net.

An upper bound on the number of places of the net is necessary. The algorithm returns a Petri net which reproduces the given language.

The strategy of the algorithm is to generate an Integer Linear Programming (ILP) problem adding algebraic constraints which force a Petri net to accept the specified language.

To select among all the solutions that satisfy stated constraints, a performance index is minimized, trying to decrease the arcs weights and number of tokens in the initial marking of the Petri net. Some definitions from [Cabasino, 2009] are presented.

Let $\mathscr{L} \subset T^*$ be a finite prefix-closed language, and k be the length of the longest string in $\mathscr{L}$.

A nonnegative integer K is given such that the following condition holds:

$$\max_i M_0(pi) + k \cdot \max_{i,j} Post(i,j) \leq K$$

Given two pairs (σ, t) and (σ', t'), $(\sigma, t) \equiv (\sigma', t')$ if $\pi(\sigma) = \pi(\sigma')$ and $t = t'$, where $\pi(\sigma)$ associates σ to its firing vector.

Let $\mathscr{E}' = \{(\sigma, t_j) \mid \sigma \in \mathscr{L}, |\sigma| < k, \sigma t_j \in \mathscr{L}\}$ and $\mathscr{E} = \mathscr{E}'|_{\equiv}$

Let $\mathscr{D}' = \{(\sigma, t_j) \mid \sigma \in \mathscr{L}, |\sigma| < k, \sigma t_j \notin \mathscr{L}\}$ and $\mathscr{D} = \mathscr{D}'|_{\equiv}$

Algorithm 1.4

A net system is a solution of the identification problem if and only if it satisfies the following set of linear algebraic constraints.

$$\mathscr{G}_m(\mathscr{E}, \mathscr{D}) \stackrel{\Delta}{=} \begin{cases} \mathrm{M}_0 + Post \cdot \pi(\sigma) - Pre \cdot (\pi(\sigma) + \vec{t}_j) \geq \vec{0} & \forall (\sigma, t_j) \in \mathscr{E} \\ -\mathrm{K}\mathrm{S}_{\sigma,j} + \mathrm{M}_0 + Post \cdot \pi(\sigma) - Pre \cdot (\pi(\sigma) + \vec{t}_j) \leq -\vec{1}_m & \forall (\sigma, t_j) \in \mathscr{D} \\ \vec{1}^T \mathrm{S}_{\sigma,j} \leq m - 1 & \forall (\sigma, t_j) \in \mathscr{D} \\ M_0 \in \mathrm{N}^m \\ Pre, Post \in \mathrm{N}^{mxn} \\ \mathrm{S}_{\sigma,j} \in \{0,1\}^m \end{cases}$$

Let $f(M_0, Pre, Post)$ be a given performance index. The solution to the identification problem that minimizes $f(M_0, Pre, Post)$ can be computed by solving the IPP:

$$\begin{cases} \min & f(M_0, Pre, Post) \\ s.t. & \mathscr{G}_m(\mathscr{E}, \mathscr{D}) \end{cases}$$

A typical choice for f is:

$$f(M_0, Pre, Post) = \vec{1}_m^T \cdot M_0 + \vec{1}_m^T \cdot (Pre + Post) \cdot \vec{1}_n$$

The algorithm has been extended in several works, but the basis principle is the same.

Example 1.3. In order to illustrate the method, let us present an example taken from [Cabasino, 2009].

Let $\mathscr{L} = \{\varepsilon, t_1, t_1t_1, t_1t_2, t_1t_1t_2, t_1t_2t_1\}$ and m = 2, thus $k = 3$. Assume that we want to determine the Petri net system that minimizes the sum of initial tokens and all arcs such that $L_3(N, M_0) = \mathscr{L}$. This requires the solution of an IPP where

$\mathscr{E} = \mathscr{E}' = \{(\varepsilon, t_1), (t_1, t_1), (t_1, t_2), (t_1t_2, t_1), (t_1t_1, t_2)\}$ and

$\mathscr{D} = \mathscr{D}' = \{(\varepsilon, t_2), (t_1t_2, t_2), (t_1t_1, t_1)\}$

The procedure identifies a net system with

$$Pre = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, Post = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, M_0 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

The complexity of this approach is exponential with respect to *k*. Also, it is well known that an IPP is an NP-hard problem.

1.2.5. Integer Linear Programming Identification

This approach [Dotoli, 2006a], [Dotoli, 2006b], [Dotoli, 2007], [Dotoli, 2008] is an extension of the works from [Giua, 2005]. However, besides a sequence of events, the available output response sequence of the DES is used to make the inference of a Petri Net model.

The method supposes that all the DES events can be detected, distinguished and not silent.

The algorithm receives a sequence of events with their corresponding output vectors. Also, it is needed an upper bound of the number of non-observable places for the net to identify.

The algorithm returns a Petri net with observable places representing the actuators of the system, non-observable places and labelled transitions representing the observed event sequence. It is also possible that the algorithm returns a 0 (zero) when there is no possible solution of the problem for the given sequence and number of places.

The strategy of the algorithm is to generate an Integer Linear Programming Problem similarly to the [Cabasino, 2009] strategy: a net system is a solution of the identification problem if and only if it satisfies the following set of linear algebraic constraints:

$$\xi(w, Y, \lambda, T, m) = \begin{cases} Pre, Post \in N^{mxn} \\ M_i \in N^m \quad \text{with} \quad i = 0,...,h \\ Post^T \vec{1}_{mx1} + Pre^T \vec{1}_{mx1} \geq \vec{1}_{nx1} \\ Post \vec{1}_{nx1} + Pre \vec{1}_{nx1} \geq \vec{1}_{mx1} \\ \forall t_{\beta i}^{\alpha i} \in \sigma \quad \text{with} \quad \lambda(\sigma) = w, Pre \vec{t}_{\beta i}^{\alpha i} \leq M_{i-1} \\ \forall t_{\beta i}^{\alpha i} \in \sigma \quad \text{with} \quad \lambda(\sigma) = w, (Post - Pre) \vec{t}_{\beta i}^{\alpha i} = M_i - M_{i-1} \end{cases}$$

Some constraints can be added if additional structural properties are known on the PN model to identify. For example, if there is no place without successor transitions it can be added: $Pre \cdot \vec{1}_{nx1} \geq \vec{1}_{mx1}$. If there is no transition without successor places: $Post^T \cdot \vec{1}_{mx1} \geq \vec{1}_{nx1}$. If there are no source transitions: $Pre^T \cdot \vec{1}_{mx1} \geq \vec{1}_{nx1}$.

A performance index is used, an indicator of the PN size, as a linear function.

$$\phi(Pre, Post, M_0) = \vec{a}^T Pre \vec{b} + \vec{c}^T Post \vec{d} + \vec{e} M_0$$

It is presented now the basis of the algorithm that solves the identification problem stated above. The complete algorithm and a best explanation of the solution are given in [Dotoli, 2008].

Algorithm 1.5

1. Initialization of the algorithm variables. The first output vector is obtained and the set of labels, the set of transitions, and the set of output vectors are initialized. Every time more information is calculated, these sets are actualized.

2. Wait until a new event and its corresponding output vector are observed.

3. Associate a transition to an event.

3.1 The event occurs for the first time. A new transition is created and associated to the event and the observed change of marking.

3.2 If the event occurred previously

3.2.1 A new transition must be associated with the event (if there is no change in the marking associated to any transition).

3.2.2 A fired transition is associated to the event.

4. Solve the ILP problem

$$\min \phi(Pre_w, Post_w, M_{0w}) \text{ s.t. } \xi(w, Y, \lambda_w, T_w, m')$$

as many times as necessary, starting with m' equal to the number of observable places and incrementing it, until it is found a solution or until m' is equal to the upper bound of the number of places.

5. Return to the condition of recording the events.

Example 1.4. It is taken from [Dotoli, 2008]. Let us consider a DES with $y \in \mathbb{N}^5$ and $\overline{m} = q = 5$. Assume the initial output is $y_0 = [00102]^T$ and the observed sequence is $w = e_{\alpha_1}, e_{\alpha_2}, e_{\alpha_3}, e_{\alpha_4} = e_1, e_2, e_2, e_1$ with the corresponding outputs $y_1 = [40101]^T$, $y_2 = [31001]^T$, $y_3 = [01011]^T$ and $y_4 = [00102]^T$.

At each event occurrence the identification algorithm is applied, adding constraints to obtain a PN neither without transitions nor places without successors. However, no solution is provided until the occurrence of the last event. The ILP solved is:

Minimize

$$[1 \quad 1 \quad 1 \quad 1 \quad 1](Pre + Post)\begin{bmatrix}1\\1\\1\\1\end{bmatrix} + [1 \quad 1 \quad 1 \quad 1 \quad 1]M_0$$

subject to:

1) $Pre, Post \in N^{m \times n}$

$$Pre = \begin{bmatrix} pre_{11} & pre_{21} & pre_{31} & pre_{41} \\ pre_{12} & pre_{22} & pre_{32} & pre_{42} \\ pre_{13} & pre_{23} & pre_{33} & pre_{43} \\ pre_{14} & pre_{24} & pre_{34} & pre_{44} \\ pre_{15} & pre_{25} & pre_{35} & pre_{45} \end{bmatrix}, Post = \begin{bmatrix} post_{11} & post_{21} & post_{31} & post_{41} \\ post_{12} & post_{22} & post_{32} & post_{42} \\ post_{13} & post_{23} & post_{33} & post_{43} \\ post_{14} & post_{24} & post_{34} & post_{44} \\ post_{15} & post_{25} & post_{35} & post_{45} \end{bmatrix}$$

2) $M_i \in N^m$ with $i = 0, \dots, h$

$$M_0 = \begin{bmatrix} m_{0_1} \\ m_{0_2} \\ m_{0_3} \\ m_{0_4} \\ m_{0_5} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 2 \end{bmatrix}, M_1 = \begin{bmatrix} m_{1_1} \\ m_{1_2} \\ m_{1_3} \\ m_{1_4} \\ m_{1_5} \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}, M_2 = \begin{bmatrix} m_{2_1} \\ m_{2_2} \\ m_{2_3} \\ m_{2_4} \\ m_{2_5} \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, M_3 = \begin{bmatrix} m_{3_1} \\ m_{3_2} \\ m_{3_3} \\ m_{3_4} \\ m_{3_5} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}, M_4 = \begin{bmatrix} m_{4_1} \\ m_{4_2} \\ m_{4_3} \\ m_{4_4} \\ m_{4_5} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 2 \end{bmatrix}$$

3) $Post^T \vec{1}_{m\times 1} + Pre^T \vec{1}_{m\times 1} \geq \vec{1}_{n\times 1}$

$$\begin{bmatrix} post_{11} & post_{12} & post_{13} & post_{14} & post_{15} \\ post_{21} & post_{22} & post_{23} & post_{24} & post_{25} \\ post_{31} & post_{32} & post_{33} & post_{34} & post_{35} \\ post_{41} & post_{42} & post_{43} & post_{44} & post_{45} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} pre_{11} & pre_{12} & pre_{13} & pre_{14} & pre_{15} \\ pre_{21} & pre_{22} & pre_{23} & pre_{24} & pre_{25} \\ pre_{31} & pre_{32} & pre_{33} & pre_{34} & pre_{35} \\ pre_{41} & pre_{42} & pre_{43} & pre_{44} & pre_{45} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \geq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

4) $Post\vec{1}_{n\times 1} + Pre\vec{1}_{n\times 1} \geq \vec{1}_{m\times 1}$

$$\begin{bmatrix} post_{11} & post_{21} & post_{31} & post_{41} \\ post_{12} & post_{22} & post_{32} & post_{42} \\ post_{13} & post_{23} & post_{33} & post_{43} \\ post_{14} & post_{24} & post_{34} & post_{44} \\ post_{15} & post_{25} & post_{35} & post_{45} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} pre_{11} & pre_{21} & pre_{31} & pre_{41} \\ pre_{12} & pre_{22} & pre_{32} & pre_{42} \\ pre_{13} & pre_{23} & pre_{33} & pre_{43} \\ pre_{14} & pre_{24} & pre_{34} & pre_{44} \\ pre_{15} & pre_{25} & pre_{35} & pre_{45} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \geq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

5) $\forall t_{\beta_i}^{\alpha_i} \in \sigma$ with $\lambda(\sigma) = w$, $Pre\vec{t}_{\beta_i}^{\alpha_i} \leq M_{i-1}$

$$\begin{bmatrix} pre_{11} & pre_{21} & pre_{31} & pre_{41} \\ pre_{12} & pre_{22} & pre_{32} & pre_{42} \\ pre_{13} & pre_{23} & pre_{33} & pre_{43} \\ pre_{14} & pre_{24} & pre_{34} & pre_{44} \\ pre_{15} & pre_{25} & pre_{35} & pre_{45} \end{bmatrix} \begin{bmatrix} t_{\beta_1 1}^{\alpha_1} \\ t_{\beta_1 2}^{\alpha_1} \\ t_{\beta_1 3}^{\alpha_1} \\ t_{\beta_1 4}^{\alpha_1} \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 2 \end{bmatrix}, \begin{bmatrix} pre_{11} & pre_{21} & pre_{31} & pre_{41} \\ pre_{12} & pre_{22} & pre_{32} & pre_{42} \\ pre_{13} & pre_{23} & pre_{33} & pre_{43} \\ pre_{14} & pre_{24} & pre_{34} & pre_{44} \\ pre_{15} & pre_{25} & pre_{35} & pre_{45} \end{bmatrix} \begin{bmatrix} t_{\beta_2 1}^{\alpha_2} \\ t_{\beta_2 2}^{\alpha_2} \\ t_{\beta_2 3}^{\alpha_2} \\ t_{\beta_2 4}^{\alpha_2} \end{bmatrix} \leq \begin{bmatrix} 4 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix},$$

$$\begin{bmatrix} pre_{11} & pre_{21} & pre_{31} & pre_{41} \\ pre_{12} & pre_{22} & pre_{32} & pre_{42} \\ pre_{13} & pre_{23} & pre_{33} & pre_{43} \\ pre_{14} & pre_{24} & pre_{34} & pre_{44} \\ pre_{15} & pre_{25} & pre_{35} & pre_{45} \end{bmatrix} \begin{bmatrix} t_{\beta_3 1}^{\alpha_3} \\ t_{\beta_3 2}^{\alpha_3} \\ t_{\beta_3 3}^{\alpha_3} \\ t_{\beta_3 4}^{\alpha_3} \end{bmatrix} \leq \begin{bmatrix} 3 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} pre_{11} & pre_{21} & pre_{31} & pre_{41} \\ pre_{12} & pre_{22} & pre_{32} & pre_{42} \\ pre_{13} & pre_{23} & pre_{33} & pre_{43} \\ pre_{14} & pre_{24} & pre_{34} & pre_{44} \\ pre_{15} & pre_{25} & pre_{35} & pre_{45} \end{bmatrix} \begin{bmatrix} t_{\beta_4 1}^{\alpha_4} \\ t_{\beta_4 2}^{\alpha_4} \\ t_{\beta_4 3}^{\alpha_4} \\ t_{\beta_4 4}^{\alpha_4} \end{bmatrix} \leq \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

6) $\forall t_{\beta_i}^{\alpha_i} \in \sigma$ with $\lambda(\sigma) = w$, $(Post - Pre)\vec{t}_{\beta_i}^{\alpha_i} = M_i - M_{i-1}$

$$\left(\begin{bmatrix} post_{11} & post_{21} & post_{31} & post_{41} \\ post_{12} & post_{22} & post_{32} & post_{42} \\ post_{13} & post_{23} & post_{33} & post_{43} \\ post_{14} & post_{24} & post_{34} & post_{44} \\ post_{15} & post_{25} & post_{35} & post_{45} \end{bmatrix} - \begin{bmatrix} pre_{11} & pre_{21} & pre_{31} & pre_{41} \\ pre_{12} & pre_{22} & pre_{32} & pre_{42} \\ pre_{13} & pre_{23} & pre_{33} & pre_{43} \\ pre_{14} & pre_{24} & pre_{34} & pre_{44} \\ pre_{15} & pre_{25} & pre_{35} & pre_{45} \end{bmatrix} \right) \begin{bmatrix} t_{\beta_1 1}^{\alpha_1} \\ t_{\beta_1 2}^{\alpha_1} \\ t_{\beta_1 3}^{\alpha_1} \\ t_{\beta_1 4}^{\alpha_1} \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 2 \end{bmatrix},$$

$$\left(\begin{bmatrix} post_{11} & post_{21} & post_{31} & post_{41} \\ post_{12} & post_{22} & post_{32} & post_{42} \\ post_{13} & post_{23} & post_{33} & post_{43} \\ post_{14} & post_{24} & post_{34} & post_{44} \\ post_{15} & post_{25} & post_{35} & post_{45} \end{bmatrix} - \begin{bmatrix} pre_{11} & pre_{21} & pre_{31} & pre_{41} \\ pre_{12} & pre_{22} & pre_{32} & pre_{42} \\ pre_{13} & pre_{23} & pre_{33} & pre_{43} \\ pre_{14} & pre_{24} & pre_{34} & pre_{44} \\ pre_{15} & pre_{25} & pre_{35} & pre_{45} \end{bmatrix} \right) \begin{bmatrix} t_{\beta_2 1}^{\alpha_2} \\ t_{\beta_2 2}^{\alpha_2} \\ t_{\beta_2 3}^{\alpha_2} \\ t_{\beta_2 4}^{\alpha_2} \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 4 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix},$$

$$\left(\begin{bmatrix} post_{11} & post_{21} & post_{31} & post_{41} \\ post_{12} & post_{22} & post_{32} & post_{42} \\ post_{13} & post_{23} & post_{33} & post_{43} \\ post_{14} & post_{24} & post_{34} & post_{44} \\ post_{15} & post_{25} & post_{35} & post_{45} \end{bmatrix} - \begin{bmatrix} pre_{11} & pre_{21} & pre_{31} & pre_{41} \\ pre_{12} & pre_{22} & pre_{32} & pre_{42} \\ pre_{13} & pre_{23} & pre_{33} & pre_{43} \\ pre_{14} & pre_{24} & pre_{34} & pre_{44} \\ pre_{15} & pre_{25} & pre_{35} & pre_{45} \end{bmatrix} \right) \begin{bmatrix} t_{\beta_3 1}^{\alpha_3} \\ t_{\beta_3 2}^{\alpha_3} \\ t_{\beta_3 3}^{\alpha_3} \\ t_{\beta_3 4}^{\alpha_3} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 3 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

$$\left(\begin{bmatrix} post_{11} & post_{21} & post_{31} & post_{41} \\ post_{12} & post_{22} & post_{32} & post_{42} \\ post_{13} & post_{23} & post_{33} & post_{43} \\ post_{14} & post_{24} & post_{34} & post_{44} \\ post_{15} & post_{25} & post_{35} & post_{45} \end{bmatrix} - \begin{bmatrix} pre_{11} & pre_{21} & pre_{31} & pre_{41} \\ pre_{12} & pre_{22} & pre_{32} & pre_{42} \\ pre_{13} & pre_{23} & pre_{33} & pre_{43} \\ pre_{14} & pre_{24} & pre_{34} & pre_{44} \\ pre_{15} & pre_{25} & pre_{35} & pre_{45} \end{bmatrix}\right) \begin{bmatrix} t_{\beta_4 1}^{\alpha_4} \\ t_{\beta_4 2}^{\alpha_4} \\ t_{\beta_4 3}^{\alpha_4} \\ t_{\beta_4 4}^{\alpha_4} \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 2 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}.$$

The IPN obtained is illustrated in Figure 1.11.

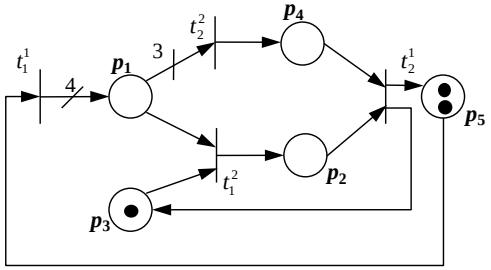


Figure 1.11 Solution for identification problem of *Example 1.4*

In the worst case the number of unknowns is linear with the number of places, of transitions, and with the length h of the firing sequence. In the cases examined by the authors, an optimal solution is obtained in a short time implementing and solving the ILP problem on a PC equipped with a standard solver of optimization problems, for example GLPK. However, in order to apply the algorithm online, the dynamics of the DES has to be slow with respect to the time required to solve the ILP problem at each occurrence.

1.2.6. Neural Networks approach

The work in [Ould El Medhi, 2006] focuses on the construction of a Petri net to represent a sequence of observed events. The identified model is used for reliability issues.

The algorithm receives a sequence of events stored during the operation of the system. A technique to include timing information in the form of a stochastic Petri net is presented, but we do not consider such a technique, since time is out of the scope of this work.

The identification strategy is the construction of a propagation relation matrix, which contains the set of preceding events information. From such a matrix, two perspectives to construct the Petri net are presented. One of them is based on the addition of places to the net in order to link consecutive events. Another one considers a learning process by neural networks.

Other work on identification of the same author has been published in [Ould El Medhi, 2012], but the technique therein presented is quite similar to the ILP methodology by [Giua, 2005].

Algorithm 1.6

The events set $E = \{e_1, e_2, \ldots, e_q\}$ is known. From observation of the system, an event sequence $SeqE = (e(k))_{1 \leq k \leq K}, e(k) \in E$ is constructed and is the basis of the synthesis algorithm.

From *SeqE*, a rank-1 propagation relation matrix $B = (b_{ij}) \in \{0, 1\}^{q \times q}$ is constructed such that $b_{ij} = 1$ if e_j, e_i are consecutive events in *SeqE*.

An event graph is constructed, such that if $b_{ij} = 1$, it is created a place n to accomplish Pre(n, i) ← 1 and Post(n,j) ← 1.

The minimal initial marking is computed setting $M(K) = 0$ and determining recurrently $M(k\text{-}1) = \max(0, M(k) - C \bullet X(k))$, where $X(k)$ represents the firing vector of transition $T(e(k))$.

An option to reduce the number of places is to reuse the output place p_i if $b_{ij} = 1$ and p_i has already been created to accomplish Pre(p_i, i)← 1 and Post(p_i,j)←1

The last method is the Learning by Multilayer Neural Network. A supervised learning method is used to build a minimal Petri Net model (in number of places or transitions).

The basic idea is to consider PN as a multi layered neural network (Figure 1.12): the hidden layer is made of *n* places and the input and output layers both correspond to the *p* transitions. The weight matrix *Q* between input and hidden layers corresponds to the connection from transitions to places (i.e. post-incidence matrix *Post*). The weight matrix between hidden and output layers *V* corresponds to pre-incidence matrix *Pre*.

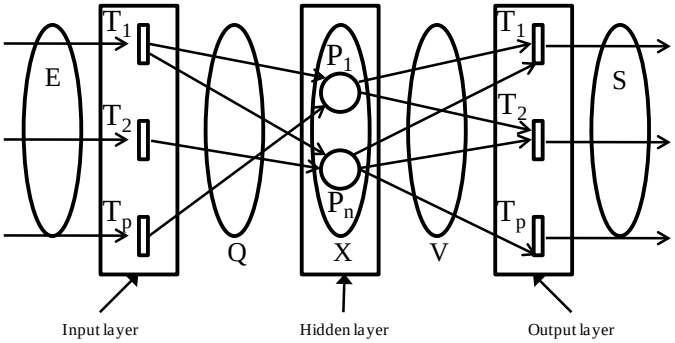


Figure 1.12 Petri net model structure construction

The output *Y* is computed by

$$Z = Q * X$$

$$Y = V * Z$$

The goal is to learn B from *SeqE*. The network is trained by couples (x_k, y_k), $k = 1,\ldots,ne$ such that the vectors for the input layer $\{x_k\}$, $k = 1,\ldots, q$ are the Parikh vectors associated with events e_k and the vectors for the output layer $\{yd_k\}$, $k = 1, \ldots, q$ are the columns of the B matrix.

For every input vector x_k, the output y_k is compared to the desired output yd_k. The objective is to reduce the error to a given threshold:

$$\varepsilon = \frac{1}{2q^2} \sum_{i=1}^{ns} \sum_{b=1}^{nt} (y_{ib} - yd_{ib})^2$$

This is accomplished by modifying *Q* and *V*:

$$\Delta v_{ij} = -\eta \frac{\partial \varepsilon}{\partial v_{ij}} = -\eta \sum_{k=1}^{nt} (y_{ik} - yd_{ik}) * z_{kj}$$

$$\Delta q_{lm} = -\eta \frac{\partial \varepsilon}{\partial q_{lm}} = -\eta \sum_{k=1}^{ns} v_{il} \sum_{k=1}^{nt} (y_{ik} - yd_{ik}) * x_{km}$$

Once the error is stabilized or once a maximal iterations number is reached, the entries in matrices *Q* and *V* are converted into binary values:

$$Br(x) = 0 \quad if \quad x < 0.5$$
$$Br(x) = 1 \quad if \quad x \geq 0.5$$

Then the Pre and Post matrices are computed:

$$\text{Post} = Br(Q)$$
$$\text{Pre} = (Br(V))^T$$

The error is computed once again. If the error is lower than a threshold, the process stops. Otherwise, a new adaptation phase is started, using the integer obtained values.

The initial number of places is *q*. Once the algorithm converges to an acceptable model, learning is restarted with a lower value. The minimal size PN is the last one obtained for which the convergence is assured.

Example 1.5. From a sequence *SeqE*, a propagation relation matrix is constructed:

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

From such a matrix, an event graph (Figure 1.13) is constructed.

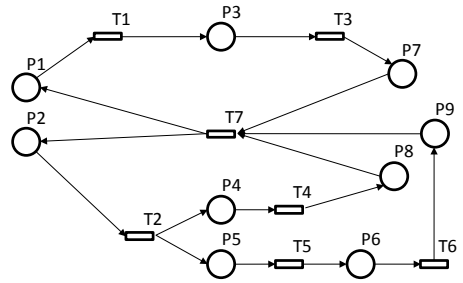


Figure 1.13 Event graph model constructed

With the technique of re-used places, the model is simplified, as shown in Figure 1.14. However, as can be seen, the reduced model is not blocking free.

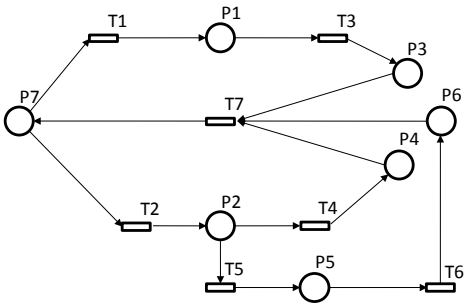


Figure 1.14 Petri net model obtained with reusing technique

The learning algorithm is run with $n = p = 7$ hidden nodes and according to $\eta = 0.01$, *error_threshold* = 0.00001, and *limit_epo* = *limit_ite* = 1000. Once the convergence obtained, algorithm runs again with 6 and then with 5 hidden nodes. Each time, the

algorithm does succeed. No convergence is obtained with $n < 5$ nodes. Error stays constant to a non zero value. The minimal size PN model is showed in Figure 1.15.

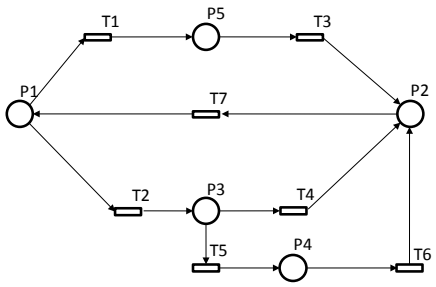


Figure 1.15 Petri net model obtained by the learning algorithm

1.2.7. Parametric interpreted Petri net identification

In [Estrada, 2009], a method for building interpreted Petri net models from observations of DES's inputs and outputs is proposed. The identification method consists of several stages that build systematically an IPN model from input-output sequences representing the external behaviour of partially observable DES. A software tool based on the presented algorithms has been developed; it processes a set of input-output vector sequences yielding the drawing of the computed IPN model.

The input data to the identification procedure is a set of input-output words that may include cyclic behaviour. Based on an accuracy parameter κ, the aim of the identification process is to obtain a safe IPN model (Q, M_0) such that $\pounds_{out}^{\kappa}(Q, M_0) = \mathcal{L}^{\kappa}(S)$. The parameter κ is used to adjust the accuracy of the identified model.

From the input-output vector words, event sequences are computed and then sequences of event substrings of length κ are built. Then every substring is associated to a transition of a PN, which describes the relation precedence among the event substrings; this PN is formed by non-observable places. A transformation method based on concurrence transformations is performed. Finally, output changes provoked by events are described by marking changes in observable places and then related to pertinent transitions in the PN; input changes are associated to such transitions. Implicit non-observable places are deleted.

We can summarise the stages of the method for IPN model identification as follows.

Algorithm 1.7

Input: A DES and an accuracy parameter κ

Output: (Q,M_0): an IPN model

1. Obtain the input symbols and the cyclic sequences of observed output vectors.
2. Compute the event sequences from the observed vectors.
3. For every sequence of events, create traces of length κ.
4. Create the non-observable behaviour of the IPN and simplify it.
5. Complete the Petri net adding the observed behaviour and delete implicit places.

Example 1.6. Consider the next illustrative example taken from [Estrada, 2010b]. It consists on a DES with n = 4 output signals, Φ = {A, B, C, D}, and m = 3 input signals Σ = {a, b, c}. Three I/O sequences have been observed.

Vector entries correspond to distribution [a b c | A B C D]

$$w_1 = \left(\begin{bmatrix}0\\0\\0\\\underline{1}\\0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\\underline{0}\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\\underline{0}\\0\\1\\0\\0\end{bmatrix}\right), w_2 = \left(\begin{bmatrix}0\\0\\0\\\underline{1}\\0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\\underline{0}\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\1\\\underline{1}\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\\underline{1}\\0\\0\\1\\1\end{bmatrix}, \begin{bmatrix}1\\0\\\underline{0}\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\\underline{1}\\0\\0\\0\\0\end{bmatrix}\right), w_3 = \left(\begin{bmatrix}0\\0\\0\\\underline{1}\\0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\\underline{0}\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\1\\\underline{1}\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\\underline{0}\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\\underline{0}\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\\underline{1}\\0\\0\\0\\0\end{bmatrix}\right)$$

The sequences τ_i of the detected event vectors e_j associated to I/O changes are obtained:

$$w_1 = \left(\begin{bmatrix}0\\0\\0\\\underline{1}\\0\\0\\0\end{bmatrix}\begin{bmatrix}\overset{e_1}{\rightarrow}\\1\\0\\\underline{0}\\-1\\0\\0\end{bmatrix}\begin{bmatrix}1\\0\\\underline{0}\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}\overset{e_2}{\rightarrow}\\0\\0\\\underline{0}\\1\\0\\0\end{bmatrix}\begin{bmatrix}1\\0\\\underline{0}\\0\\1\\0\\0\end{bmatrix}\right),$$

$\tau_1 = e_1e_2$, $\lambda'(e_1) = a\_1$, $\lambda'(e_2) = \varepsilon$

$$w_2 = \left(\begin{bmatrix}0\\0\\0\\\underline{1}\\0\\0\\0\end{bmatrix}\begin{bmatrix}\overset{e_1}{\rightarrow}\\1\\0\\\underline{0}\\-1\\0\\0\end{bmatrix}\begin{bmatrix}1\\0\\\underline{0}\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}\overset{e_3}{\rightarrow}\\0\\1\\\underline{1}\\0\\0\\1\end{bmatrix}\begin{bmatrix}1\\1\\\underline{1}\\0\\0\\1\\0\end{bmatrix}\begin{bmatrix}\overset{e_4}{\rightarrow}\\0\\-1\\\underline{0}\\0\\0\\1\end{bmatrix}\begin{bmatrix}1\\0\\\underline{1}\\0\\0\\1\\1\end{bmatrix}\begin{bmatrix}\overset{e_5}{\rightarrow}\\0\\0\\\underline{-1}\\0\\0\\-1\end{bmatrix}\begin{bmatrix}1\\0\\\underline{0}\\0\\0\\0\\1\end{bmatrix}\begin{bmatrix}\overset{e_6}{\rightarrow}\\-1\\0\\\underline{0}\\1\\0\\0\\-1\end{bmatrix}\begin{bmatrix}0\\0\\\underline{1}\\0\\0\\0\\0\end{bmatrix}\right),$$

$\tau_2 = e_1e_3e_4e_5e_6$, $\lambda'(e_3) = b\_1\ c\_1$, $\lambda'(e_4) = b\_0$, $\lambda'(e_5) = c\_0$, $\lambda'(e_6) = a\_0$

$$w_3 = \left(\begin{bmatrix}0\\0\\0\\\underline{1}\\0\\0\\0\end{bmatrix}\begin{bmatrix}\overset{e_1}{\rightarrow}\\1\\0\\\underline{0}\\-1\\0\\0\end{bmatrix}\begin{bmatrix}1\\0\\\underline{0}\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}\overset{e_3}{\rightarrow}\\0\\1\\\underline{1}\\0\\0\\1\end{bmatrix}\begin{bmatrix}1\\1\\\underline{1}\\0\\0\\1\\0\end{bmatrix}\begin{bmatrix}\overset{e_5}{\rightarrow}\\0\\0\\\underline{-1}\\0\\0\\-1\end{bmatrix}\begin{bmatrix}1\\1\\\underline{0}\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}\overset{e_4}{\rightarrow}\\0\\-1\\\underline{0}\\0\\0\\1\end{bmatrix}\begin{bmatrix}1\\0\\\underline{0}\\0\\0\\0\\1\end{bmatrix}\begin{bmatrix}\overset{e_6}{\rightarrow}\\-1\\0\\\underline{0}\\1\\0\\0\\-1\end{bmatrix}\begin{bmatrix}0\\0\\\underline{1}\\0\\0\\0\\0\end{bmatrix}\right),$$

$\tau_3 = e_1e_3e_4e_4e_6$

The sequences of traces τ_i^Kusing $\kappa = 2$ are:

$\tau_1^2 = \varepsilon e_1, e_1e_2$ for τ_1

$\tau_2^2 = \varepsilon e_1, e_1e_3, e_3e_4, e_4e_5, e_5e_6$ for τ_2

$\tau_3^2 = \varepsilon e_1, e_1e_3, e_3e_5, e_5e_4, e_4e_6$ for τ_3

The obtained IPN corresponding to the three sequences of event vector traces of the example 1 is shown in Figure 1.16. Notice that one of the sequences is not cyclic. However, all sequences start with the same I/O vector.

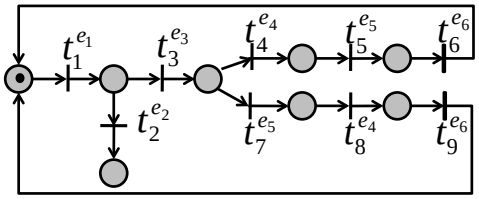


Figure 1.16 Basic model with sequences of event vector traces

The application of simplification and concurrence transformations leads to the model in Figure 1.17.

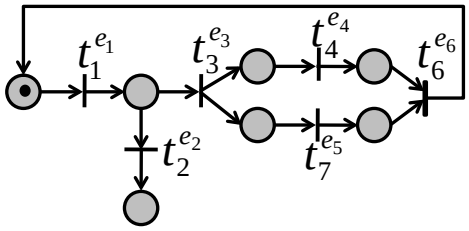


Figure 1.17 Simplified basic model

After adding input and output information, the obtained model is shown in Figure 1.18.

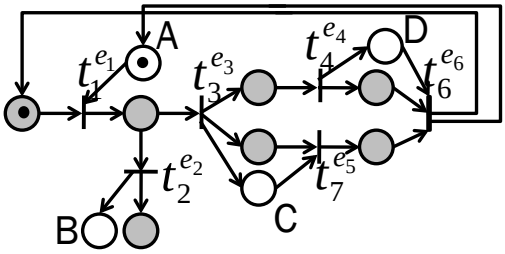


Figure 1.18 IPN model including measurable places

The final model obtained after deleting implicit places and adding the input information is shown in Figure 1.19.

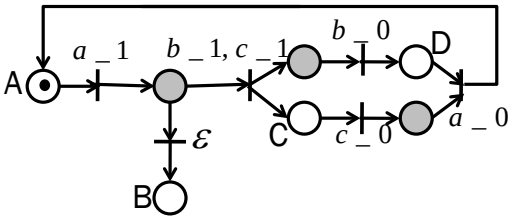


Figure 1.19 Simplified IPN model

1.3. Process mining approaches

Process mining is a research area that can be considered as similar to system identification: it consists on discovering behavioural models of the processes that capture the structured orderings of activities in a workflow. Activity logs are observed as a trace of events being produced by a black-box system. Automated techniques which seek to mine logs to discover information have been developed. We consider here two representative techniques.

1.3.1. Probabilistic workflow mining

The goal of [Cook, 2004] is to identify gross patterns of a workflow behaviour that can be useful for understanding the system functioning. Statistical and probabilistic analyses are used to determine when concurrent behaviour is occurring, and to identify dependence relationships among observed events.

As input, this technique considers a sequence of events *S* (called event trace) characterizing activities that have occurred in a workflow system. The goal of the method is not necessarily to reconstruct a model representing system's behaviour, but rather to identify patterns of behaviour that can be useful to understand how the system works. Individual threads and their individual behaviour are discovered. Also, the points where threads interact are located.

Algorithm 1.8

First, some definitions taken from [Cook, 2004] are introduced. As stated before, a sequence of events is considered. Two sequence constructors are defined:

- Prefix(*S*) is the sequence *S* with the last event removed. If the length of *S* is 1 (i.e., a single event), then Prefix(*S*) = null.
- *S* : *e* is the sequence *S* concatenated with event *e*

Occur(*S*) is the number of occurrences of sequence *S*. For example, *Occur*(AC) = 2 in *S*=CABCBACABCABCBAC. *Occur*(*null*) is the total number of events in the event trace.

The conditional probability of occurrence of a sequence (called *P*(*S*)) is:

$$\text{CondProb}(S) = P(S) = \frac{Occur(S)}{Occur(\text{Prefix}(S))}$$

Frequency tables of N-length sequences can be constructed, providing the (observed) conditional probability that the last event follows the preceding N - 1 events.

Four metrics are computed:

- Entropy. Gives a measure of the randomness in each event sequence and its occurrences:

  $$Entropy(S) = -\sum_{e \in E} P(S:e) \bullet \log_{|E|}(P(S:e))$$

  where *E* is the set of events.
  The closer an entropy value is to $log_N(T)$, the more likely it is to be signifying a T-way fork behaviour. The same metric can apply to joins by viewing the trace backwards.
- Event type counts. When an event has several successors, counting event types helps to make a decision as to whether the behaviour at this point is sequential or concurrent
  If there is a selection:

  $$Occur(S) = \sum_{e \in E, Occur(S:e)>0} Occur(e)$$

  If concurrency is occurring:

  $$\forall (e \in E, Occur(S:e) > 0), Occur(e) = Occur(S)$$

  and

  $$T \times Occur(S) = \sum_{e \in E, Occur(S:e)>0} Occur(e)$$

- Causality. If two events have already been eliminated as forks, joins or synchronization points, to decide when they are sequentially causally related and when they are not. If AB and BA have been observed:
  - If P(AB) + P(BA) ≥ 1.5, they are likely causally related in a two-event loop
  - If P(AB) + P(BA) < 1.5, they are likely independent
- Periodicity. This measure can help to find synchronization points in the process.

Position(S,i) is the position of the last event of the ith occurrence of S. The average period of a sequence is:

$$PeriodMean(S) = \frac{\sum_{i=2}^{Occur(S)} Position(S,i) - Position(S,i-1)}{Occur(S) - 1}$$

Event types with the lowest standard deviations should be event types marking the synchronization points in the process:

$$\text{PeriodStdDev}(S) = \sqrt{\frac{\sum_{i=2}^{\text{Occur}(S)} [\text{Dsq}(S,i) - \text{PeriodMean}(S)^2]}{\text{Occur}(S) - 2}}$$

Based on this four metrics, dependencies that explain as much of the event stream as possible are found. A ranking on the quality of information is defined in [Cook, 2004].

Example 1. 7. Consider the Petri net in Figure 1.20.

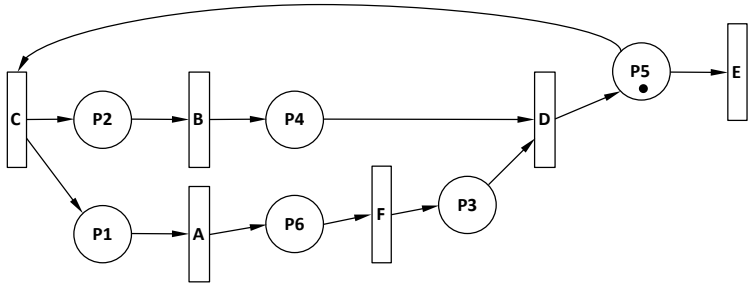


Figure 1.20 Petri net

From a 1666-event stream produced from a stochastic simulation of such a system, the Table 1.1 has been constructed.

	A	B	C	D	E	F
A	0.00	0.25	0.00	0.00	0.00	0.75
B	0.44	0.00	0.00	0.31	0.00	0.25
C	0.56	0.44	0.00	0.00	0.00	0.00
D	0.00	0.00	0.99	0.00	0.01	0.00
E	0.00	0.00	0.00	0.00	0.00	0.00
F	0.00	0.31	0.00	0.69	0.00	0.00

Table 1.1 Conditional probability of length-2 sequences

After computing defined metrics and ranking information, the model in Figure 1.21 has been reconstructed. Thicker arrows are stronger dependencies (they are inferred first).

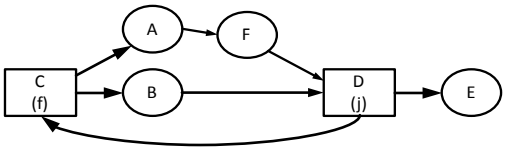


Figure 1.21 Reconstructed model for net in Figure 1.20

1.3.2. Alfa-algorithm

In [van der Aalst, 2004] the workflow mining problem is considered. Since the modelling of a workflow is a difficult task, techniques to do this automatically are required. The term process mining is used for the method of distilling a structured process description from a set of real executions.

The input of the algorithm is a workflow log of several workflow instances composed by several tasks. Workflow instances have been recorded sequentially, even if tasks may be executed in parallel.

Based on the information in the workflow log and by making some assumptions about completeness of the log, a process model in the form of a workflow net is deduced.

In order to find the workflow model, causal dependencies are searched in the workflow log. Such dependencies are used by a so called α-algorithm, which constructs the net that represents the observed workflow instances.

In order to explain the construction of the model, some definitions must be introduced.

Let T be a set of tasks. $\sigma \in T^*$ is a workflow trace and $W \in \mathrm{P}(T^*)$ is a workflow log.

Let W be a workflow log over T. Let $a, b \in T$:

- $a >_W b$ (b directly follows a) if and only if (iff) there is a trace $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ and $i \in \{1,\ldots,n\text{-}2\}$ such that $\sigma \in W$ and $t_i = a$ and $t_{i+1} = b$,
- $a \rightarrow_W b$ (a and b are in a causal relation)iff $a >_W b$ and $b \not>_W a$ (a is not observed before b),
- $a \#_W b$ (a and b never follow each other directly) iff $a \not>_W b$ and $b \not>_W a$,
- $a \|_W b$ (a and b are potentially parallel) iff $a >_W b$ and $b >_W a$.

Let A be a set, $a \in A$, and $\sigma = a_1 a_2 \ldots a_n \in A^*$ a sequence over A of length n:

- $a \in \sigma$ iff $a \in \{a_1, a_2, \ldots, a_n\}$,
- $first(\sigma) = a_1$, if $n \geq 1$,
- $last(\sigma) = a_n$, if $n \geq 1$.

Algorithm 1.9: Mining algorithm α

Let W be a workflow log over T. $\alpha(W)$ is defined as follows:

1. $T_W = \{t \in T \mid \exists \sigma \in W, t \in \sigma\}$
2. $T_I = \{t \in T \mid \exists \sigma \in W, t = first(\sigma)\}$
3. $T_O = \{t \in T \mid \exists \sigma \in W, t = last(\sigma)\}$
4. $$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \\ \wedge \forall a \in A \forall b \in B, a \rightarrow_W b \\ \wedge \forall a_1, a_2 \in A, a_1 \#_W a_2 \\ \wedge \forall b_1, b_2 \in B, b_1 \#_W b_2\}$$
5. $$Y_W = \{(A,B) \in X_W \mid \forall (A',B') \in X_W, A \subseteq A' \\ \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$$
6. $P_W = \{p(A,B) \mid (A,B) \in Y_W\} \cup \{i_W, o_W\}$
7.

$$F_W = \{(a, p_{(A,B)}) \mid (A,B) \in Y_W \wedge a \in A\}$$
$$\cup (p_{(A,B)}, b) \mid (A,B) \in Y_W \wedge b \in B\}$$
$$\cup (i_W, t) \mid t \in T_I\} \cup (t, o_W) \mid t \in T_O\}$$

8. $\alpha(W) = (P_W, T_W, F_W)$.

Example 1.8. Consider the workflow log in Table 1.2.

Case identifier	Task identifier
Case 1	Task A
Case 2	Task A
Case 3	Task A
Case 3	Task B
Case 1	Task B
Case 1	Task C
Case 2	Task C
Case 4	Task A
Case 2	Task B
Case 2	Task D
Case 5	Task A
Case 4	Task C
Case 1	Task D
Case 3	Task C
Case 3	Task D
Case 4	Task B
Case 5	Task E
Case 5	Task D
Case 4	Task D

Table 1.2 Workflow log

It can be seen as $W = \{ABCD, ACBD, AED\}$.

The α-algorithm proceeds as follows

1. $T_W = \{A, B, C, D, E\}$
2. $T_I = \{A\}$
3. $T_O = \{D\}$
4. $$X_W = \{(\{A\},\{B\}),(\{A\},\{C\}),(\{A\},\{E\}),$$
$$(\{B\},\{D\}),(\{C\},\{D\}),(\{E\},\{D\}),$$
$$(\{A\},\{B,E\}),(\{A\},\{C,E\}),(\{B,E\},\{D\}),$$
$$(\{C,E\},\{D\})\}$$
5. $$Y_W = \{(\{A\},\{B,E\}),(\{A\},\{C,E\}),$$
$$(\{B,E\},\{D\}),(\{C,E\},\{D\})\}$$
6. $$P_W = \{i_W, o_W, p_{(\{A\},\{B,E\})}, p_{(\{A\},\{C,E\})}$$
$$p_{(\{B,E\},\{D\})}, p_{(\{C,E\},\{D\})}\}$$
7.

$$F_W = \{(i_W, A), (A, p_{(\{A\},\{B,E\})}) (p_{(\{A\},\{B,E\})}), B..., (D, o_W)\}$$

8. $\alpha(W) = (P_W, T_W, F_W)$.

The result is illustrated in Figure 1.22.

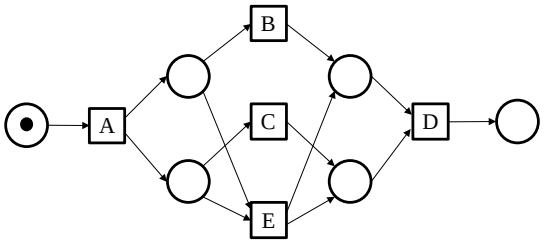


Figure 1.22 Obtained model for the workflow log

1.4. Discussion

Methods characteristics

For comparison purposes, we analyze the different approaches considering several features; some of them have already been considered in [Klein, 2005b]; some others are added to have a more complete scope during comparative analysis.

These features are structured into 4 categories: those characterizing the DES to be identified, those describing the identification process, those qualifying the identified model, and those considering general algorithm features.

DES characteristics

- **Type of inputs/outputs.** In the general case, inputs and outputs of DES to be identified are discrete (they can take a finite number of values). If all inputs and outputs can only take two values (on/off), the DES is called logic.
- **Iterative behaviour.** A DES is called cyclic if it iteratively reaches the initial state during its operation. If it iterates on the same behaviour revisiting a state that is not the initial one it is called repetitive.

Identification process characteristics

- **A-priori information.** If there is no available knowledge about the DES other than its inputs and outputs evolution, the identification is absolute (commonly called black-box). Otherwise, the identification is relative.
- **Model updating.** When the model updating is incremental, the method progressively updates the model from observed information; otherwise, the identification procedure is global: it must be executed on the whole of the observed sequences every time new sequences are collected.

Identified model characteristics

- **Concurrency.** This feature considers if the obtained model can represent explicitly concurrent behaviour observed from the system.
- **Accuracy.** This term is related with completeness of the identified model. If this model represents exactly the observed behaviour, it is complete.

Algorithm characteristics

- **Considered data.** The identification algorithm constructs an identified model starting from experimental data that can be inputs and/or outputs of the observed system.
- **Strategy.** If the identification algorithm returns all possible models representing the observed behaviour, the algorithm is called enumerative. If only one of the possible models is given, it is constructive.
- **Execution.** If the construction of the model can be performed during the system operation by computing a new model from new measurements of the system inputs and/or outputs, the execution is made on-line. Otherwise, the execution is off-line; the algorithm is not able to run at the same time than the system.
- **Complexity.** This term refers to the computational complexity of the identification algorithm. Polynomial time procedures are better than exponential ones for coping with large systems exhibiting a large amount of input-output sequences.

The main characteristics of the considered methods are summarized in Table 1.3.

Identification approach / Comparison criteria	**Progressive approach**	**Parametric automata approaches**	**Integer Programming approaches**	**Neural Networks approach**	**Parametric IPN approach**	**Workflow mining approaches**
DES to be identified characteristics						
Type of inputs/outputs	Logical	Logical	Discrete	Discrete	Logical	Discrete
Iterative Behaviour	Repetitive	Cyclic	None	None	Cyclic	Cyclic
Identification process characteristics						
A-priori information	Absolute	Absolute	Relative	Relative	Absolute	Relative
Model updating	Incremental	Incremental	Global	Global	Incremental	Global
Identified model characteristics						
Concurrency	Explicit	Implicit	Explicit	Explicit	Explicit	Explicit
Accuracy	Non-complete	Complete	Non-complete	Non-complete	Complete	Non-complete
Algorithm characteristics						
Considered data	Outputs	Inputs and outputs	Events and outputs	Events	Inputs and outputs	Events
Strategy	Constructive	Constructive	Enumerative	Enumerative	Constructive	Constructive
Execution	On-line	Off-line	Off-line/on-line	Off-line	Off-line	Off-line
Complexity	Polynomial	Polynomial	Exponential	Polynomial	Polynomial	Polynomial

Table 1.3 Identification methods characteristics

1.5. Conclusion

We have reviewed different identification techniques found in the literature. An analysis of their main characteristics has been made in which methods have been classified according to several criteria. Although most of the techniques have some of the characteristics required for addressing the identification problem in black-box approach, there are other features that are not included in such methods for dealing with large I/O sequences measured from real reactive complex DES that exhibit repetitive behaviour. In the next chapter, we will state the particular issues of the Identification problem dealt in this thesis and we show that existent identification methods are not well adapted because of their characteristics.

Chapter 2

Identification of automated Discrete Event Systems

Abstract. This chapter states the problem of identification of industrial automated discrete event systems. It presents the main characteristics of industrial systems which have to be taken into account by an identification method, and then the application reviewed methods are analysed within this context, Also, data collection for identification is described.

2.1. Problem statement

The systems considered in this work are closed loop controlled DES (Figure 2.1); they consist of a plant and its industrial controller (in many cases a Programmable Logic Controller: PLC). The behaviour of such systems (i.e. the PLC-plant) can be observed by collecting the signals exchanged between controller and plant.

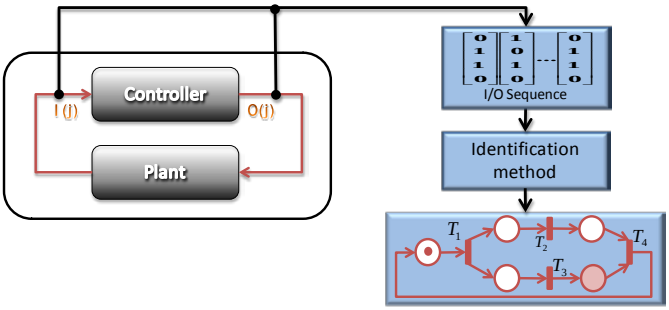


Figure 2.1 Closed loop controller-plant DES

Beyond the theoretical interest of defining model construction methods from symbol sequences, the aim of developing identification methods for actual industrial automated systems establishes challenges related to algorithms scalability and technological issues: the techniques must be efficient to cope with large and complex systems that handle actual signals.

In our approach the aim is to discover, from observations of the system behaviour expressed as a single sequence of its input and output signals, and how components of the system are related, and then construct a compact model which can explicitly show discovered behaviour. Identification of already existent systems involves two important aspects to consider: the system operation and the observation process. Technological issues of both must be considered in the proposed algorithms in order to construct suitable abstractions.

The identification is made from the point of view of the PLC (Figure 2.1). Several phenomena, due to the interaction between plant and controller, increase the complexity of the identification process; however they must be taken into account when real controlled DES have to be identified:

- Any input evolution (signal emitted by the plant through a sensor) does not always provoke an output evolution (signal emitted by the PLC to an actuator). In practice, few of input changes provoke output evolutions;
- Contrary to assumptions established in DES theory, many I/O signals may occur simultaneously; moreover, non simultaneous I/O signal changes are often simultaneously observed;
- When output changes are provoked by input changes, this causal relationship is not necessarily captured simultaneously;

Now, we are going to explain these phenomena.

2.1.1. Basics on PLC technology

Industrial Programmable Logic Controllers (PLC) are used extensively in manufacturing industries for complex control applications [Lampérière-Couffin, 1999].

A PLC cyclically performs three main steps (Figure 2.2): “input reading” (I) where it reads the signals from the sensors, “program execution” (PEX) to determine the new

outputs values for the actuators, and “output writing” (O) where the newly determined commands are sent to the plant actuators.

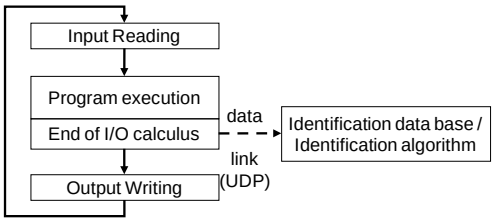


Figure 2.2 PLC cycle and data collection

At the end of the PEX phase the current values of inputs and outputs (I/O) are sent from the PLC to a computer and stored for a later treatment by the identification algorithm.

2.1.2. Experimental constraints

In the identification problem we are addressing, the PLC program is assumed to be unknown. Numerous PLC programming languages exist; most of them depend on PLC manufacturers. However, there exists a standard (IEC 61131-3) in which the semantics of four programming languages (Ladder Diagram, Function Block Diagram, Structured Text, and Instruction List) and of a structuring language (Sequential Function Chart) are given. Sequential Function Charts (SFC) are used in this thesis only for describing the diverse situations addressed by the proposed method.

SFC is a graphical programming language used for PLCs. It is an extended state machine that contains primitives to describe sequential, parallel and alternative behaviours. It enables the partitioning of a PLC program (or function block) into a set of steps and transitions interconnected by directed links. Main components of SFC are: Steps with associated actions, Transitions with associated logic conditions, Directed links between steps and transitions. Steps in an SFC diagram can be active or inactive. One step is activated when all steps above it are active and the connecting transition is validated (i.e. its associated condition is true). When a transition is fired, all steps above are deactivated and simultaneously all steps below are activated. Actions associated with steps can be of several types, the most relevant ones being Continuous (N), Set (S) and Reset (R). Apart from the obvious meaning of Set and Reset, an N action ensures that its target variable is set to 1 as long as the step is active.

Due to the PLC cycle, some situations between inputs and outputs could arise. Consider a situation described in Figure 2.3 (current active step is #10; *a* and *b* are two input signals to the PLC; A and B are two output signals).

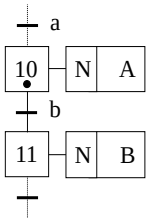


Figure 2.3 A single input is the condition for state evolution

Changes in the state and outputs will occur when signal *b* is active; however other input signals may evolve without consequence in the outputs. This must be considered in the identification algorithm.

Consider now the time diagram in Figure 2.4. Two signals are asynchronously emitted by sensors of the plant between two successive "input reading" phases (I) of the PLC cycle. These two signals will be simultaneously read during the next I phase and observed as simultaneous events in the identification data base. In DES theory events cannot occur simultaneously; so an observed event vector will therefore be defined as a change of value in an entry of an I/O vector.

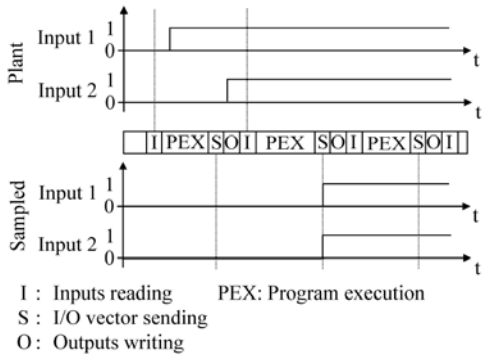


Figure 2.4 Apparent simultaneous evolution of several inputs

Now, let us consider the situation described in Figure 2.5(a). As shown in the time diagram in Figure 2.5(b), if input "b" changes its value from 0 to 1, the corresponding change in "B" is not provoked immediately, since it is necessary first a change in output "A". In this case cause and effect cannot be captured simultaneously but will be detected only if we observe a sequence of 4 consecutive events.

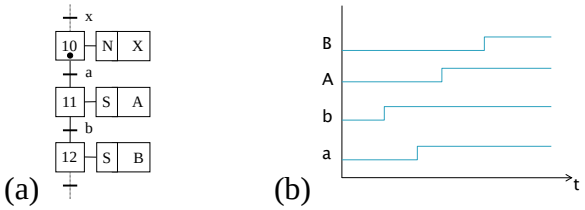


Figure 2.5 I/O causality and sequences of events

These three scenarios show that the implementation of a controller and its interaction with the plant introduces phenomena that must be taken into account by the identification algorithm.

2.2. Input data and output model

As stated before, at each end of the PLC Program execution phase, the current value of all Inputs and Outputs is captured and recorded in a data base. Thus, the only available data for the identification procedure is a single I/O vector sequence:

$$w = \left[\frac{I(1)}{O(1)}\right], \left[\frac{I(2)}{O(2)}\right], \left[\frac{I(3)}{O(3)}\right], \ldots$$

where, $I(j) = \begin{bmatrix} I_0(j) \\ I_1(j) \\ \vdots \\ I_{m-1}(j) \end{bmatrix}$ and $O(j) = \begin{bmatrix} O_0(j) \\ O_1(j) \\ \vdots \\ O_{n-1}(j) \end{bmatrix}$ are the j-th observed input and output vectors of size *m* and *n* respectively in the sequence *w*.

Our aim is to represent the system's behaviour from the I/O vector sequence into an IPN as shown in Figure 2.6. IPN has been chosen as modelling formalism because of their inherent capacity to represent reactive behaviour involving input and output information as well as complex behaviour such as parallelism.

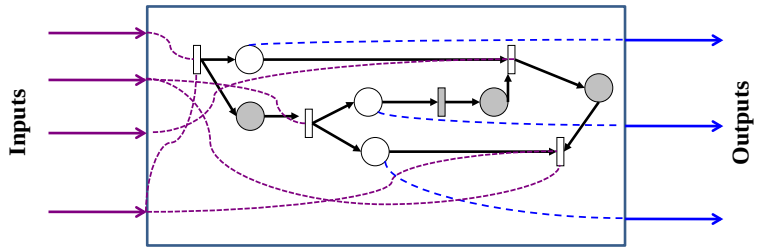


Figure 2.6 Input and output information represented by IPN

In next section we will provide a first approximation we have developed to translate I/O information into IPN models.

2.3. Assumptions

Herein we summarize the assumptions held for addressing the identification problem of industrial discrete manufacturing systems.

- The constructed model is 1-bounded (no counters will be included).
- Deadlock-free. We assume that during the acquisition of the input-output sequence, no blocking occurs during the functioning of the system.
- Binary. The input and output signals handled by the algorithm must be binary.
- Long time operation. We consider that the input-output sequence has been measured during a long time elapse in which all the programmed tasks are performed.
- Timed behaviour is not computed. Some PLCs include timers in their evolution conditions; such conditions are not computed in this work.
- Simultaneous input changes may be measured. Due to the phenomena explained in section 2.1, several input changes may appear between consecutive vectors.
- Black-box. The only available information of the system is the input-output sequence.
- Monolithic. There is only one PLC controlling the plant.

2.4. Discussion

After describing the features of the systems to be identified, we can discuss whether or not the analyzed methods are suitable for our particular problem

- **Progressive identification**

The polynomial time execution of the algorithm in [Meda, 1998], [Meda, 2000a], [Meda, 2000b], [Meda, 2001], [Meda, 2002a], [Meda, 2002b], [Meda, 2003], [Meda, 2005] is a good characteristic which should be aimed by all of the identification approaches. However, a strong limitation of this technique is that it does not take into account the inputs of the system, which is very relevant for the closed-loop behaviour systems we consider in this work. Applying this method to such kind of systems would lead to models in which same output changes caused by different input evolutions

would not be distinguished, and then incorrect behaviour could be introduced in the created model. For example, consider Figure 2.7. The model at the right has been built by ignoring the input information, and the relation between input *a* and output *B* has been lost, as well as relation between input *b* and output *D*.

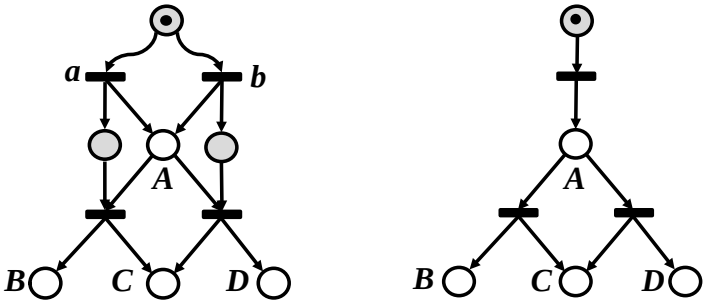


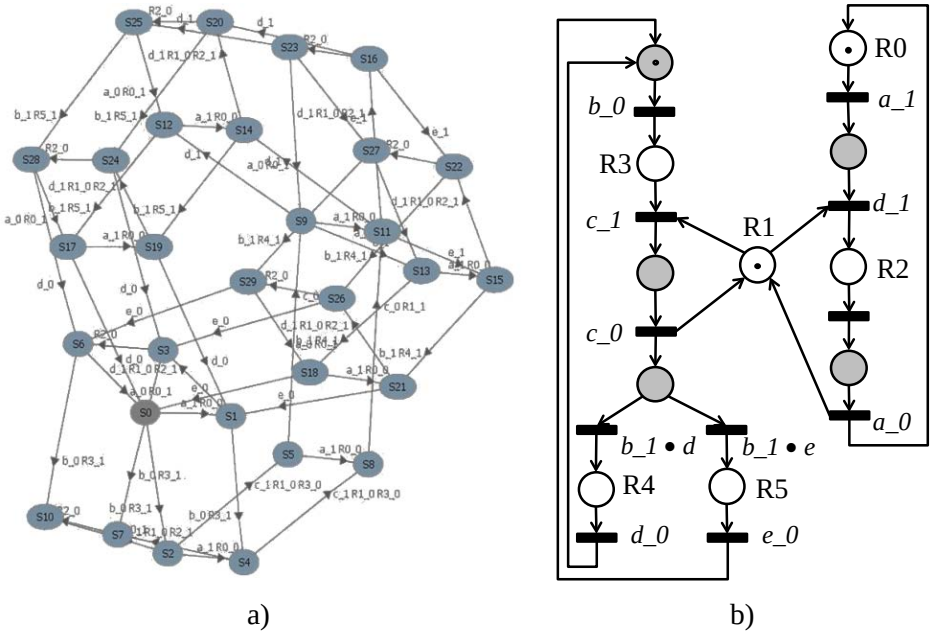
Figure 2.7 Lost of the input information

The treatment of the input information during identification is not straightforward, as will be shown in next chapters.

- **Parametric automata identification**

Even if the work in [Klein 2005a], [Klein 2005b] is well adapted for experimental treatment, which is very important for the identification problem; the methodology has some weak points.

The first one is that concurrence is not explicitly shown in automata models. Lack of explicit concurrence may not be important for applications such as fault detection and a model as shown in Figure 2.8.a would be enough. But for other applications like reengineering or supervision, an expressive and compact model would be much useful. For example, if the aim of the identification is to produce a model which can help an engineer to understand how a system works, a structurally rich model as illustrated in Figure 2.8.b would be better.



a) b)

Figure 2.8 Automata and Petri net models for the same behaviour.

The second disadvantage is that the methodology considers that cyclic sequences are provided. In order to collect such sequences, the identified system should be reinitialized at each production cycle. This assumption is not possible to fulfil in many cases in which the system does not return to the initial state for cycling.

- **Parametric automata distributed identification**

One of the limitations of [Roth, 2010a], [Roth, 2010b] concerns the ability to find concurrent behaviour. If several decisions exist in the system, the different possible conditional behaviours will be wrongly considered (by the branching degree measure) as concurrent behaviours.

This algorithm is strongly focused on fault identification, and even if it represents an improvement over [Klein, 2005a], the built models are not good to represent structural information such as parallelism inside subsystems and resource sharing. Another limitation is that the number of sub-systems is known a-priori, but this number is not always easily determined.

- **Integer Linear Programming Language Identification**

Even if the synthesis methodology in [Giua, 2005], [Cabasino, 2006a], [Cabasino, 2006b] [Cabasino, 2006c], [Cabasino, 2009] is elegant and theoretically strong, there are several reasons which make this approach inappropriate for identification on actual experimental systems. First of all, the exponential number of counterexamples in the set $\mathscr{D}$ makes the problem intractable for most of the real systems.

Also, the a priori knowledge of the transitions (events) occurring in the system is not available when a black-box identification approach is performed. Some kind of computation has to be made in order to find the events.

- **Integer Linear Programming Identification**

In the work in [Dotoli, 2006a], [Dotoli, 2006b], [Dotoli, 2007], [Dotoli, 2008], it is no longer necessary the construction of counter-examples as in the method by [Giua, 2005] and extensions; thus, the writing of the linear algebraic constrains is no longer exponential. However, the method requires the knowledge of an upper bound on the number of places. Such kind of knowledge is not available in a black-box approach and in fact, the upper bound could even have a strong influence on the result: different solutions (and even no solution) depending on the number of non-observable places may be found, each one representing a different PN structure, and thus, a different behaviour.

Also, it is well known that Integer Linear Programming is a NP-hard problem, and thus, if the number of transitions increases, the problem becomes intractable.

- **Neural Networks approach**

Similarly to other approaches, the technique presented in [Ould El Medhi, 2006] has some characteristics that do not make it suitable for our identification problem. First of all, the event set is supposed to be known, which is not true in a black-box approximation. Also the hypothesis of exhaustive knowledge of the propagation set relation between events is very strong. Another inconvenient is that when the places

reuse technique is applied, sometimes the resulting net is not blocking free and it does not allow reproducing the observed sequence.

On the other hand, the learning algorithm is strongly dependent on parameters which are specified by the user, and no theoretical basis for choosing them is given in the description of the technique.

Since the goal of this technique is the learning of the matrix B, if two events have been observed consecutively a place must exist to relate them; consequently, this technique does not include a concurrence analysis, and thus, the inherent structural expressiveness of the Petri net is not exploited.

- **Parametric interpreted Petri net identification**

Since the method in [Estrada, 2009][Estrada, 2010b] is the basis for part of the work presented here, a deeper analysis of its disadvantages is made on next chapters. However, we can enounce some of them. First of all, the algorithm considers that sequences start always at the initial state of the DES, and as discussed before, the acquisition of such sequences is not easy to achieve. Also, transformations for determining concurrency in the IPN made by the algorithm are based on the observation of all the possible combinations of an event set; however, in the general case only a subset of such combinations is observed during a finite functioning of the system to identify.

In next chapters, it will also show that the technique could be improved by means of an input-output relation analysis.

- **Probabilistic workflow mining**

Even if a concurrence analysis is made on [Cook, 2004], the found models are not well defined; their construction is strongly focused on workflow operations. Similarly to other techniques, an exhaustive observation of all possible combinations of events is supposed.

Due to the join and fork workflow syntax, an event can only be involved in a concurrence or selection situation, but not in both. It is assumed that if several events follow an event, they are all concurrent or they are all independent. However, in the type of systems we consider, such hypothesis may not be satisfied and thus, the system would not be correctly identified by this method. However, the statistical approach followed in this method has inspired our methodology proposed in Chapter 4.

- **Alfa-algorithm**

The methodology from [van der Aalst, 2004] seems to be well adapted for workflow problems. However, it presents some characteristics that are incompatible with closed-loop behaviour identification. One of these characteristics is that to assure the correctness of this method, an exhaustive list of behaviour is needed, but in our particular problem, it is not possible to assure that all of the possible sequences have been observed in a finite time. Thus, if the event log is not complete, non-observed behaviour could be introduced.

Another characteristic is that the input for the algorithm is a set of traces. In our case, traces would be system cycles, but as stated before, system cycles are not known a-priori.

Finally, in this method the events are a-priori defined, but in our case such knowledge is not available.

2.5. Conclusion

The problem of identification of industrial automated discrete event systems has been stated. The reviewed identification methodologies have been analysed considering operation characteristics of real controlled systems. It can be noticed that none of the proposed methodologies is perfectly adapted for the particular problem faced in this thesis. In order to cope with the challenges stated by identification problem, which have not been dealt by existent identification methods, two different and complementary approaches have been explored. They will be described in the next chapters. The first one is an extension to the work in [Estrada, 2009], in which the hypothesis of knowledge of the system cycles is removed. The second one is a statistical approach which produces compact and expressive models.

Chapter 3

A Stepwise Identification Method

Abstract. This chapter presents an identification method for discrete event manufacturing systems that are automated by a programmable logic controller (PLC). The behaviour of the closed loop system (PLC and Plant) is observed during its operation and represented by a single long sequence of observed input/output signals vectors. The proposed method follows a black-box and passive identification approach; it allows building stepwise an interpreted Petri net (IPN) model. The identification method is composed of several polynomial time algorithms.

3.1. Overview of the method

This method allows the progressive construction of a safe IPN (see Appendix A, *Definition A.3*) representing exactly the observed input-output language of length κ +1 of the DES.

From the I/O vector sequence, an event sequence is computed and a sequence of event substrings of length κ is built. Every substring is associated to a transition of a PN, which describes the causal relationship between event substrings. A PN node path formed by non-observable places represents the substring sequence; this path is built taking into account the possible repetitive observed behaviour (internal model). Then simplifications may be applied. Notice that the number of non-observable places is not predefined.

Finally, the model is completed by including observable places which are related to pertinent transitions in the PN according to output changes provoked by events; also input symbols are associated. This part of the algorithm can be concurrently performed at any moment, for example when a cycle is identified, whilst the internal model is updated by processing the new I/O vectors.

The procedure for building the IPN model from the I/O sequence has been published in [Estrada, 2011] and can be summarized on Figure 3.1. It consists of five main steps that are described below.

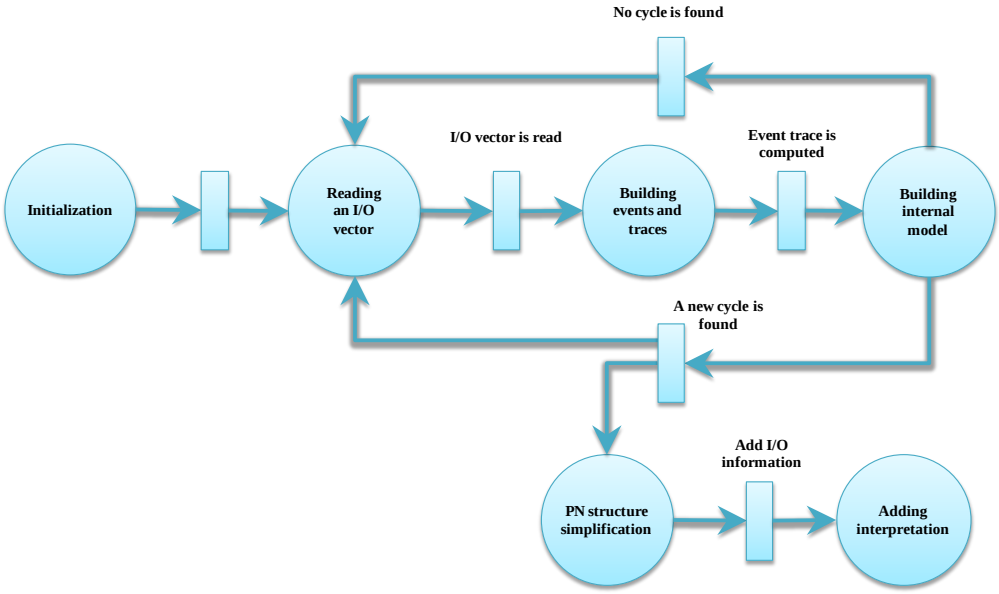


Figure 3.1 Stages of the identification algorithm

3.1.1. Initialization stage

In this step, a PN structure is initiated. This is done by the following statements:

$T \leftarrow \varnothing$; $ET \leftarrow \varnothing$; $P \leftarrow \{p_{ini}\}$; *An initial empty set of transitions T is created, as well as an initial empty event traces set ET, and an initial set of places P containing a place* p_{ini}

$M_0(p_{ini}) \leftarrow 1$; $\mu(p_{ini}) \leftarrow w(1)$; *current*$\leftarrow p_{ini}$; *//A token is placed on* p_{ini} *and such a place is associated to the first observed vector w(1)and taken as current*

3.1.2. Building events and traces

Once the net is initiated, the procedure iterates on subsequent I/O vectors in the sequence. Each I/O vector is considered to update the events sequence and the events traces according to next definitions.

Definition 3.1 An observed event vector $E(j)$ is the variation between two consecutive I/O vectors: $E(j) = w(j + 1) - w(j)$. The m first entries of $E(j)$, denoted as $IE(j)$ correspond to the variation between two consecutive input vectors $I(j)$, $I(j + 1)$ (input event). A symbolic input event $\lambda'(E(j))$ is a string representation of the input event vector $IE(j)$; it is computed as:

$$\lambda'(E(j)) = \begin{cases} I_i\_1 \text{ if } I_i(j+1) - I_i(j) = 1 \\ I_i\_0 \text{ if } I_i(j+1) - I_i(j) = -1 \\ \varepsilon \text{ if } I_i(j+1) - I_i(j) = 0 \end{cases}$$

Then for a sequence w, a sequence of observed events $E = E(1)\ E(2)...\ E(j)...$ is obtained. During the process, if the difference has not been observed before, a new event e_j is created ($E(j) = e_j$).

Definition 3.2 An event trace $\tau^{\kappa}(j)$ is the substring from E of length κ whose last event is $E(j)$. $\tau^{\kappa}(j) = E(j - \kappa + 1)E(j - \kappa + 2)\ldots E(j)$.

This notion is useful to determine during the identification process if two states represent the same internal behaviour. Then the notion of equivalent states involves the history of κ events that lead to such states.

Definition 3.3 Two states of the model representing the identified system are κ-equivalent if the event traces $\tau^{\kappa}(j)$ leading to such states are the same.

3.1.3. Building internal model

Once the sequence of event traces has been obtained, every trace $\tau^{\kappa}(j)$ is related to a transition in the IPN through a function $\gamma: T \rightarrow \{\tau^{\kappa}(j)\}$; the firing of a transition implies that κ consecutive events related to such a transition have been observed.

In order to preserve firing order between transitions, dependencies are created between them and associated with an observed marking through the function $\mu: P^u \rightarrow \{\varphi M_i |\ M_i \in R(N, M_0)\}$, which relates every non-measurable place with an observed marking, such that every transition has only one input place and one output place ($\forall t_r \in T, |{}^{\bullet}t_r| = |t_r{}^{\bullet}| = 1$).

Notice that the number of non-measurable places is not predefined. When an event trace $\tau^{\kappa}(j)$ is found again in the sequence, the associated dependency must be used if it leads to the same observed marking.

Let e_j be the last event vector in the trace $\tau^{\kappa}(j)$; the associated transition will be denoted as $t_r^{e_j}$ (more than one transition may have associated the same e_j).

This strategy can be systematically performed following the next procedure.

Algorithm 3.1. Building internal model

Input: $\tau^{\kappa}(j)$, Petri net structure G, μ, γ, ET
Output: Updated G', μ', γ', ET'

If $\tau^{\kappa}(j) \notin ET$ then *//If the computed trace is new*

$ET \leftarrow ET \cup \{\tau^{\kappa}(j)\}$; $T \leftarrow T \cup t_r^{ej}$; $\gamma(t_r^{ej}) \leftarrow \tau^{\kappa}(j)$; $\forall p_a \in P, Pre(p_a, t_r^{ej}) \leftarrow 0$, $Post(p_a, t_r^{ej}) \leftarrow 0$;
//create a transition t_r^{ej} *to represent the trace* $\tau^{\kappa}(j)$
$I(current, t_r^{ej}) \leftarrow 1$; *//create an arc from current to* t_r^{ej}
$P \leftarrow P \cup \{p_{out}\}; \forall t_b \in T$, $Pre(p_{out}, tb) \leftarrow 0$, $Post(p_{out}, t_b) \leftarrow 0$; $\mu(p_{out}) \leftarrow \mu(current) + e_j$; *//create a new place* p_{out} *and associate it with correspondent marking*
$Post(p_{out}, t_r^{ej}) \leftarrow 1$; *//create an arc from* t_r^{ej} *to* p_{out}
$current \leftarrow p_{out}$; *//take* p_{out} *as current*
else*//The computed trace is not new*
If $t_r^{ej} \in current^{\bullet}$ and $\gamma(t_r^{ej}) = \tau^{\kappa}(j)$ then *//If one of the output transitions* t_r^{ej} *of current place represents the observed trace* $\tau^{\kappa}(j)$
current$\leftarrow (t_r^{ej})^{\bullet}$; *//take the output place of* t_r^{ej} *as current*
else *//Current place has not an output transition representing* $\tau^{\kappa}(j)$
If $\exists t_r^{ej} \in T | \gamma(t_r^{ej}) = \tau^{\kappa}(j)$ and $\mu({}^{\bullet}t_r^{ej}) = \mu(current)$ then *//There is a transition* t_r^{ej} *representing* $\tau^{\kappa}(j)$*such that its input place* ${}^{\bullet}t_r^{ej}$ *has the same associated marking* $\mu({}^{\bullet}t_r^{ej})$ *than current place*
take $p_{in} = {}^{\bullet}t_r^{ej}$; *//take* ${}^{\bullet}t_r^{ej}$ *as* p_{in}
merge(*current*,p_{in}); *//merge current place with such an input place*
$current \leftarrow (t_r^{ej})^{\bullet}$; *//take the output place of the transition* t_r^{ej} *as current*
else consider $\tau^{\kappa}(j)$ as new

Subroutine *merge*(p_1, p_2) *//merges places* p_1 *and* p_2

$\forall t_b \in T$, $Pre(p_1, t_b) \leftarrow \vec{\oplus}(Pre(p_1, t_b), Pre(p_2, t_b))$; *//* $\vec{\oplus}$ *is a vector bitwise or operation;*

$\forall t_b \in T$, $Post(p_1, t_b) \leftarrow \vec{\oplus}(Post(p_1, t_b), Post(p_2, t_b))$;

$P \leftarrow P \backslash \{p_2\}$ *// delete place* p_2

Remark. The algorithm performs a search operation for each computed trace and adds a new transition for traces that have not been yet observed, which implies updates in the *Pre* and *Post* matrices. Then, it is easy to see that *Algorithm 3.1* is executed in polynomial time on length of observed sequence and number of different traces.

Property 1. The IPN *G* built through *Algorithm 3.1* represents all and only all the traces
$\tau^{\kappa}(j)$.

Proof. By construction, the sequence *E* is represented in G by a path starting from p_{ini} including the sequence of t_r^{ej}, which represents the observation of the event trace $\tau^{\kappa}(j)$. The reuse of computed transitions having associated the same event traces, during the processing of trace $\tau^{\kappa}(j)$, is done only when common paths of length κ are built, which does not introduce other sequences. ♦

3.1.4. PN structure simplification

After performing step 3, the algorithm reads a new I/O vector by returning to step 2. Nevertheless, notice that merging places through step 3 of the algorithm could lead to merging of equivalent transitions. When such a merging is performed, a cycle on the PN is created. This is considered as a representative change in the structure of the model, and thus, simultaneously with launching of step 2, step 4 is executed to make a PN simplification procedure.

Some transformations may be performed when there are transitions that appear in the sequences in different order describing their interleaved firing; this behaviour is exhibited by concurrent transitions. The analysis can be performed on a model component comprised between two transitions relied by several paths containing the concurrent transitions. If there are *m!* paths, we can explore if there exists *m* different transitions in the paths and every sequence is a permutation from each other. When it is verified, the subnet can be transformed into a concurrent component of *G'* preserving the same behaviour.

The simplification by analysis of concurrency is not strictly necessary for representing the event sequences; however the equivalent model with concurrent transitions may be simpler; the aim of this simplification is not minimizing the number of nodes in the model, but obtaining fairly reduced models useful for understanding the DES behaviour. However, the analysis could be inefficient when the number of paths in the subnet is large, and thus some improving must be done for this purpose.

3.1.5. Adding interpretation and simplifying

Once the event sequences are represented in the basic model, it must be completed by adding the output changes represented by the events and their respective inputs. Recall that events are vectors computed from the difference of consecutive vectors; thus e_j relates observable places representing the outputs yielding the incidence matrix corresponding to observable places.

Algorithm 3.2. Representing outputs changes

Input: *G'*
Output: *Q'*: the IPN including observable places

Step 1. $P \leftarrow P \cup \{p_1, p_2, \ldots, p_q\}$ *//Create q observable places for every one of the components in the output vectors*

Step 2. $\forall t_r^{ej} \in T$:
If $e_j(i) = -1$ then $Pre(p_i, t_r^{ej}) \leftarrow 1$ and $Post(p_i, t_r^{ej}) \leftarrow 0$; *//Event e_j takes 1 from component i*
If $e_j(i) = 1$ then $Pre(p_i, t_r^{ej}) \leftarrow 0$ and $Post(p_i, t_r^{ej}) \leftarrow 1$; *//Event e_j puts 1 into component i*
If $e_j(i) = 0$ then $Pre(p_i, t_r^{ej}) \leftarrow 0$ and $Post(p_i, t_r^{ej}) \leftarrow 0$; *//Event e_j does not affect component i*

Step 3. If component *i* of vector $w(1)$ is 1 then $M_0(p_i) \leftarrow 1$, otherwise $M_0(p_i) \leftarrow 0$ *//Put tokens in the observable places to represent the first output vector*

Notice that *Algorithm 3.2* executes in polynomial time in the number of outputs and the total of computed transitions in *Algorithm 3.1*

After adding observable places, some non-observable places could become implicit places. They can be removed: if there is a non-observable place whose inputs and outputs are exactly the same than any observable place, remove such a non-observable place and its input and output arcs.

Once the output adding and implicit places deleting has been performed, it only remains to add input information to complete the IPN model. Input information is associated with labels for transitions in a natural way given by the symbolic event input function of *Definition 3.1*. Next algorithm describes a systematic way to do it.

Algorithm 3.3. Representing input changes

Input: G', $\lambda'(e_j)$
Output: Q : the final model of the identification process

Step 1. $\forall t_r^{ej} \in T$, $\lambda(t_r^{ej}) \leftarrow \lambda'(e_j)$ *//Associate to each* t_r^{ej} *the symbolic input event registered at the detection of* e_j.

Is easy to see that the input addition is executed in polynomial time in the number of transitions in the net.

Proposition 4. 1. For a DES *S* and an identification parameter κ, algorithm in Figure 3.1 yields an IPN model (Q,M_0) which represents exactly $\mathcal{L}^{\kappa+1}(S)$.

Proof. Since the deletion of implicit places does not alter the language of (Q,M_0), we make the proof with the model obtained before this procedure. The firing of a transition *t* in the system is not affected by the addition of arcs to, and from *t*, since these arcs were computed from differences of vectors in *w*. Then, according to *Property 1*, every event sequence of length less or equal than κ belongs to the language of the net if and only if it was observed.

The sequences of transitions of length less or equal than κ that can be fired lead to markings in the measurable places that also have been observed (since the marking change provoked in the measurable places was obtained from the difference of observed vectors). Then, we have that sequences of observed output vectors of length less or equal than $\kappa + 1$ correspond to sequences of marking vectors in the net and the net represents exactly $\mathcal{L}^{\kappa+1}(S)$. ♦

Example 3.1 Consider a DES with three output signals, Φ = {A, B, C}, and three input signals Σ = {a, b, c}. The entries of the binary I/O vectors have the following correspondence: [a b c | A B C]^T. An I/O sequence is progressively observed. The first measured I/O vector corresponds to the initial state of the DES: $w(1) = [0\ 0\ 0 \mid 0\ 0\ 0]^T$.

When a second I/O vector $w(2) = [1\ 0\ 0 \mid 1\ 0\ 0]^T$ is read, the event vector $E(1) = e_1 = [1\ 0\ 0 \mid 1\ 0\ 0]^T$ is computed; the input event vector is $\beta(1) = [1\ 0\ 0]^T$ and its corresponding symbolic input event is $\lambda'(1) = a\_1$, i.e. the rising edge of a.

Considering a value $\kappa = 1$, we can compute the first event trace $\tau^1(1) = e_1$. Notice that, in this case, trace and event are the same. This event trace is related with a transition of the IPN. The IPN constructed after observing two I/O vectors is on Figure 3.2.

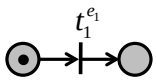


Figure 3.2 PN representing e_1

When a third I/O vector $w(3) = [1\ 0\ 0 \mid 0\ 0\ 0]^T$ is read, $E(2) = e_2 = [0\ 0\ 0 \mid -1\ 0\ 0]^T$, $\beta(2) = [0\ 0\ 0]^T$ and $\lambda'(2) = \varepsilon$ are computed and the model is updated, as showed in Figure 3.3.

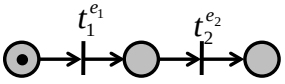


Figure 3.3 PN representing the sequence e_1e_2

Until 8th I/O vector, the situation is quite similar: new events are computed and the model is updated.

$$w = \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{e_1} \begin{bmatrix}1\\0\\0\\1\\0\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\1\\0\\0\end{bmatrix} \xrightarrow{e_2} \begin{bmatrix}0\\0\\0\\-1\\0\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{e_3} \begin{bmatrix}0\\0\\0\\0\\1\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\1\\0\\0\end{bmatrix} \xrightarrow{e_4} \begin{bmatrix}0\\0\\0\\-1\\0\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{e_5} \begin{bmatrix}0\\0\\1\\0\\0\\1\end{bmatrix} \begin{bmatrix}1\\0\\1\\0\\0\\1\end{bmatrix} \xrightarrow{e_6} \begin{bmatrix}0\\0\\-1\\0\\0\\-1\end{bmatrix} \begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{e_7} \begin{bmatrix}-1\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix}$$

When 9th vector $w(9) = [1\ 0\ 0 \mid 1\ 0\ 0]^T$ is processed, the event $E(8) = e_1 = [1\ 0\ 0 \mid 1\ 0\ 0]^T$ is computed and the trace $\tau^1(8)$ is identified through Step 3 as an already observed trace e_1. Since it leads to the same marking than the input place of $t_1{}^{e1}$, such a place and the output place of $t_7{}^{e7}$ can be merged as observed on Figure 3.4.

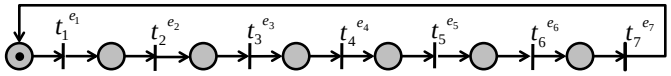


Figure 3.4 Internal model for the first detected cycle

Since a cycle is found, steps 4 and 5 of the algorithm are executed, leading to an intermediate IPN model showed on Figure 3.5.

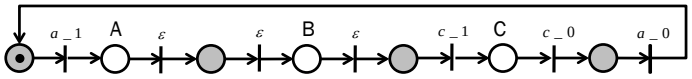


Figure 3.5 IPN for the first detected cycle

Simultaneously to the creation of the intermediate IPN, more I/O vectors are added to the observed sequence and PN is updated:

$$\ldots \begin{bmatrix}1\\0\\0\\1\\0\\0\end{bmatrix} \xrightarrow{e_3} \begin{bmatrix}0\\0\\0\\0\\1\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\1\\1\\0\end{bmatrix} \xrightarrow{e_2} \begin{bmatrix}0\\0\\0\\-1\\0\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\0\\1\\0\end{bmatrix} \xrightarrow{e_4} \begin{bmatrix}0\\0\\0\\0\\-1\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{e_7} \begin{bmatrix}-1\\0\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{e_1} \begin{bmatrix}1\\0\\0\\1\\0\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\1\\0\\0\end{bmatrix} \xrightarrow{e_3} \begin{bmatrix}0\\0\\0\\0\\1\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\1\\1\\0\end{bmatrix} \xrightarrow{e_2} \begin{bmatrix}0\\0\\0\\-1\\0\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\0\\1\\0\end{bmatrix} \xrightarrow{e_4} \begin{bmatrix}0\\0\\0\\0\\-1\\0\end{bmatrix} \begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{e_8} \begin{bmatrix}0\\1\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}1\\1\\0\\0\\0\\0\end{bmatrix}$$

Two more cycles are found in this sequence and intermediate IPN models are created. We show only the PN obtained after finding the second cycle (Figure 3.6) and its equivalent model transformed by analyzing concurrency (Figure 3.7). After applying the steps 4 and 5 the IPN obtained from this PN is showed in Figure 3.8.

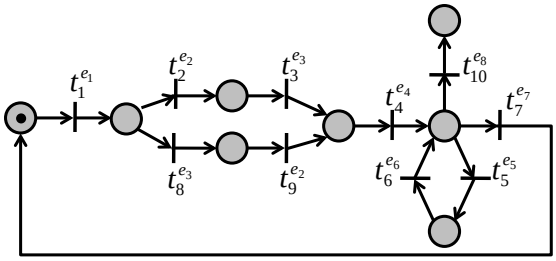


Figure 3.6 PN corresponding to the whole I/O sequence

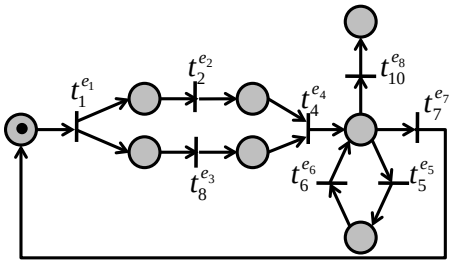


Figure 3.7 Equivalent internal model representing concurrency

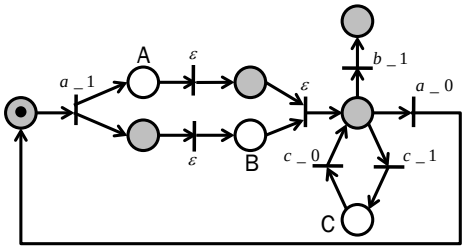


Figure 3.8 IPN for the complete sequence

3.2. Discussion

The parameter κ helps to distinguish sequences of events that look similar during the construction of the basic internal model; its value indicates the history of past events that have to be considered for deciding the state equivalence [Estrada, 2012]. On the one hand, high values of κ imply distinguishing more sequences avoiding path fusion during the model construction; thus the obtained models are more accurate but less compact. On the other hand low values of κ allow more state fusions; the obtained model are more compact but more paths can be created yielding an overrepresentation of the observed behaviour.

In general it is not possible to establish a priori the value of κ, since it is assumed that the system is unknown. However, in practice the identification procedure can be applied using several values of κ (because it is not time consuming). Compact models allow a first approximation to the understanding of the system functioning, whilst larger models provide a more precise description. For small examples, κ=1 or κ=2 allows distinguishing event sequences whilst compact models are built. In actual industrial systems the difference can be more drastic.

Now, we are going to illustrate the application of the above described algorithms to an example to analyze several characteristics of the synthesized model in order to point out the limitations of the method that have to be faced. The models are obtained automatically with the help of a software tool which will be described in Chapter 5.

Example 3.2 Consider a small size application example dealing with an automated manufacturing system (Figure 3.9). The purpose of such a system is to sort parcels

according to their size. It has 9 inputs (signal sensors) from the system: a0, a1, a2, b0, b1, c0, c1, k1, k2, and 4 outputs (signal to the actuators): A+, A-, B, C.

Parcels arrive randomly at conveyor 1 and they are sorted one by one. The sensors k1 and k2 inform whether a parcel is small (k1=1) or big (k2=1). Big parcels are pushed to conveyor 3, and small parcels to conveyor 2 using the double-acting cylinder A. When a parcel arrives at the appropriate position, it is pushed by one of the single-acting cylinders B or C on the according conveyor. A new parcel may arrive (an detected through sensors k1 and k2) while the previous one is being treated.

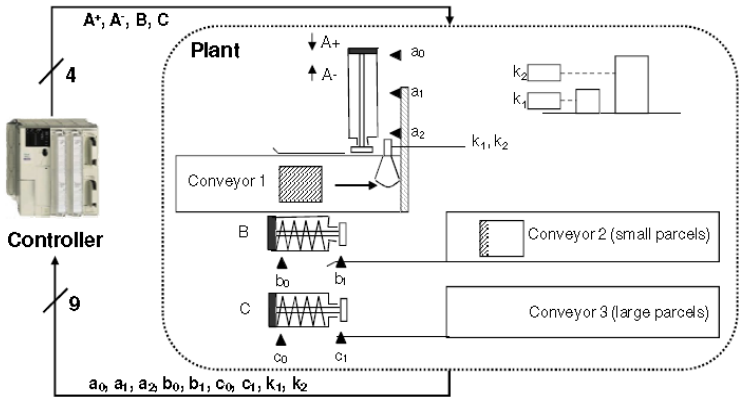


Figure 3.9 Diagram of the *Example 3.2*

The sequence in Figure 3.10 has been considered. Each line of a column corresponds to an I/O vector, where binary values correspond to signals A+, A-, B, C, a0, a1, a2, b0, b1, c0, c1, k1, k2 respectively. From such a sequence, the model on Figure 3.11 has been constructed.

0000001001010	0100100001010	0100010000110	1000010001010	0100010001000	1000000001010
1000101001010	0000101001010	0000011000010	1000000001010	0000011001000	0110000101010
1000100001010	1000101001010	1000011001010	1000000101010	1000011001010	0110000100010
1000000001010	1000100001010	1000010001010	1000000001010	1000010001010	0110000000010
0110000101010	1000000001010	1000000001010	0101000011010	1000000001010	0100000000110
0110000100010	0110000101010	1000000101010	0101000001010	1000000101010	0000101000110
0110000000010	0110000100010	1000000001010	0101000001000	1000000001010	0000101000010
0100000000110	0110010000010	0101000011010	0100000001001	0101000011010	1000101001010
0000101000110	0100010000110	0101000011000	0100000001000	0101000001000	1000100001010
0000101000010	0000011000010	0101000001000	0100000101000	0100000001001	1000000001010
1000101001010	1000011001010	0100000001001	0100000001000	0100000101000	0110000101010
1000100001010	1000010001010	0100000001000	0100010001010	0100000001010	0110000100010
1000000001010	1000000001010	0100000101000	0000011001010	0000001001010	0110000000010
0110000101010	1000000101010	0100000001000	1000011001010	1000011001010	0100000000110
0110000100010	1000000001010	0100000001010	1000010001010	1000010001010	0000101000110
0110000000010	0101100011010	0000001001010	1000000001010	1000000001010	0000101000010
0100000000110	0101100001010	1000011001010	1000000101010	1000000101010	1000101001010
0000101000110	0101100001000	1000010001010	1000000001010	1000000001010	1000100001010
0000101000010	0100100001001	1000000001010	0101000011010	0101100011010	1000000001010
1000101001010	0100100101000	1000000101010	0101000001010	0101100001010	0110000101010
1000100001010	0100100001010	1000000001010	0101000001000	0101100001000	0110000100010
1000000001010	0000101001010	0101100011010	0100000001001	0100100001001	0110000000010
0110000101010	1000101001010	0101100001010	0100000001000	0100100101000	0010001000010
0110000100010	1000100001010	0101100001000	0100000101000	0100100001010	0000001000110
0110010000010	1000000001010	0100100001001	0100000001000	0000101001010	0000001000010
0100010000110	0110000101010	0100100101000	0100010001010	1000101001010	0000001001010
0000011000010	0110000100010	0100100001010	0000011001010	1000100001010	0000101001010
1000011001010	0110000000010	0000101001010	1000011001010	1000000001010	1000101001010
1000010001010	0100000000110	1000101001010	1000010001010	0110000101010	1000100001010
1000000001010	0000101000110	1000100001010	1000000001010	0110000100010	1000000001010
1000000101010	0000101000010	1000000001010	1000000101010	0110000000010	0110000101010
1000000001010	1000101001010	0110000101010	1000000001010	0010001000010	0110000100010
0101100011010	1000100001010	0110000100010	0101000011010	0000001000110	0110000000010
0101100001010	1000000001010	0110010000010	0101010001010	0000001000010	0100000000110
0101100001000	0110000101010	0100010000110	0101010001000	0000001001010	0000101000110
0100100001001	0110000100010	0000011000010	0100010001001	1000101001010	0000101000010
0100100101000	0110010000010	1000011001010	0100010101000	1000100001010	1000101001010

Figure 3.10 I/O vector sequence for the *Example 3.2*

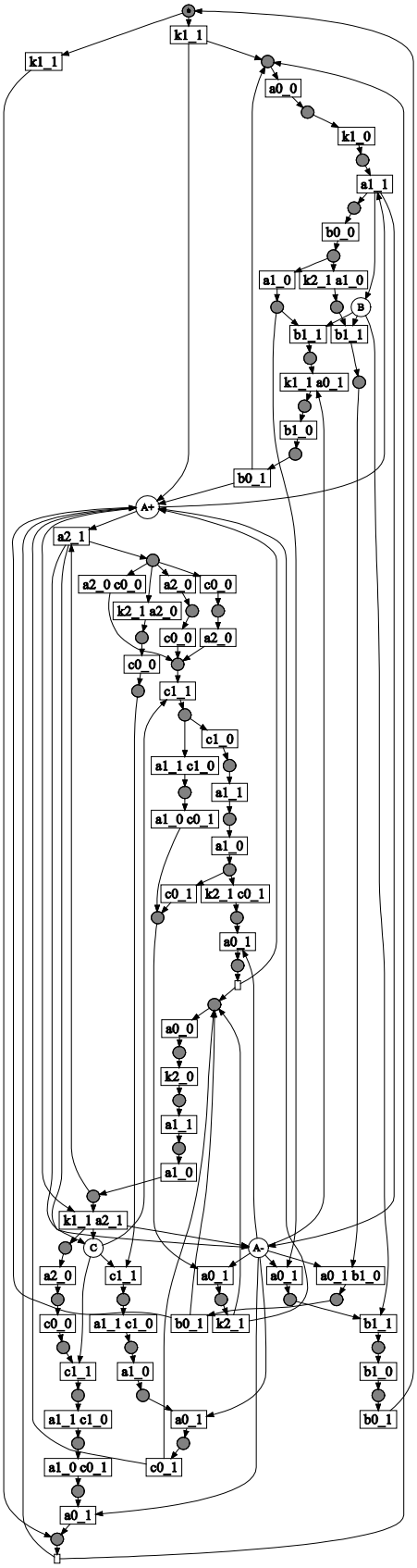


Figure 3.11 Identified IPN model of *Example 3.2*

Notice that in this IPN model there are numerous paths formed by non observable places. This is due to the observation of input changes that do not affect the outputs, but maybe they affect the controller state. In order to obtain a more compact model a

simplification strategy is applied [Estrada, 2010b]. It consists in merging several places, representing internal behaviour whose detected events do not have effect on the outputs, into a single one where an output event must occur. Consider the following I/O vector sequence involving one input x and two outputs A, B:

$$\left[\begin{array}{c} x \\ \hline A \\ B \end{array}\right] \qquad \left[\begin{array}{c} 0 \\ \hline 1 \\ 0 \end{array}\right]\left[\begin{array}{c} 1 \\ \hline 1 \\ 0 \end{array}\right]\left[\begin{array}{c} 1 \\ \hline 0 \\ 1 \end{array}\right]$$

This sequence can be represented as: $A \xrightarrow{x\_1} A \xrightarrow{\varepsilon} B$, which can be compacted as: $A \xrightarrow{x\_1} B$. This can be generalized to: $A \xrightarrow{e_i} A \xrightarrow{e_j} \ldots A \xrightarrow{e_k} B \cong A \xrightarrow{e_i e_j \ldots e_k} B$. The application of this simplification procedure, yields the IPN model showed in Figure 3.12.

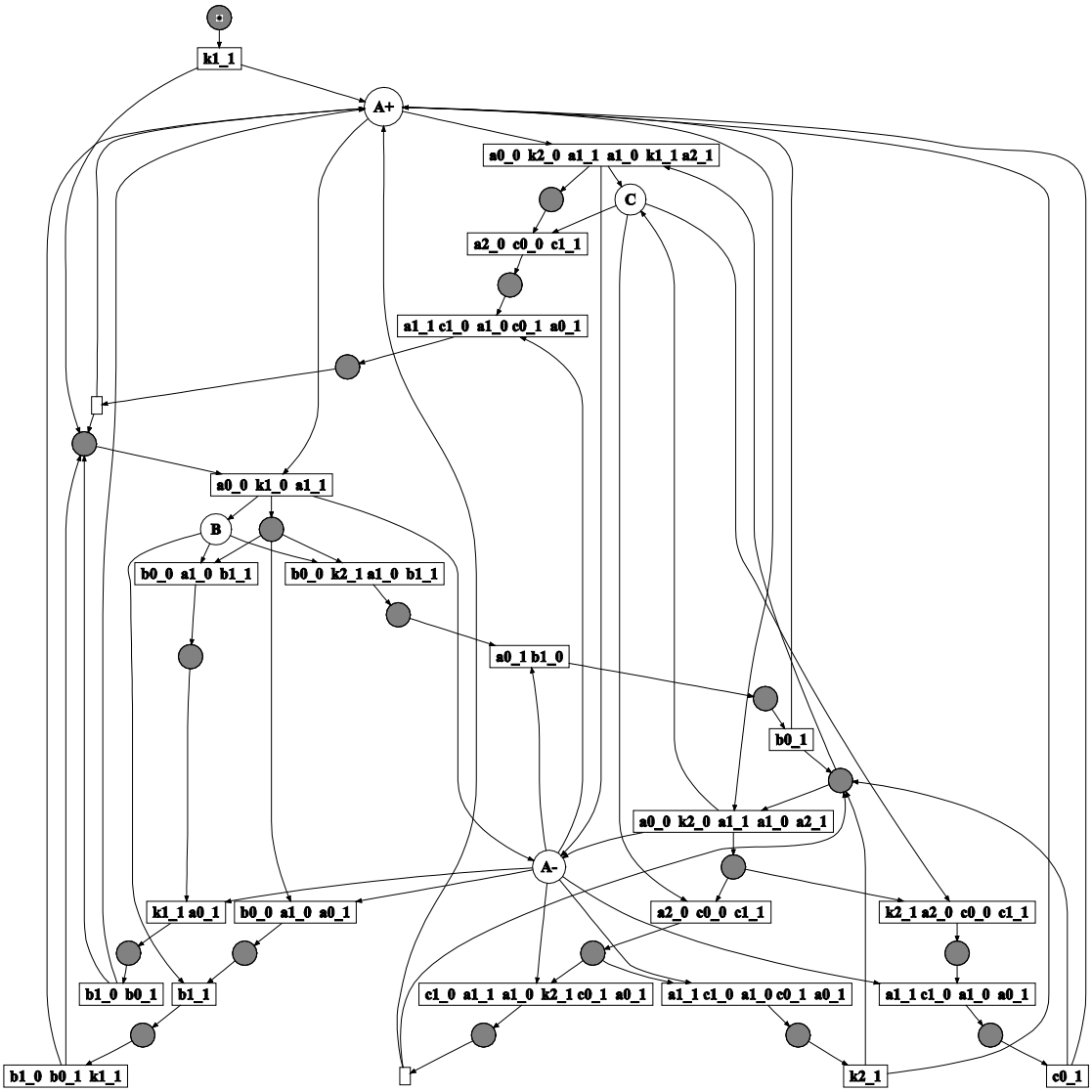


Figure 3.12 Obtained IPN model for the *Example 3.2* with defined methodology

In the model, we can observe some situations concerning the input and output information:

1. All the observed input value changes are included into the model. Some of those changes make part only of the evolution of the plant and do not have a direct influence on the evolution of the outputs (the signals to the actuators), and thus

on the evolution of the system. Including all input value changes yields to long transition paths and an apparent relationship between inputs and outputs, which is inexistent. For the *Example 3.2*, the input signals falling edge is not important in the evolution of the outputs. However, they are present in the obtained model and even observed into transitions producing output changes (Figure 3.13).

2. When the triggering of an output event is conditioned to several input values, such values could appear in different orders. All of the orders are shown within the model. In the *Example 3.2*, all of the cylinders have to return to their initial position and a piece has to arrive in order to start with the sorting of a parcel; that is, the cylinder A cannot be extended (A+=1) until a piece arrives (k1=1 or k2=1) and cylinder A is on its initial position (a0=1) and cylinder B is on its initial position (b0=1) and cylinder C is on its initial position (c0=1). But the changes of input values to accomplish this condition arrive in different orders, yielding to several paths shown into the model (Figure 3.14).
3. Some input values having influence on certain output evolutions can evolve at any part of the cycle of the system. Since those evolutions are showed into the model into a path at the moment they are observed, their influence on the outputs is not represented directly. In the *Example 3.2*, parcels can arrive at any moment while another parcel is being sorted. Then, several apparitions of k1_1 and k2_1 are over the entire model, and their relation with the starting of a cycle (the extension of cylinder A) is not directly represented (Figure 3.15).

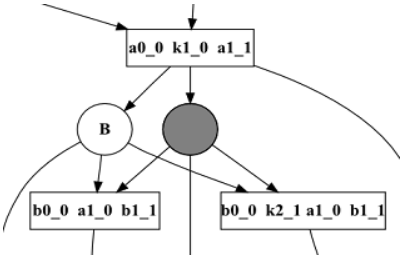


Figure 3.13 Useless input changes into the model

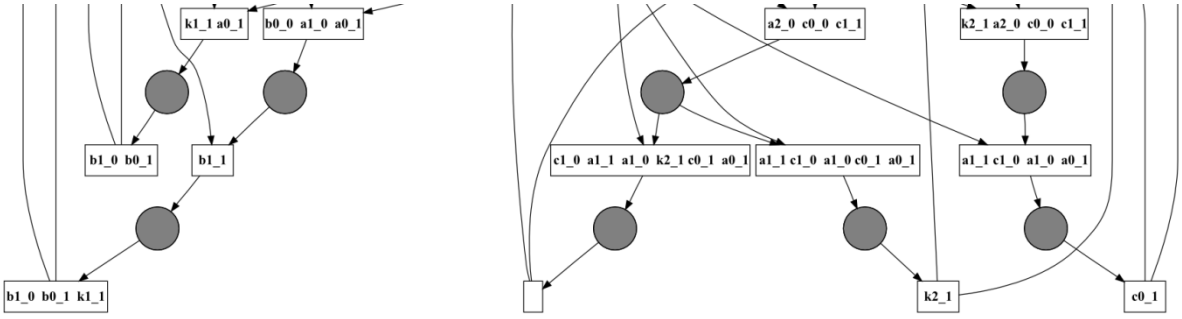


Figure 3.14 Several input evolution paths leading to the same input condition

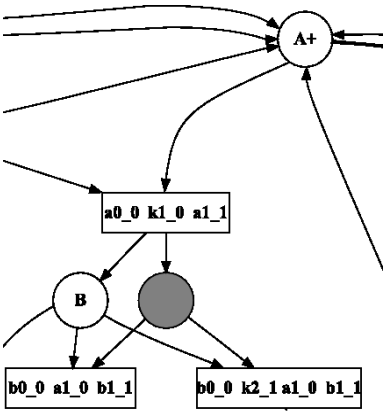


Figure 3.15 k2_1 may be observed at any moment

The evolution of the outputs is conditioned to input values and the occurrence of input events.

Example 3.3 Consider the classical illustrative example of two cars going to the right and returning (Figure 3.16). There are 5 input signal sensors: a, b, c, d and m and 4 output signals to the actuators: R1, L1, R2, L2.

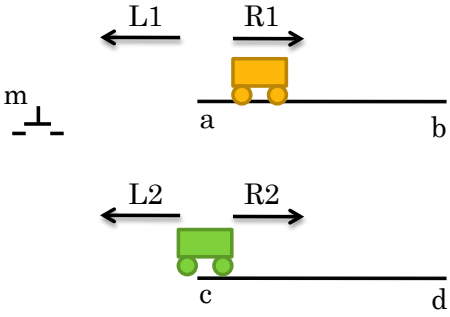


Figure 3.16 Cars going to the right and returning

The cars are initially at the leftmost position and when the signal m is given (m=1), both cars are requested to go to the right (R1_1 and R2_1). Once a car arrives to the rightmost position (b=1 or d=1), it is ordered to go back (R1_0 and L1_1 or R2_0 and L2_1). When a car arrives again to the leftmost position (a=1 or c=1), it has to wait until the other car arrives and the signal m is given to start another cycle.

As it can be seen, a condition on the inputs (a=1 and c=1) must be given in order to start the evolution of the system (R1_1 and R2_1) as well as an event trigger (m_1). Before concurrence transformations, the application of the proposed methodology would obtain the model in Figure 3.17 (a).

A clearer model is in Figure 3.17(b), in which input and output relation is showed explicitly and directly and which does not include input changes having not influence on the evolution of an output. We would like to find such a relation and take advantage of the IPN capabilities to do this. The desired causality between inputs and outputs of the system is not always easily found within the I/O sequence. However, the representation of causality can be easily done using IPN from the *Definition A.4*, but considering that a conjunction of events may provoke an output change, instead of a single event.

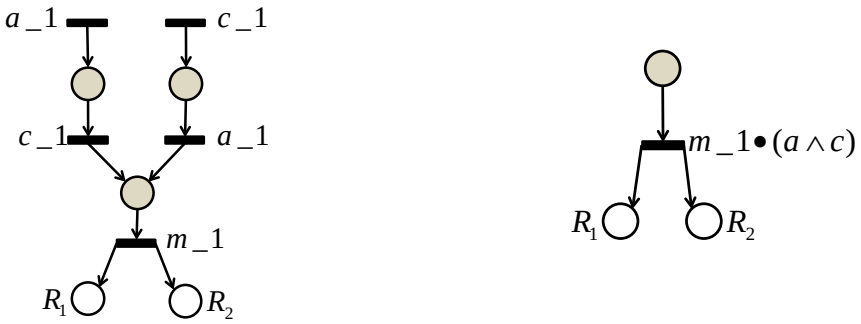


Figure 3.17 Different IPN models representing the same condition

3.3. Conclusion

In this chapter we have proposed an identification method for discrete event manufacturing systems that are automated through a programmable logic controller (PLC). The method follows a black-box and passive identification approach; it allows building stepwise an interpreted Petri net (IPN) model in polynomial time which describes in a detailed way the reactive behaviour of the controller. Several characteristics of the identified models have been analyzed in order to point out some limitations of the method. In next chapter, for the same identification problem statement, we propose a methodology to construct a compacter IPN model by inferring the input-output causality though a pre-analysis of the I/O sequence.

Chapter 4

A statistical identification method

Abstract. This chapter presents an alternative identification method that builds compact and expressive IPN models, describing clearly causal and concurrency relationships. The method is based on a statistical approach, which allows the analysis of very long I/O sequences issued from the execution of repetitive tasks performed by industrial systems.

4.1. General description

4.1.1. Motivation

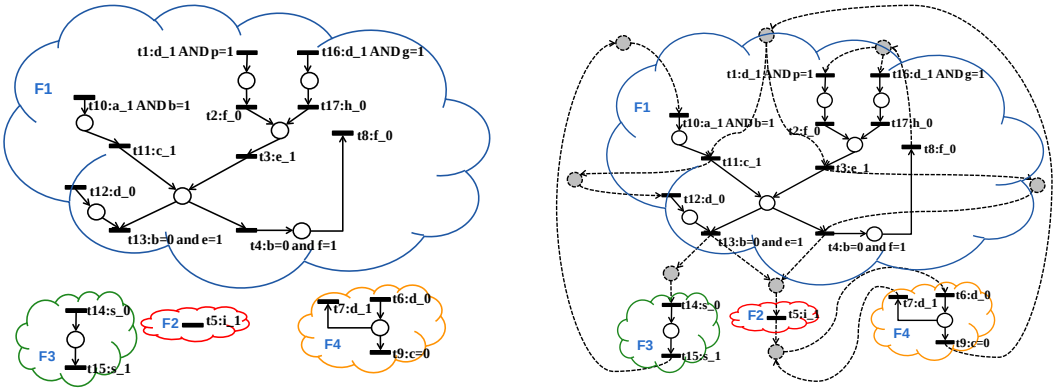
Although the method previously proposed in Chapter 3 is scalable due to the efficiency of the algorithms and the models represent the observed language, the size of the obtained models grows in proportion to the system size and complexity, especially when it includes parallelism. In industrial systems repetitive tasks are observed as similar but the sub-sequences of the measured I/O vectors are not identical; using the stepwise method they are represented within the model as different paths. This leads to internal model structures close to state machines, whose included information is not enough rich for determining a concurrency relationship of the involved transitions, and then few reductions can be applied to the IPN model.

As discussed before, our purpose in this research is not only to compute a model in which the observed sequence is reproducible, but also to achieve expressivity and compactness in the identified IPN model (see Appendix A, *Definition A.4*) allowing representing causal and concurrency relationships of the involved operations. In order to reach this goal we have conceived a two-step method based on the analysis of the relationships between inputs and outputs along the observed behaviour represented by the I/O-sequence *w*. These relationships are discovered by analyzing the relative frequency of outputs changes with respect to input changes; such relationships are stated in terms of conditional probability. This is the reason why the approach is called statistical.

4.1.2. Overview

The two steps of the method are the following:

- **Step1. Discovering the reactive input-output behaviour.** The observable part of the IPN is built, consisting of connected sub-graphs named fragments, composed by observable places and transitions labelled with algebraic expressions of input variables (Figure 4.1.a).
- **Step2. Determining the non-observable part of the IPN.** The sequence *w* is transformed into a sequence *S* of transitions created in the first step of the method; this sequence is processed for obtaining causal and concurrency relationships useful for determining the non-observable places that relate the fragments such that *S* (thus *w*) can be executed (Figure 4.1b).



a) IPN fragments b) Assembled fragments

Figure 4.1The two steps of the identification technique

4.1.3. Event types

As stated in the previous method, a sequence of observed event vectors $E = E(1)E(2)\ldots E(j)\ldots$ is derived from the I/O word w, such that $E(j)$ is the variation between two consecutive I/O vectors: $E(j) = w(j+1) - w(j)$. Each event vector can be decomposed into input and output event vectors:

$$E(j) = \begin{bmatrix} IE(j) \\ OE(j) \end{bmatrix}, \text{ where } IE(j) = \begin{bmatrix} IE_0(j) \\ IE_1(j) \\ \vdots \\ IE_{m-1}(j) \end{bmatrix} \text{ and } OE(j) = \begin{bmatrix} OE_0(j) \\ OE_1(j) \\ \vdots \\ OE_{n-1}(j) \end{bmatrix}$$

Regarding input and output event vectors and the PLC cycle described in section 2.1.1, there exist four situations (types) between consecutive I/O vectors that could be observed, which are explained by different occurring phenomena:

Type 1. $IE(j) \neq 0$ and $OE(j) \neq 0$

An input change has provoked directly an output change, and consequently, a state evolution. The I/O reactive causality is observed at the same PLC cycle.

Type 2. $IE(j) = 0$ and $OE(j) \neq 0$

The controller has arrived at step j-1 to a state in which, given the input values, an output (and state) evolution is allowed at step j.

Type 3. $IE(j) \neq 0$ and $OE(j) = 0$

a) $X(j-1) \neq X(j)$ *An input evolution has provoked a non-observable state evolution of the controller.*

b) $X(j-1) = X(j)$ *It has occurred an input evolution to which the controller is not sensitive.*

Type 4. $IE(j) = 0$ and $OE(j) = 0$

a) $X(j-1) \neq X(j)$ *It has occurred a non-observable state evolution of the controller which is not exhibited by any input nor output change.*

b) $X(j-1) = X(j)$ *The controller remains in a stable state, i.e., no state evolution condition is satisfied.*

Since situations *Type 1* and *Type 2* are directly observable by an output change, they can be straightforwardly represented in an IPN. Such a modelling will be performed by the first step of our algorithm.

The *Type 1* situation represents a direct input-output reactive behaviour, and thus the modelling is quite easy and similar to the procedure described in chapter 3: the input change is associated to the label of a transition and the output change is represented as arcs relating such a transition with the observable places representing outputs involved. The *Type 2* situation is a little more complex, since the input values which lead to the output evolution are not observed at the same PLC cycle (i.e. at the same event vector). To model such a behaviour, the step 1 of our algorithm will look into the context (the values of the inputs) in which the output changes occur. In this case, the output change will be modelled as described in the *Type 1* situation, but instead of an input change, an input condition will be associated to the label of the corresponding transition.

Thus, at the end of the step 1, a set of IPN fragments (connected sub-nets of the IPN) will be constructed as those shown in Figure 4.1.a. The initial I/O vector sequence *w* can be translated into a firing sequence of the transitions in the IPN fragments. The fragments, as well as the transition sequence will serve as input for the second step of the algorithm.

The *Type 3* situation is divided into two, depending on whether or not there is an internal state evolution of the controller. Situation *Type 3.a* is the case of the input events which provoke internal state evolutions and eventually lead to an output event of *Type 2*. Such internal evolutions cannot be directly computed, but can be inferred. By looking in the sequence built in Step 1, the order in which transitions appear can be determined. Such internal state inference will be performed by the second step of our algorithm and will be modelled by the addition of non observable places assuring the order of the transition firings, as illustrated in Figure 4.1.b.

In the situation *Type 3.b* there is no internal state evolution, and thus there is nothing to be inferred, as well as the situation *Type 4*, where there are neither input nor output events occurring in a PLC cycle. Notice that in this work we can only infer internal state evolutions by means of transition firing order. Other type of internal evolutions, such as timers or counters, is out of the scope of this work. We can now make the description of the two identification steps.

4.2. Computing the observable behaviour

4.2.1. Outline of the Step 1

The main sub-steps of the method for computing the observable part of the IPN model are given below. Accurate descriptions of such steps as well as the used notation are detailed along the section.

Algorithm 4.1 General description of the Step 1

Input: I/O sequence *w*

Output: Observable incidence matrix φC, labelling transition function λ and transition sequence *S*

---

1. Analyze sequence *w* in order to

- Compute events vector sequence and elementary events
- Compute Direct and Indirect Causality Matrices
- Construct Output Event Firing Functions from matrices
- Find Input events with differed influence

2. Use *w*, Firing Functions and Events with differed influence in order to

- Compute transitions of the IPN and their labelling λ
- Compute observable incidence matrix φC
- Compute transition sequence *S*

4.2.2. Elementary events

In order to discover the relationship between inputs and outputs of the system, we can start by computing at each event vector which are the specific changes occurring, that is, the input and output signals which have changed their value.

Definition 4.1 A rising input (output) event of the input I_i (output O_i) occurs when $IE_i(j)=1$ ($OE_i(j)=1$) and it can be denoted as I_i_1 (O_i_1). A falling input (output) event of the input I_i (output O_i) occurs when $IE_i(j)=-1$ ($OE_i(j)=-1$) and it can be denoted as I_i_0 (O_i_0).

We can decompose then each input (output) event vector as a conjunction of elementary input (output) events, which from now on we can call simply events:

$$IE(j)=IE_{j1}\bullet IE_{j2}\bullet ...=\prod IE_{ji} \text{ such that } I_{ji}(j+1)-I_{ji}(j)\neq 0$$

$$OE(j)=OE_{j1}\bullet OE_{j2}\bullet ...=\prod OE_{ji} \text{ such that } O_{ji}(j+1)-O_{ji}(j)\neq 0$$

If no input (output) event occurs in $E(j)$, we denote it as $IE(j)=\varepsilon$ ($OE(j)=\varepsilon$).

Similarly, we can represent each input vector as a conjunction of Boolean variables, depending on the values of each component of the input vector:

$$I(j)=I_{j1}\bullet I_{j2}\bullet ...=\prod I_{ji} \text{ such that } I_{ji}=\begin{cases} I_i=1 \text{ if } I_i(j)=1 \\ I_i=0 \text{ if } I_i(j)=0 \end{cases}$$

If the system has m inputs, n outputs and the length of the sequence w is h, the complexity of computing the events is O((*m* + *n*)*h*).

Example 4.1. Consider the next I/O sequence representing the 8 first I/O observed vectors during an evolution of the application presented in Figure 3.9.

$$w=\begin{array}{c} k1 \\ k2 \\ a0 \\ a1 \\ a2 \\ b0 \\ b1 \\ c0 \\ c1 \\ A+ \\ A- \\ B \\ C \end{array}\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

From the I/O sequence, we can compute the observed event sequence *E*:

$$E=\begin{array}{c} k1 \\ k2 \\ a0 \\ a1 \\ a2 \\ b0 \\ b1 \\ c0 \\ c1 \\ A+ \\ A- \\ B \\ C \end{array}\begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We can re-write *w* and *E* mixed for a clearer explanation:

$$
\begin{array}{c}
\begin{matrix} k1 \\ k2 \\ a0 \\ a1 \\ a2 \\ b0 \\ b1 \\ c0 \\ c1 \\ A+ \\ A- \\ B \\ C \end{matrix}
\end{array}
w = \begin{bmatrix} 0\\0\\1\\0\\0\\1\\0\\1\\0\\0\\0\\0\\0 \end{bmatrix}
\xrightarrow{E(1)} \begin{bmatrix} 1\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0 \end{bmatrix}
\begin{bmatrix} 1\\0\\1\\0\\0\\1\\0\\1\\0\\1\\0\\0\\0 \end{bmatrix}
\xrightarrow{E(2)} \begin{bmatrix} 0\\0\\-1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}
\begin{bmatrix} 1\\0\\0\\0\\0\\1\\0\\1\\0\\1\\0\\0\\0 \end{bmatrix}
\xrightarrow{E(3)} \begin{bmatrix} -1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}
\begin{bmatrix} 0\\0\\0\\0\\0\\1\\0\\1\\0\\1\\0\\0\\0 \end{bmatrix}
\xrightarrow{E(4)} \begin{bmatrix} 0\\0\\0\\1\\0\\0\\0\\0\\-1\\1\\1\\0 \end{bmatrix}
\begin{bmatrix} 0\\0\\0\\0\\0\\1\\0\\1\\0\\1\\1\\0\\0 \end{bmatrix}
\xrightarrow{E(5)} \begin{bmatrix} 0\\0\\0\\0\\0\\-1\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}
\begin{bmatrix} 0\\0\\0\\1\\0\\0\\0\\1\\0\\1\\1\\0\\0 \end{bmatrix}
\xrightarrow{E(6)} \begin{bmatrix} 0\\0\\0\\-1\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}
\begin{bmatrix} 0\\0\\0\\0\\0\\0\\0\\1\\0\\1\\1\\0\\0 \end{bmatrix}
\xrightarrow{E(7)} \begin{bmatrix} 0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\-1\\0\\0 \end{bmatrix}
\begin{bmatrix} 0\\0\\0\\0\\0\\0\\1\\1\\0\\0\\1\\0\\0 \end{bmatrix}
$$

The decomposition of the event vectors into elementary events is showed at Table 4.1.

Event vector	Elementary input events	Elementary output events
***E*(1)**	***IE*(1) = *k*1_1**	***OE*(1) = *A*+_1**
***E*(2)**	***IE*(2) = *a*0_0**	***OE*(2) = *ε***
***E*(3)**	***IE*(3) = *k*1_0**	***OE*(3) = *ε***
***E*(4)**	***IE*(4) = *a*1_1**	***OE*(4) = *A*+_0 • *A*–_1 • *B*_1**
***E*(5)**	***IE*(5) = *b*0_0**	***OE*(5) = *ε***
***E*(6)**	***IE*(6) = *a*1_0**	***OE*(6) = *ε***
***E*(7)**	***IE*(7) = *b*1_1**	***OE*(7) = *B*_0**

Table 4.1 Elementary events list for *Example 4.1*

4.2.3. Output Event Firing Functions

From the elementary events decomposition, the *Type 1* situation described before is clearly observed. It can be noticed that triggering of elementary output events is conditioned to the occurrence of certain elementary input events (for example, the elementary output event *B*_0 seems to be conditioned to the occurrence of *b*1_1). Also the situation *Type 2* is observed, i.e. the occurrence of an output event is conditioned to the presence of certain input levels (for example, the occurrence of *A*+_1 seems to be conditioned to *a*0=1).

Definition 4.2 An output event firing function (OEFF) χ states sufficient conditions for the occurrence of the output event OE_k. It is defined as:

$$\chi(OE_k)=G(OE_k)\bullet F(OE_k)$$

where:

$G:OE\rightarrow 2^{IE}$ is the sufficient input event combination to produce the output event OE_k.

$F:OE\rightarrow 2^{IL}$ is the sufficient input level condition to produce the output event OE_k.

Example 4.2. In the *Example 3.3*, the firing condition for the output event R_1_1 is:

$\chi(R_1\_1) = G(R_1\_1) \bullet F(R_1\_1) = m\_1 \bullet (a \wedge c)$

4.2.4. Finding causality

As explained in section 2.1.2, it could happen that at the same PLC cycle, an input event and an output event are observed, but it does not necessarily imply that such input and output events are related. In order to find $G(OE_k)$ to represent situations of *Type 1*, we analyze the relative frequency of the simultaneous emergence of an input event IE_i and an output event OE_k, with respect to the emergence of OE_k during the whole sequence of events. That relationship can be naturally expressed as the conditional probability of the occurrence of an output event OE_k, given that a certain input event IE_i has occurred at the same PLC cycle:

$$Prob(OE_k \mid IE_i) = \frac{Observ(OE_k, IE_i)}{Observ(OE_k)}$$

where $Observ(OE_k,IE_i)$ is the number of times that the elementary output event OE_k and the elementary input event IE_i are observed the same event (which implies that they occurred at the same PLC cycle) in the event sequence E, and $Observ(OE_k)$ is the number of times the elementary output event OE_k occurs in the event sequence E.

In a similar way, to find situations *Type 2* represented by $F(OE_k)$, we will compute the occurrence probability of an output event at a PLC cycle, given that certain input level is observed:

$$Prob(OE_k \mid I_i = x) = \frac{Observ(OE_k, I_i = x)}{Observ(OE_k)}, x \in \{1,0\}$$

where $Observ(OE_k,I_i = x)$ is the number of times the elementary output event OE_k occurred when the input level $I_i = x$ was present.

Conditional probabilities have been used in [Cook, 2004] for analyzing the relative occurrence between workflow events; this analysis is done in the first step of the procedure. However the remaining steps and the kind of obtained model differ from that of our method.

In order to compute previously defined probabilities, we can use counters storing the respective number of occurrences used at each equation. The values will be stored in two matrices, called Direct I/O Causality Matrix (*DCM*) and Indirect I/O Causality Matrix (*ICM*).

Algorithm 4.2 Building DCM and ICM matrices

Input: *w*,*E*

Output: *DCM*, *ICM*

1) Initialize counters

   $\forall OE_k$, $Observ(OE_k) \leftarrow 0$

   $\forall IE_i \; \forall OE_k$, $Observ(OE_k \text{ and } IE_i) \leftarrow 0$

   $\forall IL_i \; \forall OE_k$, $Observ(OE_k \text{ and } I_i = x) \leftarrow 0$

2) Update counters

$\forall E(j)$

If $OE_k \in OE(j)$, $Observ(OE_k) \leftarrow Observ(OE_k) + 1$

If $IE_i \in IE(j)$ and $OE_k \in OE(j)$, $Observ(OE_k \text{ and } IE_i) \leftarrow (OE_k \text{ and } IE_i) + 1$

If $I_i = x$ in $w(j)$ and $OE_k \in OE(j)$, $Observ(OE_k \text{ and } I_i = x) \leftarrow (OE_k \text{ and } I_i = x) + 1$

3) Compute matrices:

$$DCM_{i,k} = Prob(OE_k \mid IE_i)$$

$$ICM_{i,k} = Prob(OE_k \mid I_i = x)$$

Example 4.3. Considering the whole I/O sequence (Figure 4.2) of the *Example 4.1*, we have computed the matrices shown in Tables 4.2 and 4.3.

Each column of the *DCM* corresponds to an output event OE_k; each row corresponds to an input event IE_i. If the value in cell $DCM_{i,k}$ is not 0.000, it means that IE_i and OE_k were observed at least once occurring at the same PLC cycle. Such an occurrence could be due to the fact that there is a causality relationship between the input and the output; however, as explained in Chapter 2, it could be also due to the PLC cyclic execution mode.

0000001001010	0100100001010	0100010000110	1000010001010	0100010001000	1000000001010
1000101001010	0000101001010	0000011000010	1000000001010	0000011001000	0110000101010
1000100001010	1000101001010	1000011001010	1000000101010	1000011001010	0110000100010
1000000001010	1000100001010	1000010001010	1000000001010	1000010001010	0110000000010
0110000101010	1000000001010	1000000001010	0101000011010	1000000001010	0100000000110
0110000100010	0110000101010	1000000101010	0101000001010	1000000101010	0000101000110
0110000000010	0110000100010	1000000001010	0101000001000	1000000001010	0000101000010
0100000000110	0110010000010	0101000011010	0100000001001	0101000011010	1000101001010
0000101000110	0100010000110	0101000011000	0100000001000	0101000001000	1000100001010
0000101000010	0000011000010	0101000001000	0100000101000	0100000001001	1000000001010
1000101001010	1000011001010	0100000001001	0100000001000	0100000101000	0110000101010
1000100001010	1000010001010	0100000001000	0100010001010	0100000001010	0110000100010
1000000001010	1000000001010	0100000101000	0000011001010	0000001001010	0110000000010
0110000101010	1000000101010	0100000001000	1000011001010	1000011001010	0100000000110
0110000100010	1000000001010	0100000001010	1000010001010	1000010001010	0000101000110
0110000000010	0101100011010	0000001001010	1000000001010	1000000001010	0000101000010
0100000000110	0101100001010	1000011001010	1000000101010	1000000101010	1000101001010
0000101000110	0101100001000	1000010001010	1000000001010	1000000001010	1000100001010
0000101000010	0100100001001	1000000001010	0101000011010	0101100011010	1000000001010
1000101001010	0100100101000	1000000101010	0101000001010	0101100001010	0110000101010
1000100001010	0100100001010	1000000001010	0101000001000	0101100001000	0110000100010
1000000001010	0000101001010	0101100011010	0100000001001	0100100001001	0110000000010
0110000101010	1000101001010	0101100001010	0100000001000	0100100101000	0010001000010
0110000100010	1000100001010	0101100001000	0100000101000	0100100001010	0000001000110
0110010000010	1000000001010	0100100001001	0100000001000	0000101001010	0000001000010
0100010000110	0110000101010	0100100101000	0100010001010	1000101001010	0000001001010
0000011000010	0110000100010	0100100001010	0000011001010	1000100001010	0000101001010
1000011001010	0110000000010	0000101001010	1000011001010	1000000001010	1000101001010
1000010001010	0100000000110	1000101001010	1000010001010	0110000101010	1000100001010
1000000001010	0000101000110	1000100001010	1000000001010	0110000100010	1000000001010
1000000101010	0000101000010	1000000001010	1000000101010	0110000000010	0110000101010
1000000001010	1000101001010	0110000101010	1000000001010	0010001000010	0110000100010
0101100011010	1000100001010	0110000100010	0101000011010	0000001000110	0110000000010
0101100001010	1000000001010	0110010000010	0101010001010	0000001000010	0100000000110
0101100001000	0110000101010	0100010000110	0101010001000	0000001001010	0000101000110
0100100001001	0110000100010	0000011000010	0100010001001	1000101001010	0000101000010
0100100101000	0110010000010	1000011001010	0100010101000	1000100001010	1000101001010

Figure 4.2 The I/O vector sequence for the *Example 4.1*

	A+_1	A+_0	A-_1	A-_0	B_1	B_0	C_1	C_0
*k*1_1	0.111	0.111	0.111	0.111	0.000	0.200	0.000	0.000
*k*1_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*k*2_1	0.222	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*k*2_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*a*0_1	0.222	0.000	0.000	1.000	0.000	0.000	0.000	0.000
*a*0_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*a*1_1	0.000	0.444	0.444	0.000	1.000	0.000	0.000	0.000
*a*1_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*a*2_1	0.000	0.556	0.556	0.000	0.000	0.000	1.000	0.000
*a*2_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*b*0_1	0.333	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*b*0_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*b*1_1	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
*b*1_0	0.000	0.000	0.000	0.111	0.000	0.000	0.000	0.000
*c*0_1	0.111	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*c*0_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*c*1_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
*c*1_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 4.2 Direct Causality matrix for the *Example 4.3*

	A+_1	A+_0	A-_1	A-_0	B_1	B_0	C_1	C_0
*k*1=1	0.444	0.111	0.111	0.333	0.000	0.250	0.200	0.200
*k*1=0	0.556	0.889	0.889	0.667	1.000	0.750	0.800	0.800
*k*2=1	0.556	0.000	0.000	0.333	0.000	0.250	0.000	0.200
*k*2=0	0.444	1.000	1.000	0.667	1.000	0.750	1.000	0.800
*a*0=1	1.000	0.000	0.000	1.000	0.000	0.500	0.000	0.000
*a*0=0	0.000	1.000	1.000	0.000	1.000	0.500	1.000	1.000
*a*1=1	0.000	0.444	0.444	0.000	1.000	0.000	0.000	0.000
*a*1=0	1.000	0.556	0.556	1.000	0.000	1.000	1.000	1.000
*a*2=1	0.000	0.556	0.556	0.000	0.000	0.000	1.000	0.000
*a*2=0	1.000	0.444	0.444	1.000	1.000	1.000	0.000	1.000
*b*0=1	1.000	1.000	1.000	0.556	0.000	0.000	1.000	1.000
*b*0=0	0.000	0.000	0.000	0.444	1.000	1.000	0.000	0.000
*b*1=1	0.000	0.000	0.000	0.111	1.000	1.000	0.000	0.000
*b*1=0	1.000	1.000	1.000	0.889	0.000	0.000	1.000	1.000
*c*0=1	1.000	1.000	1.000	0.889	0.000	1.000	1.000	0.000
*c*0=0	0.000	0.000	0.000	0.111	1.000	0.000	0.000	1.000
*c*1=1	0.000	0.000	0.000	0.000	1.000	0.000	0.000	1.000
*c*1=0	1.000	1.000	1.000	1.000	0.000	1.000	1.000	0.000

Table 4.3 Indirect Causality matrix for the *Example 4.3*

Consider the second column of the *DCM* corresponding to the event A+_0. The input candidates to be in a causal relation with A+_0 are *k*1_1, *a*1_1 and *a*2_1. Notice that *Prob*(A+_0|*a*1_1) + *Prob*(A+_0|*a*2_1) = 1. By considering the second column of the *ICM* matrix, we can verify that *Prob*(A+_0|*a*1=1) + *Prob*(A+_0|*a*2=1) = 1. We can conclude that the event A+_0 is sometimes caused by *a*1_1 and sometimes caused by *a*2_1.

Consider now the first column of the *DCM* matrix which corresponds to the output event A+_1. The input candidates to be in a causal relation with such an output event

are $k1\_1$, $k2\_1$, $a0\_1$, $b0\_1$ and $c0\_1$. Observe that $Prob(A+\_1|k1\_1)$ + $Prob(A+\_1|k2\_1)$ + $Prob(A+\_1|a0\_1)$ + $Prob(A+\_1|b0\_1)$ + $Prob(A+\_1|c0\_1) = 1$, but this time, by looking at the second column of the *ICM*, we observe that $Prob(A+\_1|k1=1)$ + $Prob(A+\_1|k2=1)$ + $Prob(A+\_1|a0=1)$ + $Prob(A+\_1|b0=1)$ + $Prob(A+\_1|c0=1) \neq 1$. However, observe that $Prob(A+\_1|k1=1) + Prob(A+\_1|k2=1) = 1$ and that $Prob(A+\_1|a0=1) = 1$, $Prob(A+\_1|b0=1) = 1$, and $Prob(A+\_1|c0=1) = 1$. Thus, we can conclude that the event A+_1 always occurs under conditions $a0 = 1$, $b0 = 1$, $c0 = 1$ and that it occurs sometimes under the condition $k1 = 1$ and sometimes under $k2 = 1$.

All conditions found intuitively above will be computed and included in the function χ of each output event. In the next section the procedure for obtaining such functions is formalized.

4.2.5. Determining the firing functions

As stated before, the occurrence of an input event (or a conjunction of input events, given the particular case of the PLC) is a sufficient condition to produce the occurrence of an elementary output event. Several conjunction combinations are possible, and thus we know that $G(OE_k)$, as a Boolean function, can be expressed as a conjunction of disjunctions:

$G(OE_k) = DisjE_r \wedge DisjE_s \wedge \ldots \wedge DisjE_t$

Such that $DisjE_j = (IE_w \vee IE_x \vee \ldots \vee IE_y \vee IE_z)$

If OE_k has occurred into an event $E(j)$, it seems that $G(OE_k)$ has been made true, which implies that every one of the terms in $DisjE_r \wedge DisjE_s \wedge \ldots \wedge DisjE_t$ was true. To make true $DisjE_j$, we need that at least one of $IE_w \vee IE_x \vee \ldots \vee IE_y \vee IE_z$ be true, which implies that at least one of the input events IE_w, IE_x, …, IE_y, IE_z has occurred at the same event than OE_k. Then, we can express the number of occurrences of OE_k as:

$$Observ(OE_k) =$$
$$Observ(IE_w \text{ and } OE_k) + \ldots + Observ(IE_z \text{ and } OE_k)$$
$$- Observ(IE_w \text{ and } IE_x \text{ and } OE_k) - \ldots - Observ(IE_y \text{ and } IE_z \text{ and } OE_k)$$
$$+ Observ(IE_w \text{ and } IE_x \text{ and } IE_y \text{ and } OE_k) + \ldots$$
$$- \ldots$$

Keeping track of all the terms in the expression would be very expensive in space. We restrict the "or" disjunctions to "exclusive or":

$DisjE_j = (IE_w \oplus IE_x \oplus \ldots \oplus IE_y \oplus IE_z)$, under the hypothesis that if an input event IE_x or an input event IE_y can provoke the firing of the output event OE_k, it was only provoked by IE_x or by IE_y, but never both together.

Following this restriction, we can express the number of occurrences of OE_k as:

$$Observ(OE_k) = Observ(IE_w \text{ and } OE_k) + \ldots + Observ(IE_z \text{ and } OE_k)$$

Dividing both terms by $Observ(OE_k)$:

$$1 = \frac{Observ(IE_w \text{ and } OE_k) + \ldots + Observ(IE_z \text{ and } OE_k)}{Observ(OE_k)}$$

and substituting each term:

$$1 = Prob(OE_k \mid IE_w) + ... + Prob(OE_k \mid IE_z)$$

We obtain a sum equal to 1 corresponding to terms in the *k*-th column of the *DCM*. Thus, we can express the input event function as:

$$G(OE_k) = \Pi DisjE_j$$

With $DisjE_j = (IE_x \oplus IE_y \oplus ... \oplus IE_z)$

Such that

1. $DCM_{xj} \neq 0, DCM_{yj} \neq 0, ... DCM_{zj} \neq 0$
2. $DCM_{xj} + DCM_{yj} + ... + DCM_{zj} = 1$

If no $DisjE_j$ is found, we write $G(OE_k) = \varepsilon$

Applying the same reasoning, we can find in the *ICM* the input level condition under which output events occur. The *k*-th column of the *ICM* matrix can help us then to distinguish those input signals which must be present or absent in the occurrence of OE_j:

$$F(OE_k) = \Pi DisjL_j,$$

with $DisjL_j = (IL_x \oplus IL_y \oplus ... \oplus IL_z)$,

such that

1. $ICM_{xj} \neq 0, ICM_{yj} \neq 0, ... ICM_{zj} \neq 0$
2. $ICM_{xj} + ICM_{yj} + ... + ICM_{zj} = 1$

If no $DisjL_j$ is found, we write $F(OE_k) = (=1)$

Some of the input levels on the sum can be considered as redundant. They are simply result of the conditions under which an output event is produced, but do not have a direct influence on the occurrence of the event. Consider the *Example 3.3* of the cars. Input levels b=0 and d=0 are always present at the occurrence of R1_1 and R2_1, but they are not directly related to such output events. In order to avoid representing these non-direct relations into our model, an additional condition can be added (or in fact, substitute condition 1):

3. $DCM_{xj} \neq 0, DCM_{yj} \neq 0, ... DCM_{yj} \neq 0$

Introducing such a condition, we could ignore some input events which do not have a direct effect on the output evolution. To avoid this, we can keep a list *D* of input events with differed influence to outputs. If *DCM* rows corresponding to I_i_1 and I_i_0 are zero, we add them to *D*. If all the entries of the rows corresponding to I_i=1 and I_i=0 are 1 and 0 respectively (or 0 and 1), it means that no output event occurs between rising event and falling of the input I_i, so we can add I_i =1 or I_i=0 (the one corresponding to the row with 1 values).

From the matrices of *Example 4.3*, the obtained functions are given in table 4.4. No input event is added to *D*.

OE_k	$G(OE_k)$	$F(OE_k)$	$\chi(OE_k)$
A+_1	(ε)	$((k1 \oplus k2) \wedge a0 \wedge b0 \wedge c0)$	$(\varepsilon)\bullet ((k1 \oplus k2) \wedge a0 \wedge b0 \wedge c0)$
A+_0	$(a1\_1 \oplus a2\_1)$	$(=1)$	$(a1\_1 \oplus a2\_1) \bullet (=1)$
A–_1	$((a1\_1 \oplus a2\_1))$	$(=1)$	$((a1\_1 \oplus a2\_1)) \bullet (=1)$
A–_0	$(a0\_1)$	$(=1)$	$(a0\_1) \bullet (=1)$
B_1	$(a1\_1)$	$(=1)$	$(a1\_1) \bullet (=1)$
B_0	$(b1\_1)$	$(=1)$	$(b1\_1) \bullet (=1)$
C_1	$(a2\_1)$	$(=1)$	$(a2\_1) \bullet (=1)$
C_0	$(c1\_1)$	$(=1)$	$(c1\_1) \bullet (=1)$

Table 4.4 Firing functions of *Example 4.3*

These functions state clearly the conditions on the inputs that modify the outputs. They can be expressed in terms of IPN as the marking/unmarking of observable places. Figure 4.3 represents pictorially the IPN fragments corresponding to the above functions.

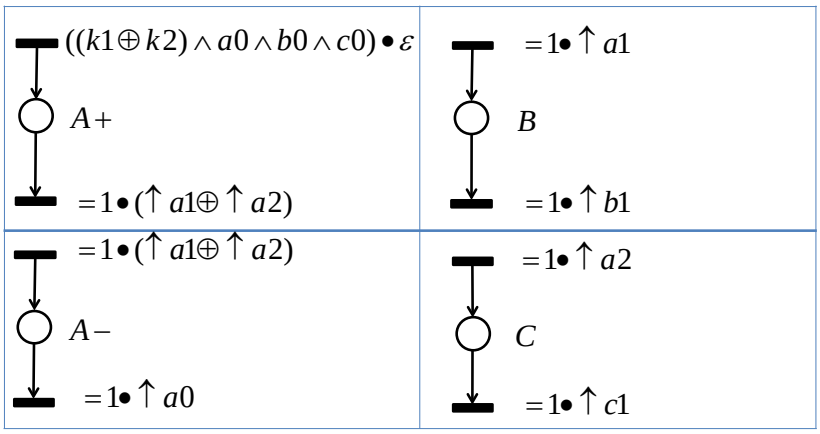


Figure 4.3 IPN fragments for *Example 4.3*

Notice that condition $DCM_{xj} \neq 0$, $DCM_{yj} \neq 0$,…, $DCM_{zj} \neq 0$ requires that the inputs related to the output change were observed at least once changing its value at the same PLC cycle for the considered output. This condition may be restrictive if the input-output reaction is not observed in the same event vector. For example, in order to avoid component damages, if there is not an input sensor to indicate that a pusher has been retracted, there may be some security temporizations which do not allow another actuator reacting at the moment an input condition has been satisfied.

In all previous cases, the input-output reaction would not be found and thus there may be an output event with empty conditions on its firing function. In order to find the correct *OEFF*, we can proceed in a similar way that in the previous chapter, where equivalent states were found at the observation of κ equal events. But in this case, the condition could be relaxed to consider input events which have been observed in previous event vectors instead of the same event vector. Formally, we can compute:

$$Prob(OE_k \mid IE_i) = \frac{Observ(OE_k, IE_i)}{Observ(OE_k)}$$

But this time, the computation is done by considering IE_i occurred at the previous event vector than the one with the occurrence of OE_k. In such a way, a matrix called One-step *DCM* matrix (1-*DCM*) is computed. A new *OEFF* can be computed using values of 1-*DCM*. If the computed *OEFF* has still empty conditions, we can take the previous to the previous event vector constructing a 2-*DCM* matrix and successively until the κ-*DCM* matrix while empty conditions are computed. The initial *DCM* matrix can be thus denoted as 0-*DCM*.

In the example of this section, such relaxing condition is not necessary, since, as it can be noticed, no empty conditions have been computed. However, in one of the experimental case studies of Chapter 5, such a technique is applied.

The previously described method for finding input-output causality can be summarized as follows.

Algorithm 4. 3Finding firing functions

Input:*DCM* and *ICM*

Output: $\chi(OE_k)$

$\forall\ OE_k$

1) Make $G(OE_k) \leftarrow \varepsilon$
   Compute $G(OE_k) \leftarrow \Pi DisjE_j$

   With $DisjE_j \leftarrow (IE_x \oplus IE_y \oplus ... \oplus IE_z)$

   Such that

   1. $DCM_{xj} \neq 0, DCM_{yj} \neq 0, ... DCM_{zj} \neq 0$
   2. $DCM_{xj} + DCM_{yj} + ... + DCM_{zj} = 1$

2) Make $F(OE_k) \leftarrow (=1)$, $\kappa \leftarrow 0$
   While ($G(OE_k) = \varepsilon$ and $F(OE_k) = (=1)$)
   Compute $F(OE_k) \leftarrow \Pi DisjL_j$

   With $DisjL_j = (IL_x \oplus IL_y \oplus ... \oplus IL_z)$

   Such that

   1. $\kappa - DCM_{xj} \neq 0, \kappa - DCM_{yj} \neq 0, ... \kappa - DCM_{zj} \neq 0$
   2. $ICM_{xj} + ICM_{yj} + ... + ICM_{zj} = 1$

   Make $\kappa \leftarrow \kappa + 1$

Remark. Given that the system has *m* inputs and *n* outputs, there could be 2*m* values different from 0 in the DCM matrix. Thus, in the worst case, the complexity of the procedure for computing the firing functions is $O(n(2^{2m}))$. However, in practice we have very often observed that only few values differ from 0 in the DCM matrix; thus the complexity of the Algorithm 4.3 can be approximated to $O(n(2^6))$, that is $O(n)$; this is due to the fact that only a small subset of input events occur at the same PLC cycle than a given output event.

The accuracy of the firing functions depends (as in all of the identification methods) on the quality of information provided by the sequence: if a certain input event is related

with the occurrence of an output event, but they never occur close enough to detect the relationship, they will not be related in the final model.

4.2.6. Construction of the observable incidence matrix

Remind that $G(OE_k)$ and $F(OE_k)$ are expressed in a Conjunctive Normal Form in which disjunctions are "exclusive or". We have previously analysed that, to make true their values (implying the firing of OE_k), all of the disjunctions in the expressions must have a value true. Remind that under the given input conditions, several output events could be produced at the same PLC cycle.

We can check the I/O events sequence (and I/O sequence) to compute those conditions under which output events fired along it. We can represent those firing conditions by observable transitions in the IPN as shown in Figure 4.4.

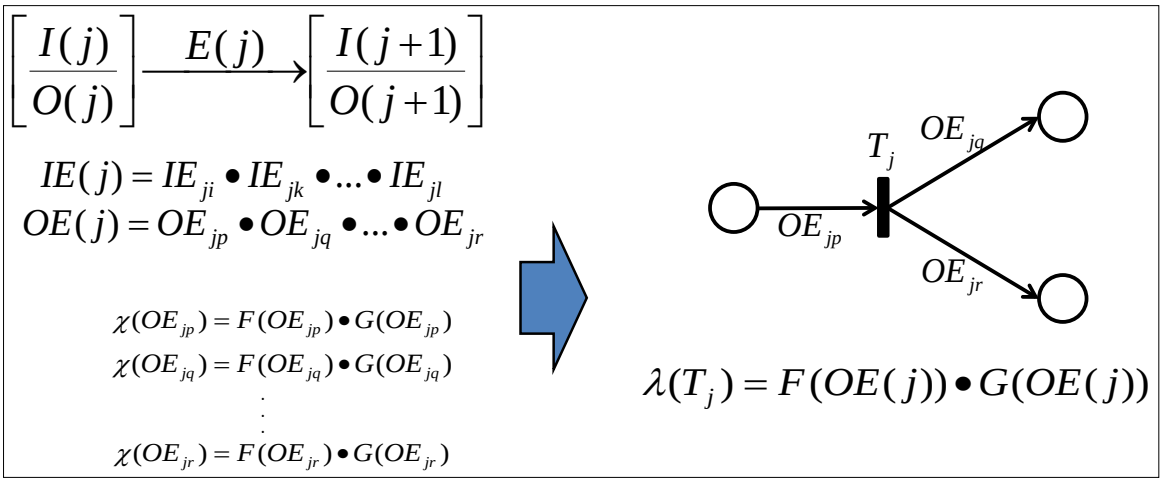


Figure 4.4. IPN representation of several output events enabled at the same cycle

We also will represent input events with differed output influence by transitions with no corresponding output change.

Computed transitions will form an incidence matrix corresponding to the observable part of the system. We can systematically describe the above procedures in the following algorithm.

Algorithm 4.4 Building observable behaviour

Input: I/O sequence w, Events sequence E, Differed inputs D, Firing functions $\chi(OE_k)$

Output: Observable incidence matrix φC, labelling transition function λ, sequence S

---

1) Create a row in the incidence matrix for every output of the system
2) $S \leftarrow \varepsilon$
3) $\forall E(j)$ Consider the I/O sequence and I/O events sequence:
   a) If $OE(j)=0$ and $IE(j)$ contains elementary input events $IE_s,\ldots,IE_u$ belonging to D
      - If it has not been created before, create a new zero transition T_j (a zero column in the incidence matrix φC) such that $\lambda(T_j) \leftarrow IE_s,\ldots,IE_u$
      - $S \leftarrow S \cdot T_j$
   b) If $OE(j) \neq 0$

- Consider the output events $OE(j) = OE_{jp} \bullet OE_{jq} \bullet OE_{jr}$ included in $E(j)$
- Compute a new firing function considering $\chi(OE_{jp})$, $\chi(OE_{jq})$, $\chi(OE_{jr})$:

i) For every $\chi(OE_{jk}) = G(OE_{jk}) \bullet F(OE_{jk})$:

(1) For every $DisjE_i = (IE_x \oplus IE_y \oplus ... \oplus IE_z)$ in $G(OE_{jk})$ look into $IE(j)$ the input event IE_{ik} which has made true $DisjE_i = (IE_x \oplus IE_y \oplus ... \oplus IE_z)$ and make $DisjE_i' \leftarrow IE_{ik}$

(2) $G'(OE_{jk}) \leftarrow \Pi DisjE_i'$

(3) $G(OE(j)) \leftarrow G'(OE_{jp}) \bullet G'(OE_{jq}) \bullet ... \bullet G'(OE_{jr})$

(4) For every $DisjL_i = (IL_x \oplus IL_y \oplus ... \oplus IL_z)$ in $F(OE_{jk})$ look into $w(j + 1)$ the input levels IL_{ik} which have made true $DisjL_i = (IL_x \oplus IL_y \oplus ... \oplus IL_z)$ and make $DisjL_i' \leftarrow IL_{ik}$

(5) $F'(OE_{jk}) \leftarrow \Pi DisjL_i'$

(6) $F(OE(j)) \leftarrow F'(OE_{jp}) \bullet F'(OE_{jq}) \bullet ... \bullet F'(OE_{jr})$

- If it has not been created before, create a new transition T_j (a new column in the incidence matrix φC) such that $\lambda(T_j) \leftarrow F(OE(j)) \bullet G(OE(j))$ and relate it to its provoked output changes:

i) For all elementary output events in $OE(j) = OE_{jp} \bullet OE_{jq} \bullet OE_{jr}$, put a -1 into the line corresponding to OE_{jk} if it is a falling event, or a 1 if it is a rising event; for the rest of the lines, put a 0.

- $S \leftarrow S \cdot T_j$

It can be noticed that a simplified sequence transition S will be created by concatenating progressively each one of the computed transitions.

The complexity of the procedure for building the transition sequence and fragments is $O((n' \log m')h)$, where n' and m' are the maximum number of input and output elementary events in an event vector. Consequently, the *Algorithm 4.1* can be executed in polynomial time.

Property 4.1. The sequence transition S is a translation of the I/O sequence w into transition firings of the *PN*-fragments built by *Algorithm 4.4*.

Proof. It is easy to see that at the end of the steps 3.a and 3.b of *Algorithm 4.4*, S is formed by concatenating the computed transitions from the event sequence produced by w. This allows that the reactive behaviour can be reproduced in the partially created IPN model.♦

Example 4.4. After treating the long I/O vector sequence, we can compute the transitions in the fourth column of Table 4.4.

I/O vector	Elementary input events	Elementary output events	Computed transition
$w(1) = [00100101000000]^T$	$IE(1) = k1\_1$	$OE(1) = A+\_1$	$\chi(t_1) = (k1 \bullet a0 \bullet b0 \bullet c0) \bullet (\varepsilon)$
$w(2) = [10100101010000]^T$	$IE(2) = a0\_0$	$OE(2) = \varepsilon$	*No transition*
$w(3) = [10000101010000]^T$	$IE(3) = k1\_0$	$OE(3) = \varepsilon$	*No transition*
$w(4) = [00000101010000]^T$	$IE(4) = a1\_1$	$OE(4) = A+\_0 \bullet A-\_1 \bullet B\_1$	$\chi(t_2) = (=1) \bullet (a1\_1)$
$w(5) = [00010101001100]^T$	$IE(5) = b0\_0$	$OE(5) = \varepsilon$	*No transition*
$w(6) = [00010001001100]^T$	$IE(6) = a1\_0$	$OE(6) = \varepsilon$	*No transition*
$w(7) = [00000001001100]^T$	$IE(8) = k1\_1 \bullet a0\_1$	$OE(8) = A-\_0$	$\chi(t_3) = (=1) \bullet (a0\_1)$
$w(8) = [10100001000100]^T$	$IE(7) = b1\_1$	$OE(7) = B\_0$	$\chi(t_4) = (=1) \bullet (b1\_1)$
$w(9) = [10100011000000]^T$	$IE(9) = b1\_0$	$OE(9) = \varepsilon$	*No transition*
$w(10) = [10100001000000]^T$	$IE(10) = b0\_1$	$OE(10) = A+\_1$	$\chi(t_1) = (k1 \bullet a0 \bullet b0 \bullet c0) \bullet (\varepsilon)$
$w(11) = [10100101010000]^T$	$IE(11) = a0\_0$	$OE(11) = \varepsilon$	*No transition*
$w(12) = [10000101010000]^T$	$IE(12) = k1\_0$	$OE(12) = \varepsilon$	*No transition*
$w(13) = [00000101010000]^T$	$IE(13) = a1\_1$	$OE(13) = A+\_0 \bullet A-\_1 \bullet B\_1$	$\chi(t_2) = (=1) \bullet (a1\_1)$
$w(14) = [00010101001100]^T$	$IE(14) = b0\_0$	$OE(14) = \varepsilon$	*No transition*
$w(15) = [00010001001100]^T$	$IE(15) = a1\_0$	$OE(15) = \varepsilon$	*No transition*
$w(16) = [00000001001100]^T$	$IE(16) = b1\_1$	$OE(16) = B\_0$	$\chi(t_4) = (=1) \bullet (b1\_1)$
$w(17) = [00000011001000]^T$	$IE(17) = a0\_1$	$OE(17) = A-\_0$	$\chi(t_3) = (=1) \bullet (a0\_1)$
$w(18) = [00100011000000]^T$	$IE(18) = b1\_0$	$OE(18) = \varepsilon$	*No transition*
$w(19) = [00100001000000]^T$	$IE(19) = b0\_1$	$OE(19) = \varepsilon$	*No transition*
$w(20) = [00100101000000]^T$	$IE(20) = k2\_1$	$OE(20) = A+\_1$	$\chi(t_5) = (k2 \bullet a0 \bullet b0 \bullet c0) \bullet (\varepsilon)$
$w(21) = [01100101010000]^T$	$IE(21) = a0\_0$	$OE(21) = \varepsilon$	*No transition*
$w(22) = [01000101010000]^T$	$IE(22) = k2\_0$	$OE(22) = \varepsilon$	*No transition*
$w(23) = [00000101010000]^T$	$IE(23) = a1\_1$	$OE(23) = \varepsilon$	*No transition*
$w(24) = [00010101010000]^T$	$IE(24) = a1\_0$	$OE(24) = \varepsilon$	*No transition*
$w(25) = [00000101010000]^T$	$IE(25) = a2\_1$	$OE(25) = A+\_0 \bullet A-\_1 \bullet C\_1$	$\chi(t_6) = (=1) \bullet (a2\_1)$
$w(26) = [00001101001010]^T$	$IE(26) = a2\_0$	$OE(26) = \varepsilon$	*No transition*
$w(27) = [00000101001010]^T$	$IE(27) = c0\_0$	$OE(27) = \varepsilon$	*No transition*
$w(28) = [00000100000101]^T$	$IE(28) = k1\_1 \bullet c1\_1$	$OE(28) = C\_0$	$\chi(t_7) = (=1) \bullet (c1\_1)$
$w(29) = [10000100101000]^T$	$IE(29) = a1\_1$	$OE(29) = \varepsilon$	*No transition*
$w(30) = [10010100001000]^T$	$IE(30) = a1\_0 \bullet c0\_1$	$OE(30) = \varepsilon$	*No transition*
$w(31) = [10000101001000]^T$	$IE(31) = a0\_1$	$OE(31) = A-\_0$	$\chi(t_3) = (=1) \bullet (a0\_1)$
$w(32) = [10100101000000]^T$	From here, previously observed transitions are found		
…			

Table 4.5 Computed transitions for *Example 4.3*

Not all the sequence treatment is shown since it consists of 222 I/O vectors; at the end of the procedure, we get the following observable incidence matrix which represents the structure of Figure 4.5 .

	t_1	t_2	t_3	t_4	t_5	t_6	t_7
A+	1	-1	0	0	1	-1	0
A-	0	1	-1	0	0	1	0
B	0	1	0	-1	0	0	0
C	0	0	0	0	0	1	-1

By concatenation of all computed transitions in the fourth column, we obtain the following transition sequence:

S = t_1 t_2 t_3 t_4 t_1 t_2 t_4 t_3 t_5 t_6 t_7 t_4 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_4 t_1 t_2 t_3 t_4 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_4 t_5 t_6 t_7 t_4 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_4 t_5 t_6 t_7 t_4 t_5 t_6 t_7 t_4 t_5 t_6 t_7 t_4 t_1 t_2 t_4 t_3 t_1 t_2 t_3 t_4 t_1 t_2 t_3 t_4 t_1 t_2 t_4 t_3 t_1 t_2 t_3 t_4 t_1

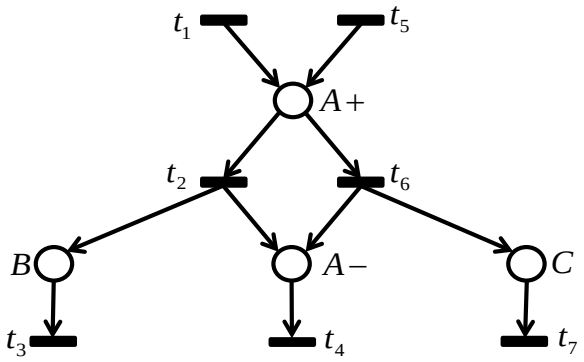


Figure 4.5 Obtained structure for *Example 4.3*

After this processing, we have only defined the observable part of the IPN, i.e., the reactive behaviour of the compound PLC + controller. It remains to infer the internal state evolutions of *Type 3.a*. Such a problem can be seen as finding how to preserve the firing of the computed transitions through non observable places to obtain a net reproducing *S*. To do this, we can work now over the simplified transition sequence obtained, considering the incidence matrix and firing functions we have computed.

4.3. Determining the non observable PN model

4.3.1. Problem re-statement

The previously described procedures allow obtaining an observable structure which represents the reactive behaviour of the system. Given that events and transitions of the net are completely defined, we need to add non-observable places to translate an aggregation of the non-observable dynamics of the process in such a way that the global PN will reproduce the whole behaviour of the system.

The problem of determining the non observable part of the *IPN* model complementary to that describing the observable (reactive) behaviour is stated as follows.

Given an observable *IPN* model whose structure is (P^{obs}, T, Pre^{obs}, $Post^{obs}$) and a transitions sequence $S = t_1\ t_2\ \ldots\ t_j\ \ldots\ \in T^*$ reproducing the I/O sequence *w*, a *PN*

structure (P^{nobs}, T, Pre^{nobs}, $Post^{nobs}$) that reproduces S and an initial marking M_0 enabling S must be found. The new PN structure is $N=(P, T, I, O)$ with $P= P^{obs} \cup P^{nobs}$, $Pre= Pre^{obs} \cup Pre^{nobs}$, $Post= Post^{obs} \cup Post^{nobs}$. The PN must be ordinary, free-choice and safe.

Observe that in S there are not consecutive apparitions of the same transition, due to the nature of the considered events (rising and falling edges of binary signals).

As reviewed in Chapters 1 and 2 of this thesis, in the literature there are many approaches which tackle the described identification problem, However, remember that the hypothesis made on such works are not satisfied by our scenario or their characteristics can be improved:

a) We only have a single sequence which could contain system cycles

b) We have no counterexamples available (we don't know the whole language of the system)

c) We look for polynomial time algorithms

d) We want to build an *IPN* model that allows showing structurally parallelism

New places and arcs must be determined such that they join PN fragments that have been built. We will connect them by relating transitions with non-observable places.

Since the tasks in different processes can occur simultaneously or at some predefined order, each two fragments can be related in two manners: sequentially or concurrently. Thus, several connecting forms are possible as illustrated in Figure 4.6, where "clouds" represent the fragments.

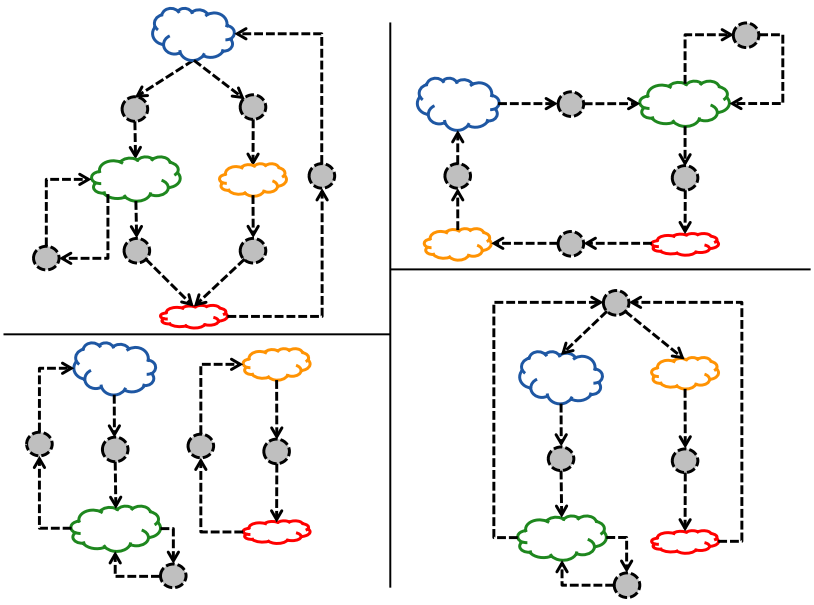


Figure 4.6 Some different possibilities for fragments assembling

In this section, we present a procedure to find precedence and concurrency relations among transitions, which will determine the final structure of the identified model. First some properties derived from the sequence S are introduced. Afterwards, based on such properties, an analysis technique allowing determining causal and concurrency relationships among the transitions in S are proposed. Then, the rules for building a net structure observing the causal and concurrency relationships are presented.

4.3.2. Dynamical properties

Since the construction method is based on the analysis of causal and concurrency relationships, some notions must be defined before introducing construction procedure of the non-observable behaviour.

Definition 4.3 The relationship between transitions in *S* that are observed consecutively is expressed in a relation $Seq \subseteq T \times T$ which is defined as $Seq = \{(t_j, t_{j+1}) | 1 \leq j < |S| \}$. If $(t_a, t_b) \in Seq$, this is denoted by $t_a < t_b$.

In a *PN* model every pair in *Seq* may in fact be represented differently. If t_a, t_b were observed consecutively in *S*, this behaviour could be issued from one of two situations in *N* described in the following definition.

Definition 4.4 Every couple of consecutive transitions t_a, t_b in *Seq* can be classified in one of the following situations:

Causal relationship. If the occurrence of t_a enables t_b. In a PN structure, this implies that there must be at least one place from t_a to t_b (Figure 4.7a).

Concurrent relationship. If both t_a and t_b are simultaneously enabled, but t_a occurs first and its firing does not disable t_b. In a PN structure, this implies that it is impossible the existence of a place from t_a to t_b. In this case, t_a and t_b are said to be concurrent, denoted as $t_a \| t_b$. (Figure 4.7b).

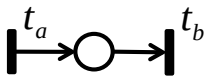


Figure 4.7a Causal relationship from t_a to t_b

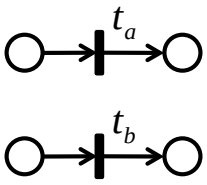


Figure 4.7b Concurrent relationship from t_a to t_b

Figure 4.7 Structures that represent $t_a < t_b$

The following notion is the *systematic precedence* of a transition t_j with respect to another transition t_k; it establishes a necessary condition for t_j to occur repeatedly.

Definition 4.5 A transition t_j is *preceded systematically* by t_k, denoted as $t_k \angle t_j$ iff t_k is always observed between two apparitions of t_j in *S*. By convention, we say that $t_j \angle t_j$ if t_j was observed at least twice in *S*. Then the *Systematic Precedence Set* of a transition t_j is given by the function *PS*: $T \rightarrow 2^T$, that indicates which transitions **must** be fired to re-enable the firing of t_j, i.e. $PS(t_j) = \{t_k | t_k \angle t_j\}$. If t_j was observed only once in *S*, then $PS(t_j) = \varnothing$.

Definition 4.6 Two transitions t_a, t_b are called transitions in a *two-cycle* if *S* contains the subsequence $t_a t_b t_a$ or the subsequence $t_b t_a t_b$. The two-cycle transitions set *TC* of S is given by $TC = \{(t_a, t_b) | t_a, t_b \text{ are in a two-cycle}\}$.

Example 4.5. In the sequence $S = t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_4\ t_3\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_4\ t_3\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_4\ t_3\ t_1\ t_2\ t_3\ t_4\ t_1$ from *Example 4.4*, one may observe that $t_2 \angle t_1$, $t_3 \angle t_1$, $t_4 \angle t_1$, thus $PS(t_1) = \{t_1, t_2, t_3, t_4\}$. Notice that $PS(t_j)$ is the set of transitions that must invariantly occur to fire t_j repeatedly. The rest of the PS sets are:

$PS(t_2) = \{t_1, t_2, t_3, t_4\}$, $PS(t_3) = \{t_1, t_2, t_3\}$,

$PS(t_4) = \{t_4\}$, $PS(t_5) = \{t_4, t_5, t_6, t_7\}$,

$PS(t_6)=\{t_4, t_5, t_6, t_7\}$, $PS(t_7)=\{t_4, t_5, t_6, t_7\}$.

The set of consecutively observed transitions is $Seq=\{ (t_1, t_2), (t_2, t_3), (t_3, t_4), (t_4, t_1), (t_2,t_4), (t_4, t_3), (t_3, t_5), (t_5, t_6), (t_6, t_7), (t_7, t_4), (t_4, t_5), (t_3, t_1)\}$. The set of transitions in a two-cycle is $TC= \varnothing$.

Definition 4.7 A Petri net *circuit* is a path starting and ending in the same node. A circuit is said to be *simple* if it does not use the same transition more than once, and *elementary* if it does not use the same place more than once.

We will now extract some structural properties regarding N from S. The previously defined notions will be used to determine which situation between causality and concurrence is the most appropriate for every pair of consecutively observed transitions in S.

4.3.3. Causal and concurrency relationships

4.3.3.1. Causal relationship

Proposition 4.2 If $t_a \angle t_b$ ($t_a \in PS(t_b)$) then, there must exist in N a simple elementary circuit (SE circuit) to which both t_a and t_b belong.

Proof. Suppose there is not a SE circuit containing t_a and t_b. Thus, right after the firing of t_b, all the tokens in $t_b^\bullet$ could be displaced by transition firings through some path to ${}^\bullet t_b$, enabling t_b without needing to fire t_a, which implies that $t_a \notin PS(t_b)$. ♦

Proposition 4.3. If $t_a < t_b$ and $t_a \angle t_b$, then there must exist in N a place from t_a to t_b.

Proof. Suppose that there is not a place from t_a to t_b. In order to allow the observation $t_a<t_b$, both t_a and t_b should be enabled simultaneously. By *Proposition 4.2*, there is at least one SE circuit containing t_a and t_b and thus, at least one path from t_a to t_b. Thus, if t_a and t_b are enabled simultaneously and t_a is fired, all paths from t_a to t_b contain two tokens. If all transitions in a path from t_a to t_b are fired, then there will be two tokens in one of the input places of t_b, resulting in a non-safe net. Then, at least one of the transitions t_i in each path from t_a to t_b must be conditioned to the previous firing of t_b. But if t_b is fired, all the transitions in paths from t_a to t_b can be fired and all the transitions in paths from t_b to t_b which do not include t_a can be fired; thus t_b will be enabled before t_a fires and as a consequence $t_a \notin PS(t_b)$. ♦

Proposition 4.4 If $t_a < t_b$ and $t_b \angle t_a$, then there must exist in N a place from t_a to t_b.

Proof. Suppose that there is not a place from t_a to t_b. Then, before the observation of $t_a < t_b$, both t_a and t_b must be enabled, and thus the occurrence of $t_b<t_a$ is possible, which together with $t_b \angle t_a$ and by *Proposition 4.3* implies that there should be a place from t_b to t_a. However, at the firing of t_b there are two tokens in such a place, and thus the net is not safe. ♦

Proposition 4.5 If $(t_a,t_b) \in TC$, then there must exist in N a place from t_a to t_b and a place from t_b to t_a.

Proof. The sequence $t_a t_b t_a$ must be reproducible in N. Right after the firing of t_a there is a token on its output places, and thus t_b must be at the output of such places; otherwise, there would be two tokens in such places after the second firing of t_a. Similarly, right after the first firing of t_a, there are no tokens on its input places, and thus t_b must be at the input of such places; otherwise, t_a could not be fired again. The same reasoning can be applied to reproduce the sequence $t_b t_a t_b$. ♦

Notice that when two transitions are observed consecutively and one is systematically preceded by the other, a causal relationship is found. Also, when two transitions are involved in a two-cycle relation, they are in a causal relationship each other. Observe that all of these relationships are structural, and thus they do not depend of the initial marking of the net.

Definition 4.8 The causal relationship set *CausalR* keeps track of all the causal relationships in *S*. $CausalR = \{(t_a, t_b) \mid (t_a < t_b)$ and $(t_a \angle t_b$ or $t_b \angle t_a$ or $(t_a,t_b) \in TC)\}$.

Example 4.6 From the *PS* sets and the Seq set in the ***Example 4.5***, we compute $CausalR=\{(t_1, t_2), (t_2, t_3), (t_4, t_1), (t_2, t_4), (t_5, t_6), (t_6, t_7), (t_7, t_4), (t_4, t_5), (t_3, t_1)\}$.

If a couple of transitions (t_a, t_b) in the *Seq* set, belongs also to *CausalR*, there must be a place from t_a to t_b in order to preserve the observed firing order. For the rest of the transition couples in *Seq*, we must decide if a place should exist to relate them. Next, we will discuss some cases where the existence of a place can be discarded.

4.3.3.2. Concurrency relationship

If two transitions t_a and t_b are concurrent, there must not exist a place neither from t_a to t_b nor from t_b to t_a; otherwise, the firing of one would constrain the firing of the other one.

Definition 4.9 The set of all pairs of concurrent transitions is called $ConcR=\{(t_a,t_b) \mid t_a \| t_b\}$.

If the sequence *w* is complete, (consequently, *S*) i.e., if it shows all of the possible behaviour of the observed system, we can find concurrency between transitions that are not in a causal relation, as showed in the next proposition.

Proposition 4.6. Let t_a, t_b be two transitions which have been observed consecutively in a complete sequence *S* in both orders, i.e. $(t_a, t_b) \in Seq$, $(t_b, t_a) \in Seq$. Then $(t_a, t_b) \notin CausalR$ and $(t_b, t_a) \notin CausalR$ if and only if $t_a \| t_b$.

Proof. **(→)** Suppose that $(t_a,t_b) \notin ConcR$. Without lose of generality, we suppose there is a place p_{ab} from t_a to t_b. Since $(t_b, t_a) \in Seq$, there must also be a place p_{ba} from t_b to t_a; otherwise, t_a could be enabled simultaneously with t_b to allow $t_b < t_a$ and t_a may be fired, yielding to the presence of two tokens in the place p_{ab} and breaking the safeness condition. Since $(t_a,t_b) \notin CausalR$, $t_b \notin PS(t_a)$ and thus there must be at least one path from p_{ab} to p_{ba} which does not contain t_b. Similarly, there must be at least one path from p_{ba} to p_{ab} which does not contain t_a. Since $(t_a,t_b) \notin TC$, $t_a t_b t_a$ should not be enabled and thus, there must be at least one SE circuit to which t_a belongs, but t_b does not belong. The resulting net violates the free-choice conditions (observe Figure 4.8).

(←) Suppose now that $(t_a,t_b) \in ConcR$. This means that they can be both enabled simultaneously and one can be fired without needing the firing of the other one, and thus $t_a \notin PS(t_b)$ and $t_b \notin PS(t_a)$. Also, since there cannot be any place from t_a to t_b nor from t_b to t_a, neither the subsequence $t_a t_b t_a$, nor the subsequence $t_b t_a t_b$ can be enabled, and thus $(t_a,t_b) \notin CausalR$ and $(t_b, t_a) \notin CausalR$. ♦

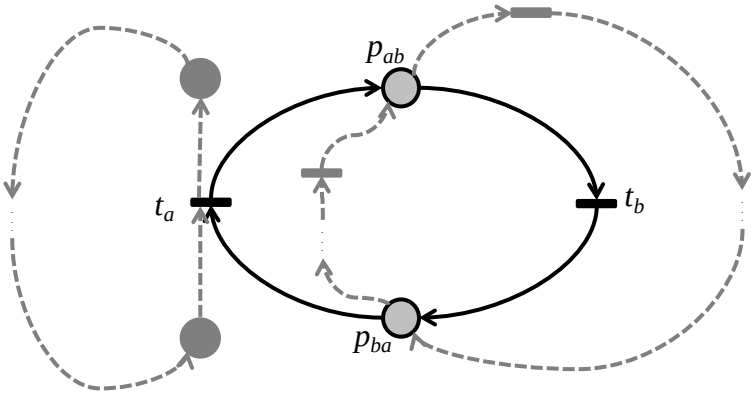


Figure 4.8 (t_a, t_b)∈*Seq*, (t_b, t_a)∈*Seq* and (t_a,t_b)∉*ConcR*

Notice that our methodology allows computing also non free-choice nets. Only in the case where the system includes behaviour like the one shown in Figure 4.8, the transitions t_a and t_b would be wrongly considered as concurrent and the existence of links from t_a to p_{ab} and from t_b to p_{ba} would be missed. However, the obtained model would be still capable to reproduce the sequence *S*.

It is well known that in practice, the sequence *w* is not complete, since in the general case, the observed systems do not show all their possible behaviour during the collection data. In fact, it is not possible to assure that the whole behaviour of a system has been observed. The consideration of *Proposition 4.6* is then very restrictive, since it demands the observation of all possible behaviour; it could lead to the construction of incorrect models in case of incomplete sequences. Then, some less constraining rules to find concurrency must be considered. Next, we present several properties which allow us to identify couples of transitions which must be concurrent in the identified net *N*.

First, we will introduce the notion of *Sequential Independence*, which is a characteristic of concurrent transitions. Later, the propositions to find concurrency will be introduced.

Definition 4.10 Two transitions t_a and t_b are *Sequentially Independent* if $t_a \notin PS(t_b)$ and $t_b \notin PS(t_a)$.

Proposition 4.7. Let t_a and t_b be two transitions in *S* which have been observed consecutively in both orders ($t_a < t_b$ and $t_b < t_a$). If:

a) $(t_a,t_b) \notin CausalR$ and $(t_b,t_a) \notin CausalR$

b) and $|PS(t_a)| > 1$ and $|PS(t_b)| > 1$

Then $t_a || t_b$.

Proof. Suppose that t_a and t_b are not concurrent. Without lose of generality, suppose there is a place p_{ab} from t_a to t_b. Since $t_b < t_a$ has been observed, there must be also a place p_{ba} from t_b to t_a (and as consequence *N* contains a two-transition cycle); otherwise, t_a could be enabled simultaneously with t_b to allow $t_b < t_a$ and t_a may be fired, yielding to the presence of two tokens in the place p_{ab} and breaking the safeness condition. Since $t_b \notin PS(t_a)$, there must be at least one path leading from p_{ab} to p_{ba} not including t_b. Since $|PS(t_a)| > 1$, there must be at least one circuit including t_a and not including p_{ab}, p_{ba} nor t_b. Since $t_a \notin PS(t_b)$, there must be at least one path leading from p_{ba} to p_{ab} not including t_a. Consider the first transition t_x of this path. The free-choice conditions are not

satisfied, since t_x and t_a share p_{ba} as input place, but t_a has at least one different input place.♦

Observe the net in Figure 4.9 which is composed by two independent t-components X_1 and X_2 with supports $<X_1>= \{t_a,t_i\}$ and $<X_2>= \{t_b,t_k\}$ respectively. In a sequence belonging to the language of such a net, transitions belonging to different t-components are sequentially independent. In fact, *PS* sets of this net correspond exactly to t-components of the net.

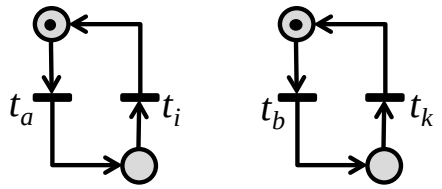


Figure 4.9 A net with two t-components

The *PS* set of a given transition $PS(t_j)$ is very useful to find concurrency when it is not a singleton. However, if $PS(t_j)$ is singleton, it means that it belongs to several elementary circuits and then *Proposition 4.7* does not allow anymore to find concurrent transitions to t_j. However if t_j is included in the *PS* of other transitions, we may find some concurrency relations, as shown in the next proposition.

Proposition 4.8. Let t_a and t_b two transitions in S that have been observed consecutively in both orders ($t_a < t_b$ and $t_b < t_a$). If t_a and t_b

a) are *Sequentially Independent* and

b) there exists a transition t_k such that $t_a \angle t_k$ ($t_a \in PS(t_k)$) and $t_b \angle t_k$ ($t_b \in PS(t_k)$)

then $t_a || t_b$.

Proof. Suppose that it does not hold that $t_a || t_b$. Without loss of generality, suppose that there is a place from t_a to t_b. Since $t_a \in PS(t_k)$ and $t_b \in PS(t_k)$, after the firing of t_k, both t_a and t_b must be fired before the next firing of t_k. Since $t_b < t_a$ may happen, the place from t_a to t_b must be marked. However $t_a < t_b$ may occur too, leading to the presence of two tokens in the same place after the firing of t_a, and making the net not safe.♦

Example 4.7. Figure 4.10 shows an example of the case characterised by *Proposition 4.8*. It is the general case of transitions belonging to concurrent threads (t_a, t_c and t_b, t_d, t_e, t_f respectively), which are eventually synchronized by one transition (t_k). If we make several firings to build a transition sequence, eventually the *PS* sets would be:

$PS(t_k) = \{t_k, t_a, t_c, t_b, t_d, t_f\}$

$PS(t_a) = PS(t_c) = \{t_k, t_a, t_c\}$

$PS(t_b) = PS(t_f) = \{t_k, t_b, t_d, t_f\}$

$PS(t_e) = \{t_e, t_d\}$, $PS(t_d) = \{t_d\}$

Even if $PS(t_d)$ is singleton, the synchronization point t_k help us to find by *Proposition 4.8* that $t_d \parallel t_a$ and that $t_d \parallel t_c$.

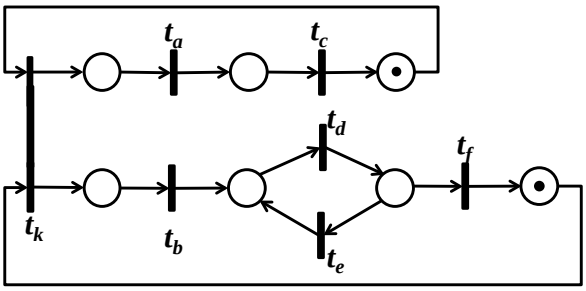


Figure 4.10 Concurrent threads synchronized by a transition

If concurrent transitions do not belong to synchronized threads, conditions of the next propositions help us to find a subset of concurrent transitions which do not depend from another transition t_k.

Proposition 4.9. Let be two transitions t_a and t_b which have been observed consecutively in both orders ($t_a < t_b$ and $t_b < t_a$). If t_a and t_b are:

a) *Sequentially Independent* and

b) $\exists\ t_k$ such that $t_k \in PS(t_b)$, $t_k \notin PS(t_a)$, and

c) $(t_a, t_k) \in Seq$

then $t_a \| t_b$.

Proof. Suppose there is a place p_{ab} from t_a to t_b. Since $t_b < t_a$ has also been observed, there must be also a place p_{ba} from t_b to t_a; otherwise, t_a should be enabled simultaneously with t_b to allow $t_b < t_a$ and t_a may be fired, yielding to the presence of two tokens in p_{ab}. Since there exist t_k such that $t_k \in PS(t_b)$, then there must be a SE circuit containing both t_b and t_k. If such a circuit contains places p_{ba} or p_{ba}, it is not possible to fire $t_a < t_k$ and thus such a circuit must contain another input place p_{kb} of t_b and another output place p_{bk} of t_b. Now, to accomplish that $t_b \notin PS(t_a)$, there must be at least one path leading from p_{ab} to some input place of t_a not including t_b. Consider the first transition t_x of this path. In order to respect the free-choice conditions, p_{kb} should be an input place of t_x, making the occurrence of $t_a < t_k$ impossible.♦

Definition 4.11 The *Inverse Systematic Precedence* set of a transition PS^{-1}: $T \rightarrow 2^T$ contains the transitions which are dependent of a common transition to re-enable their firing:

$$PS^{-1}(t_j) = \{t_k \mid t_k \neq t_j \text{ and } t_j \in PS(t_k)\}$$

Proposition 4.10 Let be t_a and t_b two transitions which have been observed consecutively in both orders ($t_a < t_b$ and $t_b < t_a$). If t_a and t_b are:

a) *Sequentially Independent*, and

b) $PS^{-1}(t_a) \neq \varnothing$, $\forall t_j \in PS^{-1}(t_a)$, $t_j \| t_b$,

then $t_a \| t_b$.

Proof. Suppose there is a place p_{ab} from t_a to t_b. Since $t_b < t_a$ has also been observed, there must be also a place p_{ba} from t_b to t_a; otherwise, t_a should be enabled simultaneously with t_b to allow $t_b < t_a$ and thus t_a may be fired, yielding to the presence of two tokens in the place from t_a to t_b. Since $t_b \notin PS(t_a)$, there must be at least one path leading from p_{ab} to p_{ba} not including t_b. Similarly, there must be at least one path leading from p_{ba} to p_{ab} not including t_a. Since $PS^{-1}(t_a) \neq \varnothing$, there is at least one transition t_j concurrent to t_b such that $t_j \angle t_a$ and there must be a SE circuit including t_a and t_j. Such a

circuit cannot contain p_{ab} nor p_{ba} otherwise t_j may be able to fire without need of firing t_a. Consider the input place p_x of t_a in this path. The free-choice conditions are not satisfied between p_x and p_{ba}: they share t_a as output transition, but p_{ba} has at least another output transition.♦

An example where *Proposition 4.10* can be used is shown in Figure 4.11. $PS^{-1}(t_a) = \{t_{j1}, t_{j2}\}$ and $t_{j1} \| t_b$, $t_{j2} \| t_b$ are determined by *Proposition 4.7*. Consequently, $t_a \| t_b$.

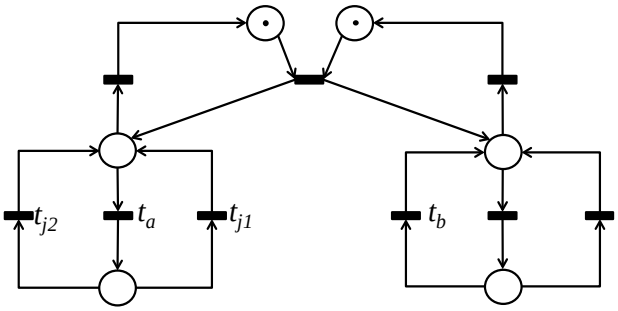


Figure 4.11 Concurrence between transitions whose *PS* is a singleton

4.3.4. Building the non-observable PN

We will use now the computed data from sequence *S* to infer internal evolutions of the system. We will make an analysis of causal and concurrency relations that have been found between consecutive transitions in order to compute non-observable places of the net.

Definition 4.12 The set *Seq*'= (*Seq* - *CausalR*) - *ConcR* contains the set of transition pairs (t_a,t_b) which have been observed consecutively, but are not in a causal relation or in a concurrency relation.

If *Seq*' ≠ ∅, there are two possibilities for the remaining transition pairs (t_a,t_b) in *Seq*':

a) They are both input and output transitions of a place with several input and output transitions
b) They are concurrent, but *w* (thus, *S*) is not complete enough to find such a relationship

Since our goal is to approximate as much as possible the language generated by the net *N* to the observed sequence *S*, we assume that if we have observed two transitions consecutively (t_a<t_b) but by none of the previous propositions we have determined that they are concurrent, thus the firing of t_a has enabled t_b. This is made in order to preserve in *N* the firing order observed in *S*. Then, a place will be added from t_a to t_b; this denoted by [t_a, t_b].

When it is found that [t_a, t_c] and [t_b, t_c], and the involved transitions are related by a single place, this is represented as [$t_a t_b$, t_c]. In general, a place *p* can be denoted as [t_{a1} $t_{a2...}$ t_{al}, t_{b1} $t_{b2...}$ t_{bh}], where t_{ai} are the input transitions of *p* and t_{bi} are the output transitions of *p*, and $l=|{}^\bullet p|$, $h=|p^\bullet|$, as illustrated in Figure 4.12.

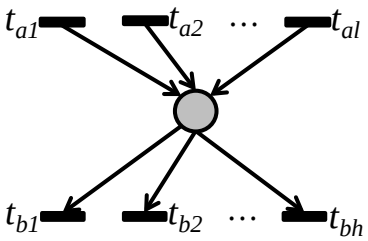
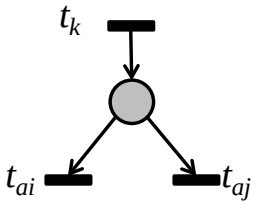
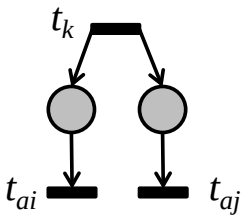


Figure 4.12 A PN place $p = [t_{a1}\ t_{a2...}t_{al}, t_{b1}\ t_{b2...}\ t_{bh}]$

The same place could be used to relate several consecutive transitions. If a transition t_k has been observed followed by two transitions t_{ai}, t_{aj} in S ($t_k<t_{ai}$ and $t_k<t_{aj}$), there are two cases to represent such observations into the PN model: the case of selection, where they are represented with the same place [t_k, t_{ai} t_{aj}] (Figure 4.13a) or the case of concurrence, where they are represented with different places [t_k,t_{ai}] [t_k, t_{aj}] (Figure 4.13b).



a) t_{ai}, t_{aj} are not concurrent and have not been observed consecutively



b) t_{ai}, t_{aj} are concurrent or have been observed consecutively

Figure 4.13 Selection and parallelism representation

In a generalized form, for every set $t_k<t_{a1},\ldots, t_k<t_{aw}$ of non-concurrent consecutive transition pairs with the same first transition t_k, we can thus merge all $t_k<t_{a1},\ldots, t_k<t_{ax}$ whose second transitions $t_{a1}\ldots t_{aw}$ are non-concurrent nor consecutive and represent them into a single place [t_k, $t_{a1}\ldots t_{aw}$], as illustrated in Figure 4.14.

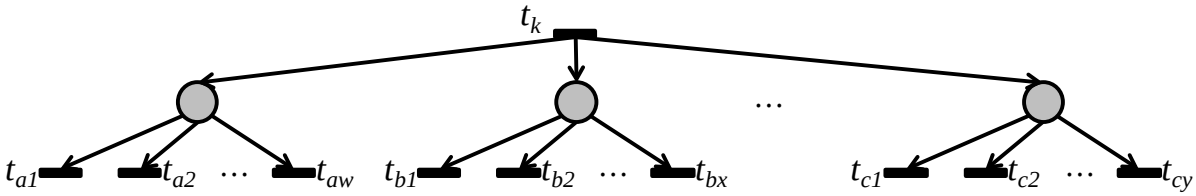


Figure 4.14 Selection and concurrence between post-transitions

Once we have made the first merging, all places [t_{k1}, $t_{a1}\ldots t_{aw}$], [t_{k2}, $t_{a1}\ldots t_{aw}$],…, [t_{kz}, $t_{a1}\ldots t_{aw}$] whose input transitions are non-concurrent nor consecutive and whose output transitions are the same, can be merged into a single place as illustrated in Figure 4.15.

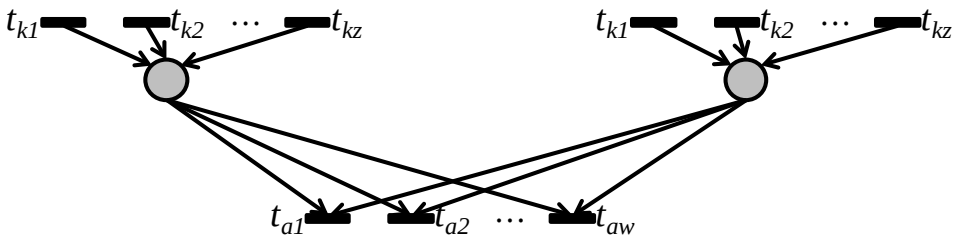


Figure 4.15 Selection and concurrence between pre-transitions

Once the structure of the net is built, the initial marking can be computed allowing the firing of *S*. All transitions are processed, from the last transition till the first one. The processing of a transition is as follows:

- If its output places are marked, the tokens in such places are retired
- Tokens are added to its unmarked input places

When there are transitions with an empty *PS*, it means that they were observed only once and thus, they will not be included in any cycle of the constructed model. In case of one of these transitions be a source transition, we will add an input marked place for avoiding non-safeness; accordingly for a sink transition we will add an unmarked output place.

Example 4.8. In the sequence from the *Example 4.5*, $S = t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_4\ t_3\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_4\ t_3\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_4\ t_3\ t_1\ t_2\ t_3\ t_4\ t_1$, the concurrent transitions determined are $ConcR = \{(t_3,t_4)(t_4,t_3)\}$. Since $Seq=\{(t_1, t_2), (t_2, t_3), (t_3, t_4), (t_4, t_1), (t_2,t_4), (t_4, t_3), (t_3, t_5), (t_5, t_6), (t_6, t_7), (t_7, t_4), (t_4, t_5), (t_3, t_1)\}$ and $CausalR=\{(t_1, t_2), (t_2, t_3), (t_4, t_1), (t_2, t_4), (t_5, t_6), (t_6, t_7), (t_7, t_4), (t_4, t_5), (t_3, t_1)\}$, we have that $Seq' = \{(t_3, t_5)\}$ and thus, there is a relationship that has not been explained as sequential nor as concurrent. We consider all couples of consecutive non-concurrent transitions (all couples in *Seq* - *ConcR*) to compute the places: $[t_1, t_2]$, $[t_2, t_3]$, $[t_3, t_1\ t_5]$, $[t_4, t_1\ t_5]$, $[t_5, t_6]$, $[t_6, t_7]$, and $[t_2\ t_7, t_4]$. The structure and the computed initial marking correspond to the net in Figure 4.16.

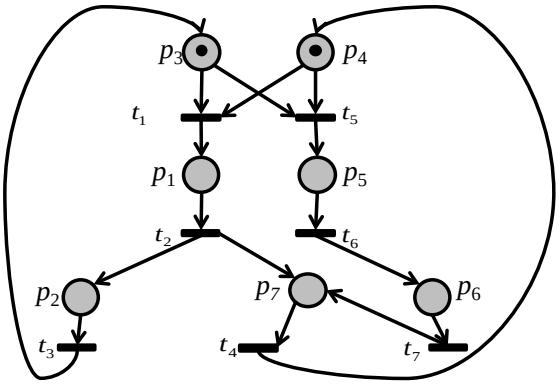


Figure 4.16 Non-observable model

4.3.5. Places verification

As stated before, with the proposed mechanisms in last section, the sequence *w* may not have shown enough combinations which allow us to determine concurrency. If the sequence *w* were complete, all the concurrent and sequential behaviour could be found and represented, according to *Proposition 4.6*. However, since we know that *w* might not be complete, in order to approximate the language of *N* to *S* as much as possible, we have considered that if two transitions have not been declared as concurrent, they must be in a sequential relationship. But if the transitions are actually concurrent, the sequential consideration could lead us to arcs or places in the built model which restrict too much the behaviour of the system and do not allow the firing of *S*. Now, we present some notions that will help us to verify if added places until now do not interfere in the correct reproduction of *S*.

Proposition 4.11. If the IPN model has been correctly build, every computed non-observable place *p* in *N* must fulfil the place input-output flow equation:

$$\sum_{t_i \in {}^\bullet p} Occ(t_i) = \sum_{t_i \in p^\bullet} Occ(t_i) \pm 1$$

where $Occ(t_k)$ is the number of occurrences of t_k in S

Proof. Equation follows straightforward from the IPN transition enabling and firing conditions and from the fact that N must be safe. ♦

Proposition 4.12 If there exists a place p such that $|{}^\bullet p|=1$, then $\forall t_j \in p^\bullet$, $t_k \in PS(t_j)$, where t_k is the input transition of p. Also, if there exists a place p such that $|p^\bullet|=1$, then $\forall t_j \in {}^\bullet p$, $t_k \in PS(t_j)$, where t_k is the output transition of p.

Proof. If $|{}^\bullet p|=1$, for the re-enabling of t_j, p must be marked and the only way to do so is the firing of t_k, and thus $t_k \in PS(t_j)$. Similarly, if $|p^\bullet|=1$, for the re-enabling of t_j, p must be unmarked and the only way to do so is the firing of t_k, thus $t_k \in PS(t_j)$.♦

Correction rule. If the input-output flow equation or the conditions in *Proposition 4.12* are not satisfied by some place, the arcs relating transitions which are not in *CausalR* are removed. If there are not *CausalR* represented in such a place, it is deleted.

Example 4.9 In the model of Figure 4.16, we check the input-output flow equation for each place. $Occ(t_1)$ = 12, $Occ(t_2)$ = 11, $Occ(t_3)$ = 11, $Occ(t_4)$ = 20, $Occ(t_5)$ = 9, $Occ(t_6)$ = 9, $Occ(t_7)$ = 9. We check also the condition of *Proposition 4.12*.

p_1: $Occ(t_1) = Occ(t_2)$ (±1), $t_1 \in PS(t_2)$, $t_2 \in PS(t_1)$

p_2: $Occ(t_2) = Occ(t_3)$ (±1), $t_2 \in PS(t_3)$, $t_3 \in PS(t_2)$

p_3: $Occ(t_3) \neq Occ(t_1) + Occ(t_5)$ (±1) → wrong place

p_4: $Occ(t_4) = Occ(t_1) + Occ(t_5)$ (±1), $t_4 \in PS(t_1)$, $t_4 \in PS(t_5)$

p_5: $Occ(t_5) = Occ(t_6)$ (±1), $t_5 \in PS(t_6)$, $t_6 \in PS(t_5)$

p_6: $Occ(t_6) = Occ(t_7)$ (±1), $t_6 \in PS(t_7)$, $t_7 \in PS(t_6)$

p_7: $Occ(t_2) + Occ(t_7) = Occ(t_4)$ (±1), $t_4 \in PS(t_2)$, $t_4 \in PS(t_7)$

Observe that the condition p_3: $Occ(t_3) = Occ(t_1) + Occ(t_5)$ (±1) is not satisfied, and thus the place p_3 must be corrected to allow reproducing the observed behaviour. Since $t_3 \in PS(t_1)$, $t_1 \in PS(t_3)$, $t_3 \notin PS(t_5)$ $t_5 \notin PS(t_3)$, we can conclude that (t_3, t_5) are not in a casual relationship (remember that *Seq*' = {(t_3, t_5)}), and thus, to correct the place, we delete the link going from p_3 to t_5. The resulting net is shown at Figure 4.17. Notice that the computed net is not free-choice.

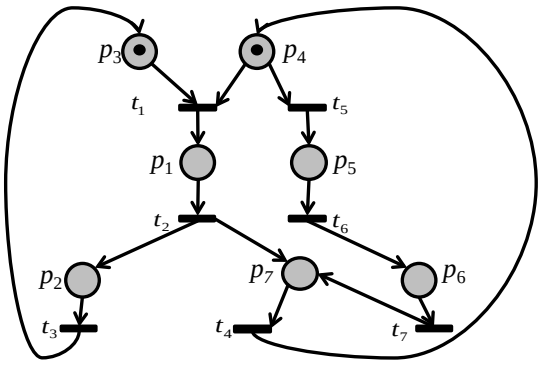


Figure 4.17 Non-observable model

Observe that in Figure 4.5 observable places [t_2, t_3] and [t_6, t_7] already exist. By adding computed non-observable places to such a model and deleting implicit places [t_2,t_3] and [t_6,t_7], we obtain the final result shown in Figure 4.18, which reproduces *w*.

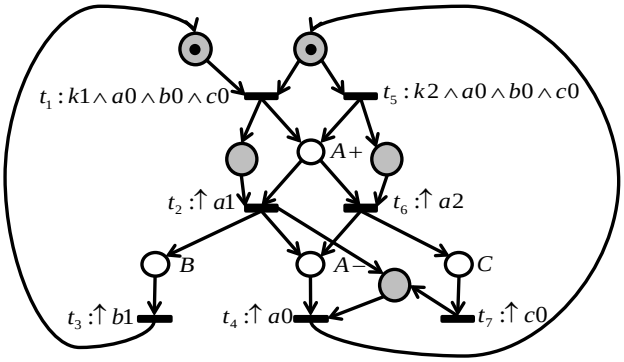


Figure 4.18 Final model representing the system from *Example 4.1*

Now, we present another example with the whole non-observable behaviour identification procedure developed.

Example 4.10. In order to illustrate our non-observable behaviour discovering technique, consider the net in Figure 4.19 composed by only non-observable places:

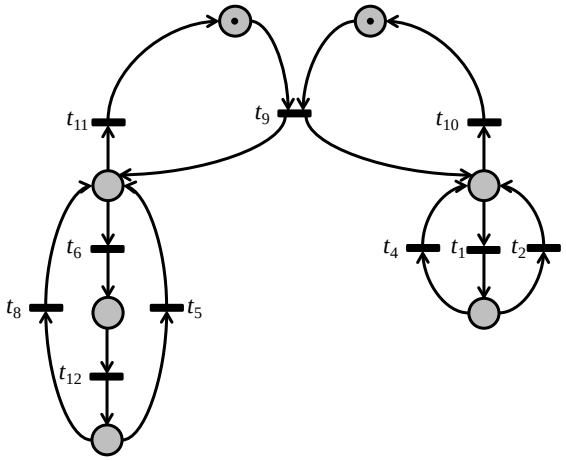


Figure 4.19 A test IPN with non-observable places

We have built the net in the Platform Independent Petri net Editor (PIPE), which is an editor for visualization and analysis of Petri nets. With such tool, we have generated a transition sequence of length 1000 by firing randomly transitions of the net : *S* = t9 t11 t1 t4 t10 t9 t6 t1 t2 t12 t5 t6 t12 t8 t11 t10 t9 t6 t10 t12 t8 t11 t9 t10 t11 t9 t10 t11 t9 t11 t10 t9 t6 t12 t10 t5 t11 t9 t1 t4 t1 t6 t2 t12 t5 t1 t2 t10 t6 t12 t8 t11 t9 t10 t11 t9 t10 t11 t9 t10 t11 t9 t1 t4 t11 t10 t9 t11 t10 t9 t11 t1 t4 t10 t9 t1 t11 t2 t1 t2 t1 t2 t1 t2 t10 t9 t6 t1 t12 t2 t10 t8 t11 t9 t11 t10 t9 t10 t6 t12 t8 t11 t9 t6 t1 t2 t10 t12 t5 t11 t9 t11 t10 t9 t6 t10 t12 t5 t11 t9 t6 t1 t2 t1 t2 t12 t1 t5 t6 t2 t1 t4 t10 t12 t8 t11 t9 t1 t4 t10 t6 t12 t5 t11 t9 t10 t11 t9 t10 t6...

From such a sequence, the following data has been computed:

PS(t11) = {t11, t9}, *PS*(t10) = {t10, t9}, *PS*(t9) = {t9, t11, t10}, *PS*(t8) = {t8, t6, t12}, *PS*(t6) = {t6, t12}, *PS*(t5) = {t5, t6, t12}, *PS*(t4) = {t4, t1}, *PS*(t2) = {t2, t1}, *PS*(t1) = {t1}, *PS*(t12) = {t12, t6}.

Occ(t11)=127, *Occ*(t10)=128, *Occ*(t9)=128, *Occ*(t8)=60, *Occ*(t6)=123, *Occ*(t5)=63, *Occ*(t4)=55, *Occ*(t2)=69, *Occ*(t1)=124, *Occ*(t12)=123

Observed consecutive transitions: *Seq* = {

(t9, t11),	(t11, t1),	(t1, t4),	(t4, t10),	(t10, t9),	(t9, t6),
(t6, t1),	(t1, t2),	(t2, t12),	(t12, t5),	(t5, t6),	(t6, t12),
(t12, t8),	(t8, t11),	(t11, t10),	(t6, t10),	(t10, t12),	(t11, t9),
(t9, t10),	(t10, t11),	(t12, t10),	(t10, t5),	(t5, t11),	(t9, t1),
(t4, t1),	(t1, t6),	(t6, t2),	(t5, t1),	(t2, t10),	(t10, t6),
(t4, t11),	(t1, t11),	(t11, t2),	(t2, t1),	(t1, t12),	(t12, t2),
(t10, t8),	(t12, t1),	(t1, t5),	(t6, t4),	(t8, t6),	(t2, t6),
(t12, t4),	(t11, t4),	(t1, t8),	(t8, t2),	(t2, t8),	(t4, t12),
(t2, t11),	(t8, t4),	(t4, t6),	(t8, t10),	(t2, t5),	(t5, t10),
(t8, t1)}					

Concurrences: *ConcR* = {

(t2, t12), (t11, t10), (t6, t10), (t10, t12), (t10, t5), (t6, t2), (t4, t11), (t11, t2), (t10, t8), (t6, t4), (t12, t4), (t8, t2), (t1, t11), (t1, t6), (t1, t12) }

After the creation of the net as specified in section 4.3.4, the following places have been considered: p11:[t11 ,t9], p10:[t10 ,t9], p9:[t8 t5 ,t1], p8:[t1 ,t8 t5], p7:[t12 ,t8 t5], p6:[t8, t4], p5:[t6 ,t12], p4:[t9 t8 t5 ,t6 t11], p3:[t2 ,t5], p2:[t9 t4 t2 ,t10 t1] and p1:[t1 , t4 t2].

Observe that t2<t5 has been observed, but t5<t2 has not been observed and this leads to the creation of the place p3:[t2, t5]. However, we make the previously described verification on this and other places and some corrections are made:

Place p11:[t11 ,t9] Correct place

Place p10:[t10 ,t9] Correct place

Place p9:[t8 t5 ,t1] Wrong place: $t1 \notin PS(t8)$, $t1 \notin PS(t5)$

Place p8:[t1 ,t8 t5] Wrong place: $t1 \notin PS(t8)$, $t1 \notin PS(t5)$

Place p7:[t12 ,t8 t5] Correct place

Place p6:[t8 ,t4] Wrong place: $Occ(t8) \neq Occ(t4) \pm 1$

Place p5:[t6 ,t12] Correct place

Place p4:[t9 t8 t5 ,t6 t11] Correct place

Place p3:[t2 ,t5] Wrong place: $Occ(t2) \neq Occ(t5) \pm 1$

Place p2:[t9 t4 t2 ,t10 t1] Correct place

Place p1:[t1 ,t4 t2] Correct place

All places not satisfying the flow equation are deleted. The final set of places is p11:[t11,t9], p10:[t10,t9], p7:[t12,t8 t5], p5:[t6,t12], p4:[t9 t8 t5,t6 t11], p2:[t9 t4 t2 ,t10 t1] and p1:[t1,t4 t2], which corresponds in fact to the net in Figure 4.19.

All algorithms described to construct the non-observable part of the net can be summarized in the following procedure.

***Algorithm 4.5* Non-observable behaviour construction**

Input: The sequence *S*
Output: Non-observable model representing *S*

1. Compute *Seq*, *PS* and *TC* from *S*
2. From the information in *Seq*, *PS* and *TC* compute *CausalR*
3. From *Seq* and *CausalR*, compute *ConcR*
4. Merge transitions as specified in 4.3.4
5. Validate and correct places as specified in 4.3.5

Proposition 4.13 The *PN* model *N* built with the previous procedures summarized in *Algorithm 4.5* executes the sequence *S*

Proof. Regard that we have computed the following sets:

- *Seq* containing all the consecutive transition couples in *S*. If we represent into a net all couples in *Seq*, the net will be able to reproduce *S*.
- *CausalR* containing transition couples $(t_a, t_b) \in Seq$ that must be related by a place.
- *ConcR* containing transition couples $(t_a, t_b) \in Seq$, that must not be related by any place.

If the set $Seq' = (Seq - CausalR) - ConcR = \varnothing$, that means that all transition couples $(t_a,t_b) \in Seq$ are correctly represented in *N* and thus the sequence *S* is reproducible. This follows from the consideration of ordinary safe Petri nets. If $Seq' \neq \varnothing$, it means that there are still transition couples that cannot be distinguished as concurrent or sequential. Thus, by merging several couples in *Seq*, all couples in *Seq*' are considered as sequential by creating places with several input and output transitions. If they are actually sequential, all the verification rules are satisfied. Otherwise, they are actually concurrent and they are corrected with the described procedure. Once they are corrected, it only remains places relating sequential transitions and thus the sequence *S* is executable. ♦

Properties of *Algorithm 4.5*

- Given that all of the procedures of *Algorithm 4.5* are executed in polynomial time the construction of the non-observable IPN is efficiently performed.
- The application of *Algorithm 4.5* to a sequence S provides always the same PN model, due to that all the constructive steps in the procedures are deterministically performed, i.e. there are not random selections on the input and intermediate data.

4.3.6. Test examples

The method for building the non-observable IPN has been tested with diverse *PN* structures following the same procedure detailed in the *Example 4.10*. We present here a subset of some representative examples which have been correctly identified.

For the sequence $S = t_3\ t_5\ t_4\ t_8\ t_1\ t_2\ t_7\ t_3\ t_2\ t_1\ t_4\ t_8\ t_7\ t_6\ t_5\ t_8\ t_5\ t_3\ t_4\ t_8\ t_5\ t_3\ t_8\ t_2\ t_3\ t_2\ t_1\ t_5\ t_8\ t_5\ t_6\ t_4\ t_1\ t_5\ t_6\ t_5\ t_6\ t_7\ t_4\ t_8\ t_7\ t_8\ t_1\ t_5\ t_6\ t_7\ t_4\ t_8\ t_3\ t_7\ t_4\ t_3\ t_8\ t_4\ t_5\ t_8\ t_5\ t_1\ t_8\ t_7\ t_2\ t_6\ t_1\ t_4\ t_3\ t_5\ t_8\ t_5\ t_8\ t_2\ t_7\ t_1\ t_6\ t_2\ t_1\ t_7\ t_8\ t_4\ t_7\ t_1\ t_2\ t_3\ t_6\ t_2\ t_1\ t_2\ t_5\ t_3\ t_8\ t_7\ t_4\ t_6\ t_3\ t_7\ t_8\ t_5\ t_2\ t_1\ t_2\ t_6\ t_1\ t_7\ t_6\ t_7\ t_8\ t_5\ t_4\ t_6\ t_7\ t_3\ t_2\ t_6\ t_3\ t_7\ t_4\ t_8\ t_5\ t_3\ t_4\ t_8\ t_5\ t_1\ t_6\ t_5\ t_6\ t_7\ t_6\ t_5\ t_8\ t_5\ t_6\ t_5\ t_8\ t_5\ t_8\ t_2\ t_3\ t_2\ t_1\ t_7\ t_4\ t_8\ t_1\ t_4\ t_5\ t_3\ t_6\ t_5\ t_2\ t_6\ t_1\ t_4\ t_1\ t_4\ t_1\ t_7\ t_2\ t_6\ t_5\ t_3\ t_6\ t_7\ t_2\ t_1\ t_6\ t_2\ t_5\ t_8\ t_7\ t_6\ t_5\ t_6\ t_1\ t_5\ t_2\ t_3\ t_2\ t_3\ t_8\ t_2\ t_1\ t_4\ t_3\ t_5\ t_8\ t_7\ t_4\ t_1\ t_8\ t_7\ t_2\ t_1\ t_2\ t_3\ t_6\ t_4\ t_7\ t_8\ t_7\ t_6$, the obtained *PN* model is shown in Figure 4.20, which is in fact two independent *PN* systems.

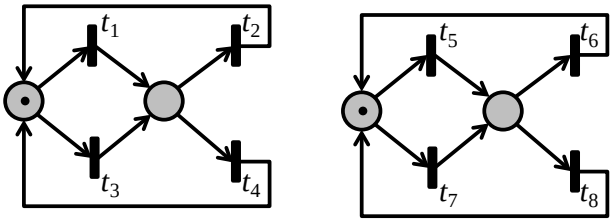


Figure 4.20 Two equal components running concurrently

The identification method applied to the sequence S = t_4 t_1 t_2 t_3 t_1 t_5 t_6 t_7 t_2 t_3 t_4 t_5 t_6 t_1 t_2 t_3 t_7 t_1 t_4 t_2 t_5 t_4 t_3 t_5 t_1 t_4 t_5 t_2 t_6 t_3 t_1 t_7 t_6 t_7 t_6 t_2 t_3 t_1 t_2 t_3 t_7 t_1 t_4 t_5 t_2 t_4 t_5 t_3 t_4 t_5 t_4 t_5 t_1 t_6 t_7 t_4 t_2 t_3 t_1 t_5 t_6 t_7 t_6 t_2 t_3 t_7 t_4 t_1 t_5 t_2 t_4 t_5 t_6 t_7 t_3 t_1 t_6 t_2 t_7 t_6 t_7 t_6 t_7 t_6 t_3 t_7 t_4 t_1 t_2 t_3 t_5 t_1 t_4 t_5 t_6 t_7 t_2 t_6 t_3 t_1 t_7 t_4 t_2 t_3 t_5 t_6 t_1 t_2 t_3 t_1 t_7 t_4 t_5 t_6 t_7 t_4 t_2 t_3 t_1 t_5 t_4 t_2 t_5 t_6 t_3 t_7 t_6 t_1 t_7 t_4 t_2 t_3 t_5 t_4 t_5 t_1 t_6 t_2 t_7 t_4 t_5 t_6 t_3 t_7 t_1 t_6 t_7 t_2 t_3 t_1 t_2 t_6 t_7 t_4 t_5 t_4 t_5 t_3 t_4 t_5 t_4 t_1 t_5 t_4 t_5 t_4 t_5 t_2 t_6 t_3 t_1 t_2 t_3 t_1 t_7 t_2 t_3 t_1 t_4 t_5 t_6 t_2 t_3 t_1 t_2 t_3 t_1 t_2 t_3 t_1 t_7 t_2 t_6 t_7 t_3 t_1 t_2 t_4 t_5 t_4 yields two different *PN* models shown in Figure 4.21.

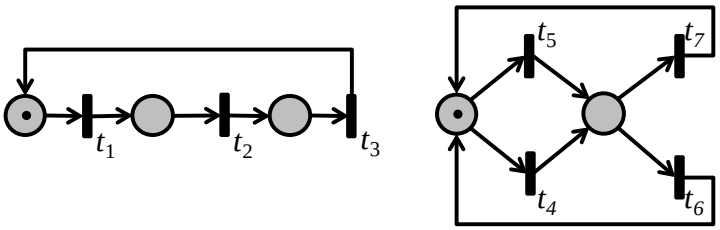


Figure 4.21 Two different components running concurrently

From the sequence S = t_1 t_2 t_3 t_1 t_4 t_5 t_6 t_7 t_8 t_2 t_3 t_1 t_2 t_4 t_5 t_6 t_3 t_4 t_5 t_1 t_7 t_2 t_8 t_6 t_3 t_4 t_1 t_2 t_5 t_7 t_8 t_6 t_7 t_8 t_3 t_1 t_2 t_4 t_5 t_6 t_3 t_1 t_2 t_7 t_4 t_8 t_5 t_6 t_7 t_3 t_1 t_2 t_4 t_5 t_8 t_6 t_7 t_8 t_3 t_4 t_5 t_1 t_2 t_6 t_7 t_3 t_8 t_1 t_2 t_4 t_5 t_6 t_3 t_7 t_8 t_1 t_2 t_4 t_5 t_6 t_7 t_3 t_8 t_4 t_5 t_6 t_1 t_7 t_2 t_3 t_4 t_8 t_1 t_5 t_2 t_6 t_7 t_3 t_1 t_4, the obtained model is shown in Figure 4.22, which is a composition of smaller sequential models exhibiting together concurrent behaviour.

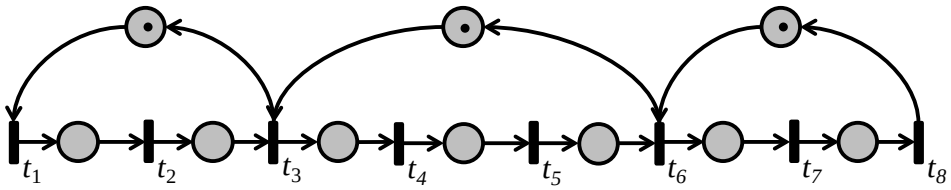


Figure 4.22 Concurrent machines net

The processing of the sequence S = t_5 t_3 t_1 t_0 t_1 t_0 t_2 t_4 t_5 t_4 t_5 t_4 t_0 t_1 t_5 t_4 t_5 t_3 t_2 t_4 t_0 t_2 t_3 t_1 t_0 t_2 t_4 t_0 t_2 t_4 t_5 t_4 t_5 t_3 t_1 t_0 t_1 t_5 t_3 t_2 t_3 t_2 t_4 t_0 t_2 t_4 t_0 t_1 t_5 t_4 t_5 t_4 t_0 t_2 t_3 t_2 t_3 t_2 t_4 t_0 t_1 t_0 t_2 t_4 t_0 t_1 t_0 t_1 t_5 t_3 t_2 t_3 t_2 t_3 t_1 t_0 t_1 t_5 t_4 t_5 yields a *PN* state machine (Figure 4.23) in which it is possible to move from one place to another in one step.

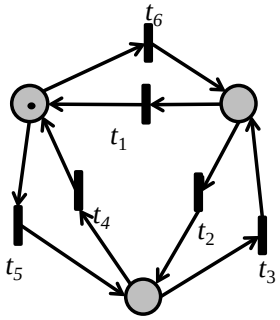


Figure 4.23 *PN* state machine whose PS(t_j)=∅ ∀t_j

The sequence $S = t_1\ t_4\ t_2\ t_1\ t_2\ t_4\ t_1\ t_4\ t_2\ t_1\ t_2\ t_3\ t_1\ t_3\ t_2\ t_1\ t_3\ t_2\ t_1\ t_3\ t_2\ t_1\ t_4\ t_2\ t_1\ t_2\ t_3\ t_1\ t_2\ t_4\ t_1\ t_3\ t_2\ t_1\ t_2\ t_4\ t_1\ t_3\ t_2\ t_1\ t_4\ t_2\ t_1\ t_2\ t_4\ t_1\ t_2\ t_3\ t_1\ t_3\ t_2\ t_1\ t_4\ t_2\ t_1\ t_3\ t_2\ t_1\ t_4\ t_2\ t_1\ t_3\ t_2\ t_1\ t_2\ t_4\ t_1\ t_4\ t_2\ t_1\ t_2\ t_3\ t_1\ t_4\ t_2\ t_1\ t_2\ t_4\ t_1\ t_4\ t_2\ t_1\ t_4\ t_2\ t_1\ t_2\ t_3\ t_1\ t_3\ t_2\ t_1\ t_3\ t_2\ t_1\ t_4\ t_2\ t_1\ t_3\ t_2\ t_1$ yield the model in Figure 4.24 when it is treated with the identification method.

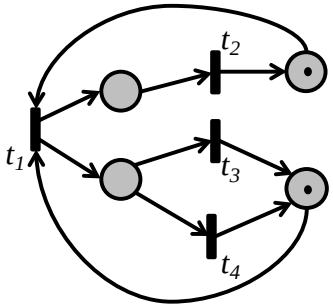


Figure 4.24 Selection and concurrency combined

The model in Figure 4.25 is obtained after the treatment of the sequence $S = t_0\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_1\ t_2\ t_4\ t_5\ t_3\ t_6\ t_1\ t_2\ t_4\ t_3\ t_5\ t_6\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_1\ t_2\ t_4\ t_5\ t_3\ t_6\ t_1\ t_2\ t_4\ t_5\ t_3\ t_6\ t_1\ t_2\ t_4\ t_3\ t_5\ t_6\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_1\ t_2\ t_4\ t_5\ t_3\ t_6\ t_1\ t_2\ t_4\ t_3\ t_5\ t_6\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_1\ t_2\ t_4\ t_7$. Notice that there are transitions which only fire once in the sequence (t_0 and t_7).

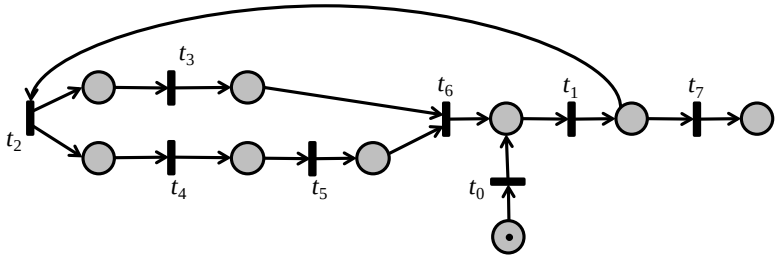


Figure 4.25 A net with starting and ending transitions

Finally, we present two cases in which the built models are not the same than the net from which the sequence is obtained. The first case corresponds to the original *PN* model in Figure 4.26.a, from which the sequence $S = t_1\ t_2\ t_3\ t_0\ t_2\ t_3\ t_1\ t_2\ t_3\ t_0\ t_2\ t_3\ t_1\ t_2\ t_3\ t_0\ t_2\ t_3\ t_0\ t_2\ t_3\ t_1\ t_2\ t_3\ t_0\ t_2\ t_3\ t_1\ t_2\ t_3\ t_0\ t_2\ t_3\ t_0\ t_2\ t_3\ t_0\ t_2\ t_3\ t_1\ t_2\ t_3\ t_1\ t_2\ t_3\ t_0\ t_2\ t_3\ t_1\ t_2\ t_3\ t_0\ t_2\ t_3\ t_0\ t_2\ t_3\ t_0$ is obtained. The identified model, which is illustrated in Figure 4.26.b is not equal to the original one, but it is equivalent given it represents the same behaviour. Indeed is easy to see that the original model has some redundancy (if we abstract for the observable part).

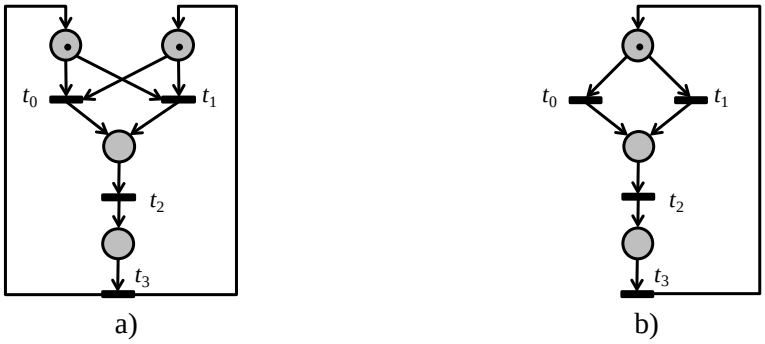


Figure 4.26 Original Free-choice PN and identified model

The second case corresponds to original and identified models in Figure 4.27.a and Figure 4.27.b respectively. The reproduced sequence we have used for identification is $S = t_1\ t_3\ t_4\ t_6\ t_2\ t_3\ t_5\ t_6\ t_1\ t_3\ t_4\ t_6\ t_1\ t_3\ t_4\ t_6\ t_2\ t_3\ t_5\ t_6\ t_2\ t_3\ t_5\ t_6\ t_1\ t_3\ t_4\ t_6\ t_1\ t_3$.

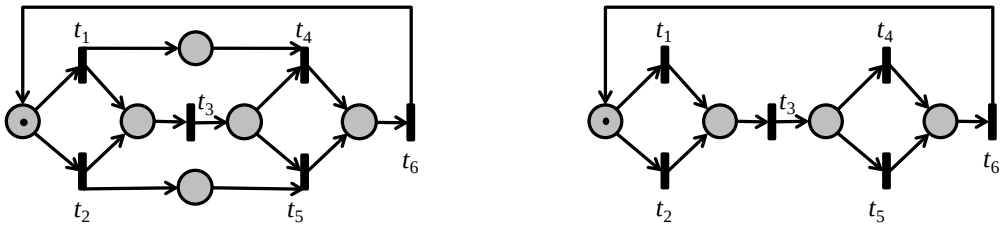


Figure 4.27 Non-identified memory places

Notice that nor the place from transition t_1 to t_4 has not been found, neither the place from t_2 to t_5. This is due to the couples (t_1 , t_4) and (t_2 , t_5) do not appear in the set *Seq* because they are never observed consecutively. Such places can be considered as a memorisation of the choice of firing t_1 or t_2; these places may be found if it is possible to determine the t-invariants from S, but this is out of the reach of this thesis. However, notice that the sequence *S* is executable in the identified model.

4.4. Conclusion

A method to discover the actual input-output relation of PLC controlled discrete event systems has been presented. This yields a concise IPN representation of the compound Plant + Controller behaviour by associating to transitions sufficient conditions on the inputs which represent both the input changed and the current inputs values (execution context). None of the approaches analyzed in Chapter 1 have made such kind of analysis or representation of the input conditions. The obtained structure is remarkably more clear and expressive than the identified models with the stepwise method of Chapter 3.

Chapter 5

Implementation and experimental tests

Abstract. This chapter presents the application of the identification methods to cases studies defined on two experimental sites. First, the proposed methods have been implemented as software tools that allow processing large actual data; such tools are briefly described. Then, the experimental sites are presented, and with the help of the tools, the identification algorithms are tested with input-output sequences obtained during the operation of the closed loop controlled manufacturing processes.

Two tools have been developed to automate the IPN model identification algorithms presented in this thesis (one for each method). In this chapter, we show some of the results obtained with the first and second techniques of chapters 3 and 4 respectively.

5.1. Software tools description

The input data is the same for the two described algorithms and several software components have been used in both tools. However, there are some differences: the stepwise technique needs an identification parameter κ, and each one of the software tools returns a different graphical format for the IPN. All the implementations have been completed with IDE Netbeans 6.5, java jdk 1.6.0.

The user interface can be observed in Figure 5.1. In order to start an identification process, there are several fields that the user must fill in:

- The name of the input file containing the observed I/O sequence. If the input file name text field is mouse clicked, an open dialog is showed and the txt file containing the sequence can be chosen by the user.
- The desired name for the output file. A save dialog appears if the output file name text field is clicked.
- The accuracy identification parameter. The tool for the stepwise method needs the insertion of the desired accuracy parameter κ. There is a text field where the user can provide it.
- The inversed order condition allows being compliant with two different forms of data file (I/O or O/I). There is a check box that must be selected to let the software know if the order is O/I.
- The total elementary inputs number. Also it might happen that the input file does not have a tab to divide input values from output values. In that case, the user must insert the total number of inputs to indicate the software how to divide between input and output signals.
- The index mask. If we want to take into account only some of the inputs or outputs of the I/O vectors, we can use a mask by selecting the corresponding check box and inserting the indices of fields we want to take into account. This allows, for example, masking push buttons or lights that are not mandatory in the identified model.
- The mnemonics of inputs and outputs. The desired names of the outputs and inputs (in that order) are written separated with a blank.
- The same PLC cycle condition. As specified at the end of section 4.2.5, there are two options for the construction of firing conditions: the observation of input and output events at the same PLC cycle or the consideration of previous PLC cycles. There is check box where the user can choose the construction mode for identification.

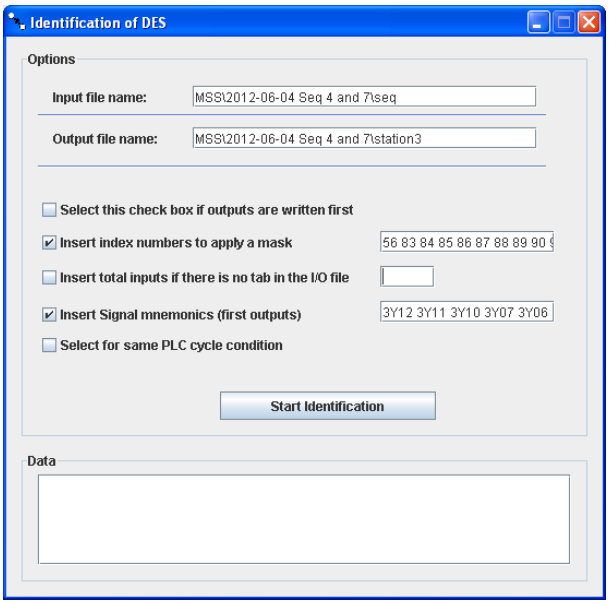


Figure 5.1 User interface

Figure 5.2 shows the global software description. When the button *Start identification* is pressed, the I/O input sequence is read from a file containing several rows, each one representing an I/O vector. Each row is composed of three parts separated by tabs: the instant at which the I/O vector has been stored (which is not used by the proposed algorithms, but could be used in future applications), the values of the inputs and the values of the outputs at each PLC cycle. When a line of such a file is read, time information is ignored and input and output are transformed into binary vectors. The identification algorithm described in Chapter 3 or that in Chapter 4 is then applied and the IPN is constructed in the correspondent output format.

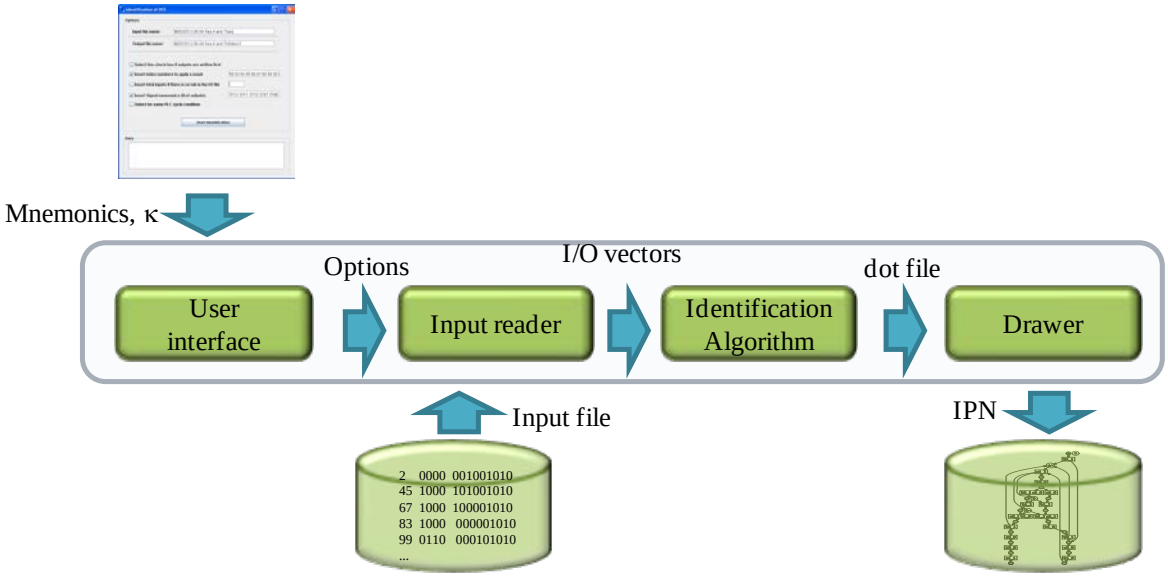


Figure 5.2 Software architecture

For the stepwise method, the creation of the graphic model images has been performed with the hierarchical layout module dot of Graphviz. At the end of the identification, a file written in the format read by such a drawing tool is created. Then, the command dot is invoked from the application to produce the output result, in the form of a jpg or a svg file.

For the statistical method, the XML format has been chosen to represent the IPN models. They can later be opened by the software PIPE.

Once the identification algorithm has been executed, the user interface displays into the text area called Data some information about the identification process, such as the number of transitions and places of the IPN obtained and the execution time for identification.

Now, we present two experimental systems which have been used for testing the identification methods with the help of both tools.

5.2. Interactive Training System for PLC

The Interactive Training System for PLC® (ITS PLC) Professional Edition is a tool for PLC programming which offers virtual systems for education and training in PLC programming. Each system is a behavioural and visual simulation of an industrial system including virtual sensors and actuators, so its state can be sensed by a real PLC. The objective is to program the PLC to control each virtual system as if it was a real system. The sensors and actuators data is exchanged between the PLC and the system by a data acquisition board (DAQ) with 32 I/O isolated channels and USB interface.

For our experimental work, we have chosen the so called *Sorting system*. It transports cases from a feeder to a couple of elevators, sorting them by height (Figure 5.3). It consists of 11 inputs (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10) and 7 outputs (A0, A1, A2, A3, A4, A5, A6).

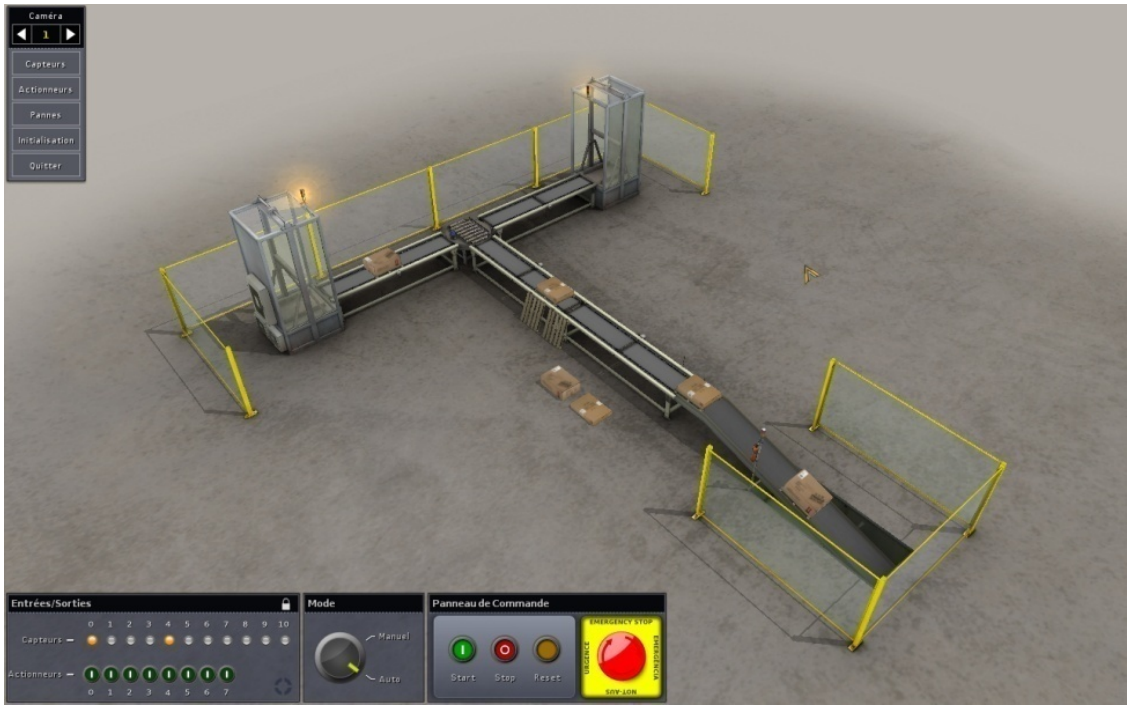


Figure 5.3 The Sorting system from ITS PLC

For the collection of data, a routine written on Python has been launched at the computer with the ITS PLC (see Figure 5.4). It uses the Modbus communication protocol to read values of the inputs and outputs at each PLC cycle. The PLC is a Modicon TSX Premium.

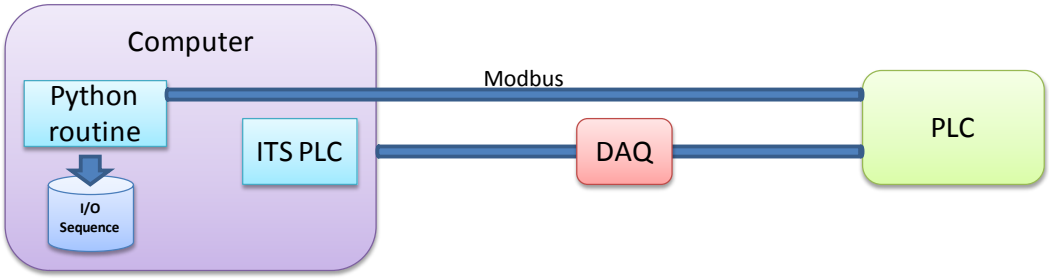


Figure 5.4 Scheme of the data collection procedure

If a I/O vector read is equal to the previous one, it is ignored. Otherwise, the I/O vectors as well as the time value are written into a file. Figure 5.5 shows the experimental environment.

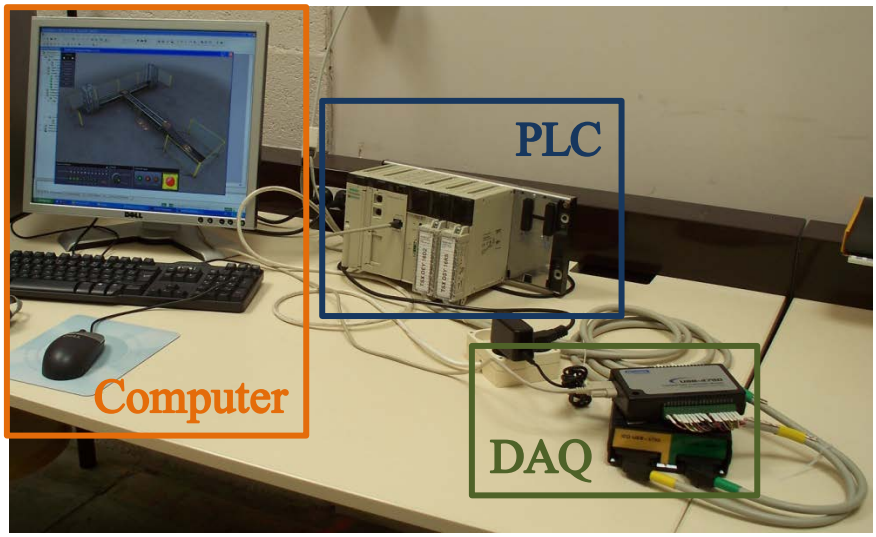
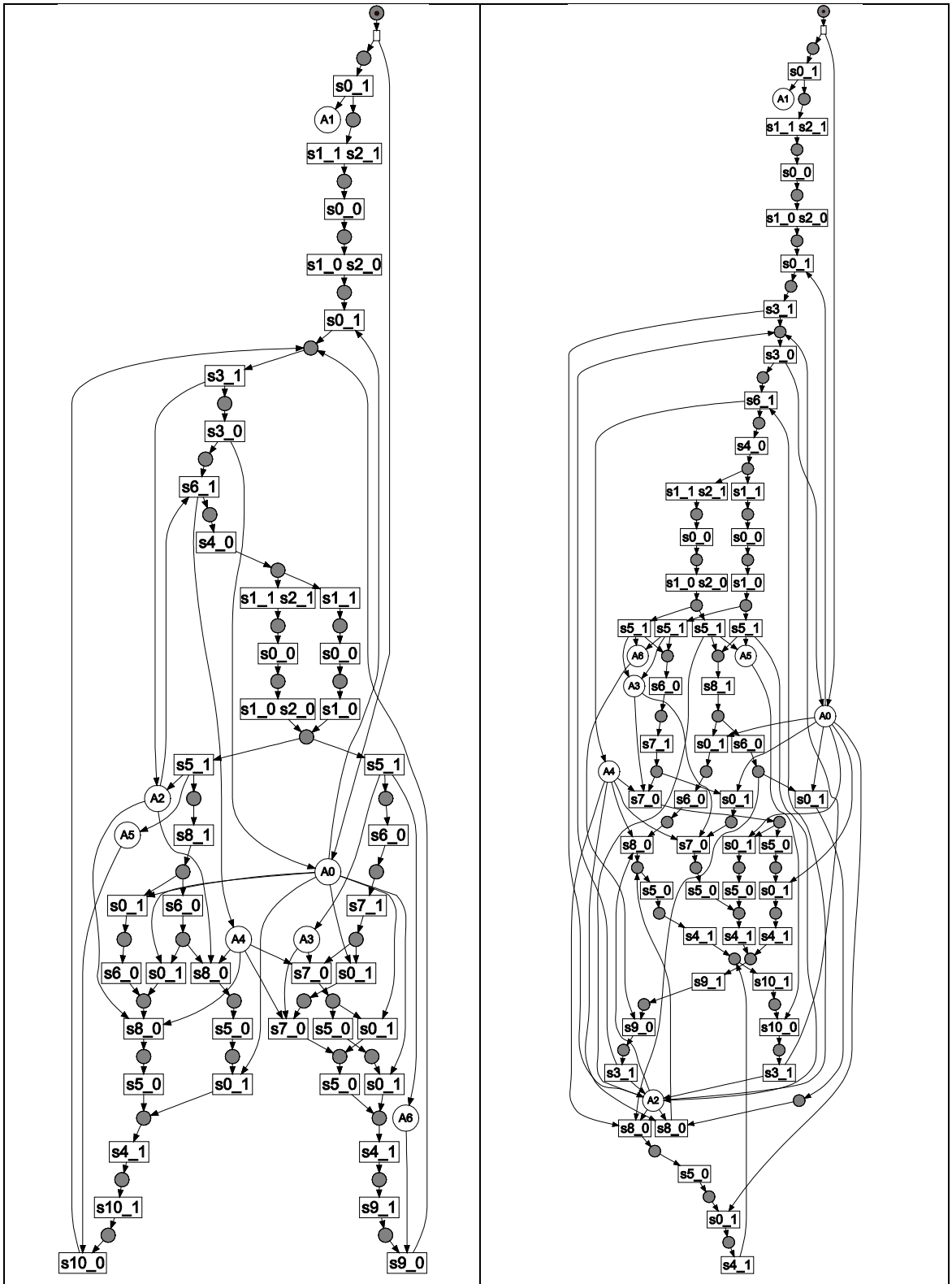


Figure 5.5 Experimental environment

After the treatment of 30 pieces, the data collection has been stopped, giving as result a file containing 472 I/O vectors.

5.2.1. Application of the stepwise method

The stepwise identification procedure from Chapter 3 has been applied with different values of κ. Identified models for κ = 1 and κ = 2 are showed in Figure 5.6. The execution time for the identification was 156ms and 157ms respectively.



a) Identified model with κ = 1 | b) Identifed model with κ = 2

Figure 5.6 Identified models for the Sorting system

Observe that increasing the value of κ provokes that several transition paths are created. However, the net computed with κ = 2 does not represent a significant change from the net computed with κ = 1 and thus, we decided that it is not necessary to compute new nets with largest values of κ, and the value κ = 1 is enough.

By expertise knowledge, we have been able to validate models, checking that they are able to reproduce the behaviour of the plant and controller compound. However, expressiveness of the nets is reduced: it is hard for someone not familiarised with the system to know how the system works by looking at the models.

A compact model (Figure 5.7) has been created following the procedures described in section 3.2. The execution time to produce this model has been 125ms. Even if the model is easier to read, notice how several input events are related to a single transition and as a consequence, it is not clear to distinguish which input sensors belong to a condition and which ones are the causal events that actually produce the setting of the output values to the actuators.

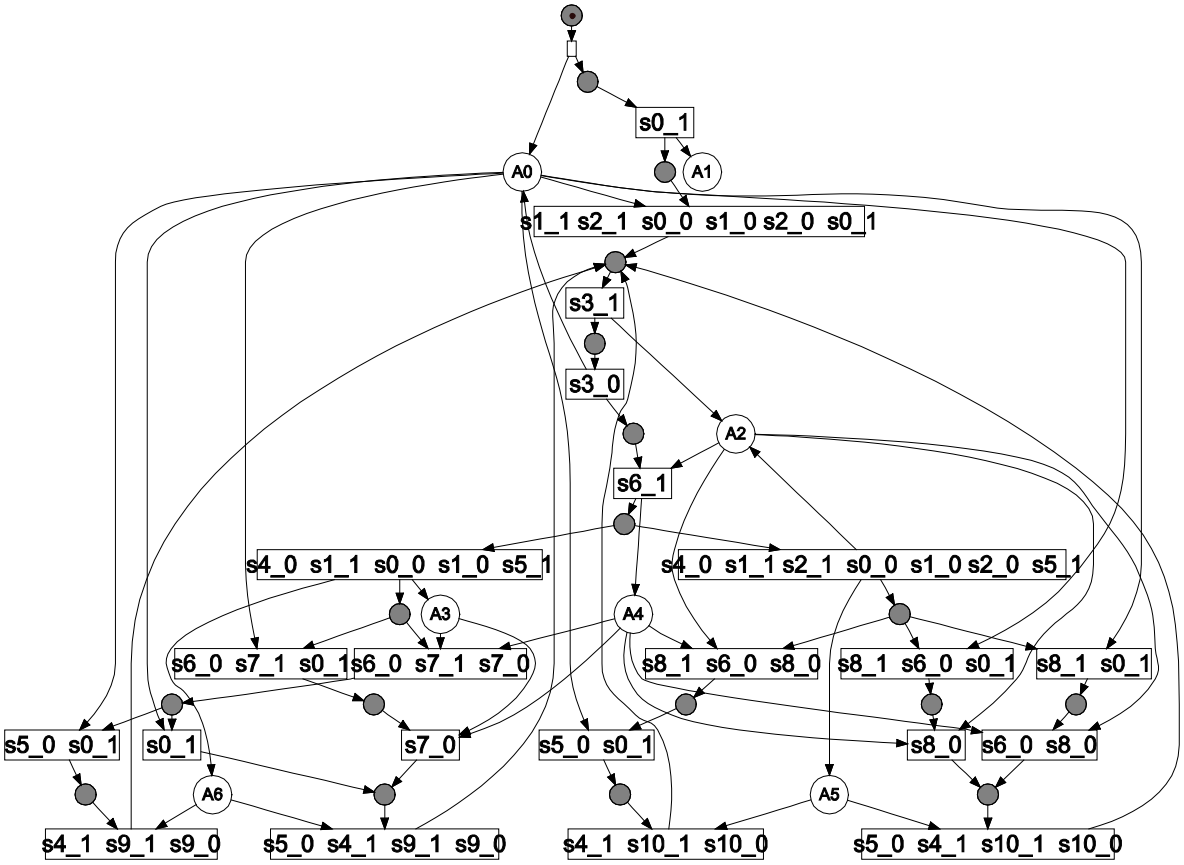


Figure 5.7 Reduced model for the sorting system

5.2.2. Application of the statistical method

Consider now the results obtained by the statistical method of Chapter 4. Direct and context matrices contain values in Figure 5.8 and Figure 5.9 respectively.

	A0_1	A0_0	A1_1	A1_0	A2_1	A2_0	A3_1	A3_0	A4_1	A4_0	A5_1	A5_0	A6_1	A6_0
s0_1	0.000	1.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s0_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s1_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s1_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s2_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s2_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s3_1	0.000	0.000	0.000	0.000	0.630	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s3_0	0.967	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s4_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s4_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s5_1	0.000	0.000	0.000	0.000	0.370	0.000	1.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000
s5_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s6_1	0.000	0.000	0.000	0.000	0.000	0.630	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
s6_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s7_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s7_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.414	0.000	0.000	0.000	0.000
s8_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s8_0	0.000	0.000	0.000	0.000	0.000	0.370	0.000	0.000	0.000	0.586	0.000	0.000	0.000	0.000
s9_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s9_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
s10_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s10_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000

Figure 5.8 Direct Causality Matrix of the Sorting system

	A0_1	A0_0	A1_1	A1_0	A2_1	A2_0	A3_1	A3_0	A4_1	A4_0	A5_1	A5_0	A6_1	A6_0
s0=1	0.967	1.000	1.000	0.000	0.630	0.717	0.000	0.250	1.000	0.241	0.000	1.000	0.000	1.000
s0=0	0.033	0.000	0.000	0.000	0.370	0.283	1.000	0.750	0.000	0.759	1.000	0.000	1.000	0.000
s1=1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s1=0	1.000	1.000	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
s2=1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s2=0	1.000	1.000	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
s3=1	0.000	0.000	0.000	0.000	0.630	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s3=0	1.000	1.000	1.000	0.000	0.370	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
s4=1	1.000	0.033	1.000	0.000	0.630	0.630	0.000	0.000	1.000	0.000	0.000	1.000	0.000	1.000
s4=0	0.000	0.967	0.000	0.000	0.370	0.370	1.000	1.000	0.000	1.000	1.000	0.000	1.000	0.000
s5=1	0.000	0.300	0.000	0.000	0.370	0.370	1.000	1.000	0.000	1.000	1.000	0.000	1.000	0.000
s5=0	1.000	0.700	1.000	0.000	0.630	0.630	0.000	0.000	1.000	0.000	0.000	1.000	0.000	1.000
s6=1	0.000	0.100	0.000	0.000	0.370	0.630	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
s6=0	1.000	0.900	1.000	0.000	0.630	0.370	0.000	1.000	0.000	1.000	0.000	1.000	0.000	1.000
s7=1	0.000	0.100	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s7=0	1.000	0.900	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
s8=1	0.000	0.133	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s8=0	1.000	0.867	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
s9=1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s9=0	1.000	1.000	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
s10=1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
s10=0	1.000	1.000	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Figure 5.9 Indirect Causality Matrix of the Sorting system

By analysis of the I/O sequence and direct and indirect matrices, the statistical procedure has computed the model fragments in Figure 5.10. Notice how the number of transitions has been reduced with respect to models in Figure 5.6. We have verified that the obtained firing functions are correct.

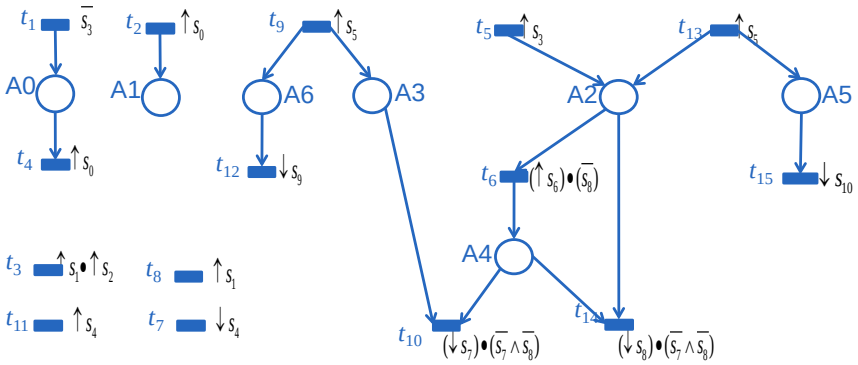


Figure 5.10 Observable behaviour identified for the Sorting system

By the first step of the statistical procedure we have also computed the corresponding transition sequence:

S = t_1 t_2 t_3 t_4 t_5 t_1 t_6 t_7 t_8 t_9 t_4 t_{10} t_{11} t_{12} t_5 t_1 t_6 t_7 t_3 t_{13} t_4 t_{14} t_{11} t_{15} t_5 t_1 t_6 t_7 t_3 t_9 t_{10} t_4 t_{11} t_{12} t_5 t_1 t_6 t_7 t_8 t_9 t_4 t_{10} t_{11} t_{12} t_5 t_1 t_6 t_7 t_8 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_8 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_3 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_8 t_9 t_{10} t_4 t_{11} t_{12} t_5 t_1 t_6 t_7 t_3 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_3 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_8 t_9 t_{10} t_4 t_{11} t_{12} t_5 t_1 t_6 t_7 t_3 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_3 t_9 t_{10} t_4 t_{11} t_{12} t_5 t_1 t_6 t_7 t_8 t_9 t_{10} t_4 t_{11} t_{12} t_5 t_1 t_6 t_7 t_8 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_8 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_3 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_8 t_9 t_{10} t_4 t_{11} t_{12} t_5 t_1 t_6 t_7 t_8 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_8 t_9 t_{10} t_4 t_{11} t_{12} t_5 t_1 t_6 t_7 t_3 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_8 t_9 t_{10} t_4 t_{11} t_{12} t_5 t_1 t_6 t_7 t_3 t_{13} t_{14} t_4 t_{11} t_{15} t_5 t_1 t_6 t_7 t_3 t_9 t_{10} t_4 t_{11} t_{12}

The sequence contains following pairs of consecutive transitions:

t_1<t_2, t_2<t_3, t_3<t_4, t_4<t_5, t_5<t_1, t_1<t_6, t_6<t_7, t_7<t_8, t_8<t_9, t_9<t_4, t_4<t_{10}, t_{10}<t_{11}, t_{11}<t_{12}, t_{12}<t_5, t_7<t_3, t_3<t_{13}, t_{13}<t_4, t_4<t_{14}, t_{14}<t_{11}, t_{11}<t_{15}, t_{15}<t_5, t_3<t_9, t_9<t_{10}, t_{10}<t_4, t_4<t_{11}, t_8<t_{13}, t_{13}<t_{14}, t_{14}<t_4

The following *PS* sets have been computed:

PS(t_1) = { t_1 t_4 t_5 }
PS(t_2) = { }
PS(t_3) = { t_3 t_4 t_5 t_1 t_6 t_7 t_{11} }
PS(t_4) = { t_4 t_5 t_1 t_6 t_7 }
PS(t_5) = { t_5 t_1 t_6 t_7 t_4 t_{11} }
PS(t_6) = { t_6 t_7 t_4 t_{11} t_5 t_1 }
PS(t_7) = { t_7 t_4 t_{11} t_5 t_1 t_6 }
PS(t_8) = { t_8 t_4 t_{11} t_5 t_1 t_6 t_7 }
PS(t_9) = { t_9 t_4 t_{10} t_{11} t_{12} t_5 t_1 t_6 t_7 }
PS(t_{10}) = { t_{10} t_{11} t_{12} t_5 t_1 t_6 t_7 t_4 t_9 }
PS(t_{11}) = { t_{11} t_5 t_1 t_6 t_7 t_4 }
PS(t_{12}) = { t_{12} t_5 t_1 t_6 t_7 t_4 t_{11} t_9 t_{10} }
PS(t_{13}) = { t_{13} t_4 t_{14} t_{11} t_{15} t_5 t_1 t_6 t_7 }
PS(t_{14}) = { t_{14} t_{11} t_{15} t_5 t_1 t_6 t_7 t_{13} }
PS(t_{15}) = { t_{15} t_5 t_1 t_6 t_7 t_4 t_{11} t_{13} t_{14} }

Based on the computed data, the model in Figure 5.11 has been inferred. We have shaded non-observable places to facilitate the reading of the model. The initial marking has been computed backwards in order to reproduce the sequence *S*. However, start of the sequence 't_1 t_2 t_3 t_4', would lead to non-safe markings. This is due to the first events occurring in *S* are actually to initialise the machine and the remainders correspond to the repetitive behaviour of the net (from the 5th transition t_5 t_1 t_6 t_7 t_8…). An initialization sequence replicating the first four transitions of the *S* can be added to the model to allow the firing of the whole sequence (see Figure 5.12).

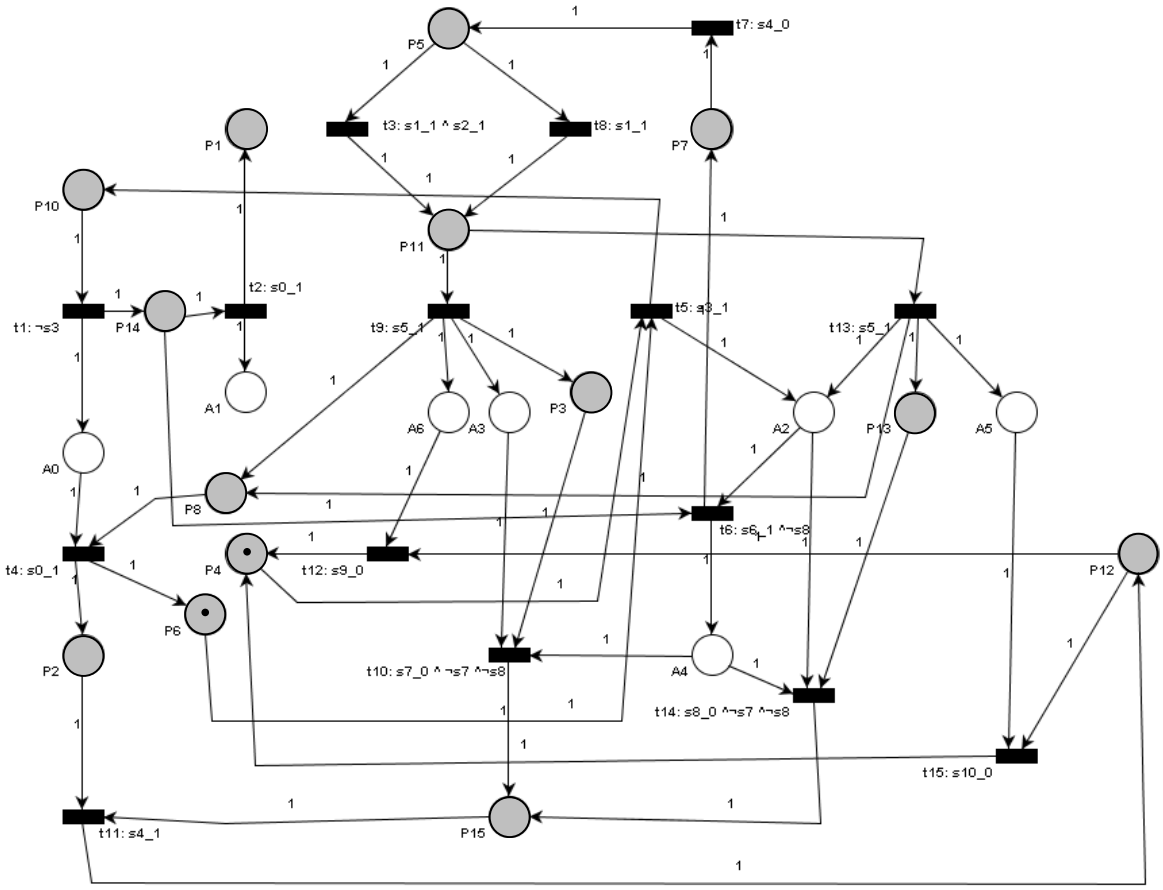


Figure 5.11 Final IPN model for the Sorting system

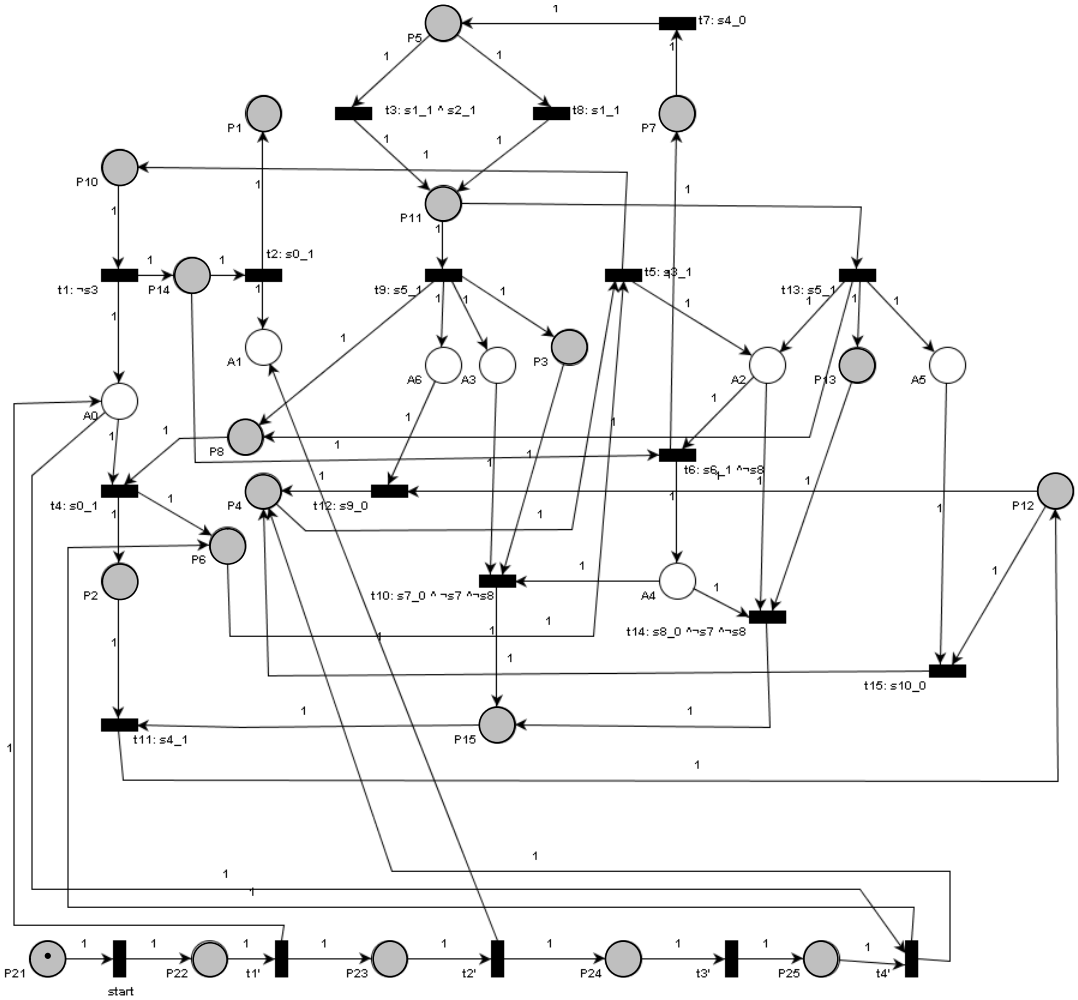


Figure 5.12 IPN model for the Sorting system including initialization sequence

5.3. Assembly System

The second case study is an experimental facility in LURPA (see Figure 5.13). It is called Mechatronics Standard System (MSS) from Bosch: an assembly machine composed by four stations. The machine treats several gearwheels in order to insert or remove bearings into them. At the end of the treatment, the work pieces are sorted by material into a warehouse.

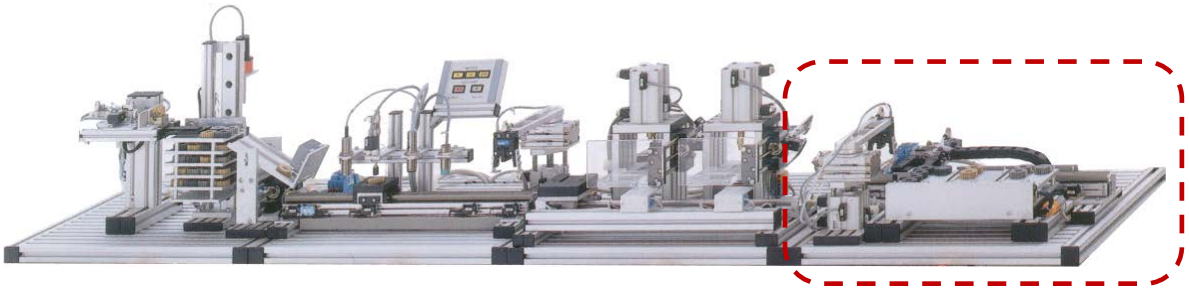


Figure 5.13 Mechatronics Standard System

Figure 5.14 shows the MSS installations at LURPA. The controller communicates with the plant via Ethernet [Roth, 2010c]. The data collection has been made with a routine in Python allowing a computer to acquire the input sequence from the automata through the Modbus communication protocol.

Figure 5.14 MSS experimental environment

The machine has several operation modes: the bearings can be removed or inserted (or both), and the sorting mode can be by material or by arriving order. For purposes of this work, only the fourth station was identified (dotted part of the Figure 5.13), which is in charge of the arrangement and storage of the work pieces. We have made the data collection with a scenario where the gearwheels have been sorted by material in one of the three available pallets. An I/O sequence of 63,797 vectors has been stored. The index of the inputs and outputs belonging to the fourth station were inserted in the user interface. They correspond to 16 inputs (3B11, 4S24, 4S23, 4S22, 4S21, 4S20, 4S17, 4B16, 4B15, 4B14, 4B13, 4B12, 4B11, 4B10, 4B07, 4S06) and 6 outputs (4Y11, 4Y10, 4Y07, 4Y06, 4K05, 4K04).

5.3.1. Application of the stepwise method

The stepwise algorithm has processed the input-output sequence using different values of κ from 1 to 6. In general it is not possible to establish a-priori the value of κ, since it is assumed that the system is unknown. However, in practice the identification procedure can be applied using several values of κ (because it is not time consuming). Compact models allow a first approximation to the understanding of the system functioning, whilst larger models provide a more precise description. However, one more time, larger models are huge and close to automata, and the expressiveness of the Petri nets is not exploited.

Since the obtained models are huge, we present only the identified models in Figure 5.15 for $\kappa = 1$ and in Figure 5.16 for $\kappa = 2$. The size of the rest of identified models are summarised in Table 5.1. The execution time of the identification procedure is also included to provide an idea of the performance of the algorithm. The tests have been performed in a computer based on an Intel Core 2 Duo T7300 processor at 3.00 GHz with 2.00 GB of RAM under Windows XP Professional 2002 Service Pack 2. The time has been measured excluding the execution of the Graphviz visualisation software.

The compacting procedure has been also applied, yielding the model in Figure 5.17. However, once again, the input-output relationship is hidden by the long computed transitions.

κ	Transitions	Places	Total of nodes	Δ	Processing time
1	142	85	227		3093 ms
2	218	152	370	143	3094 ms
3	305	227	532	162	3141 ms
4	396	314	710	178	3297 ms
5	498	404	902	192	3375 ms
6	606	508	1114	212	3469 ms
7	718	615	1333	219	3531 ms

Table 5.1 Size of identified models for different values of κ

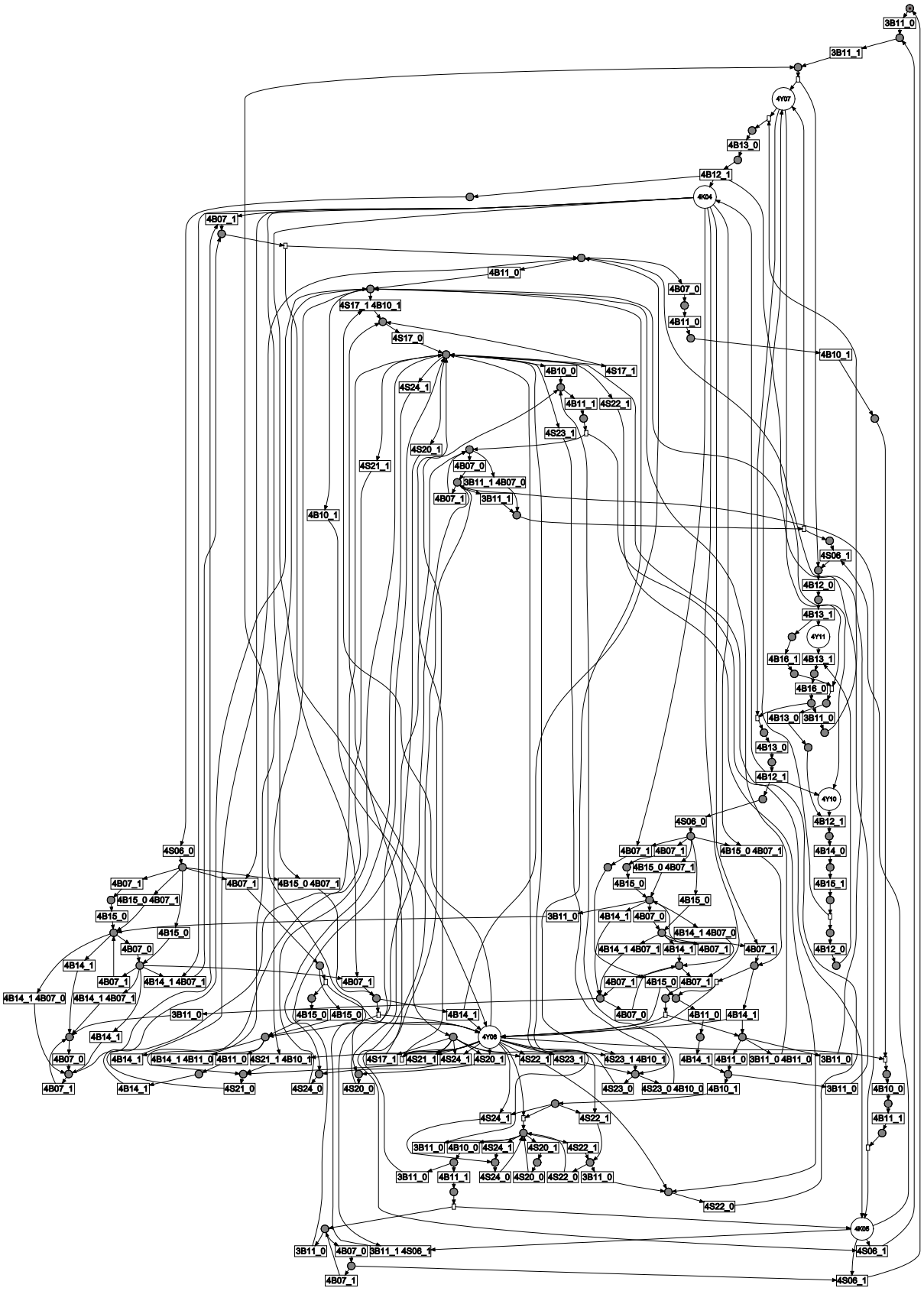


Figure 5.15 Identified model with κ = 1

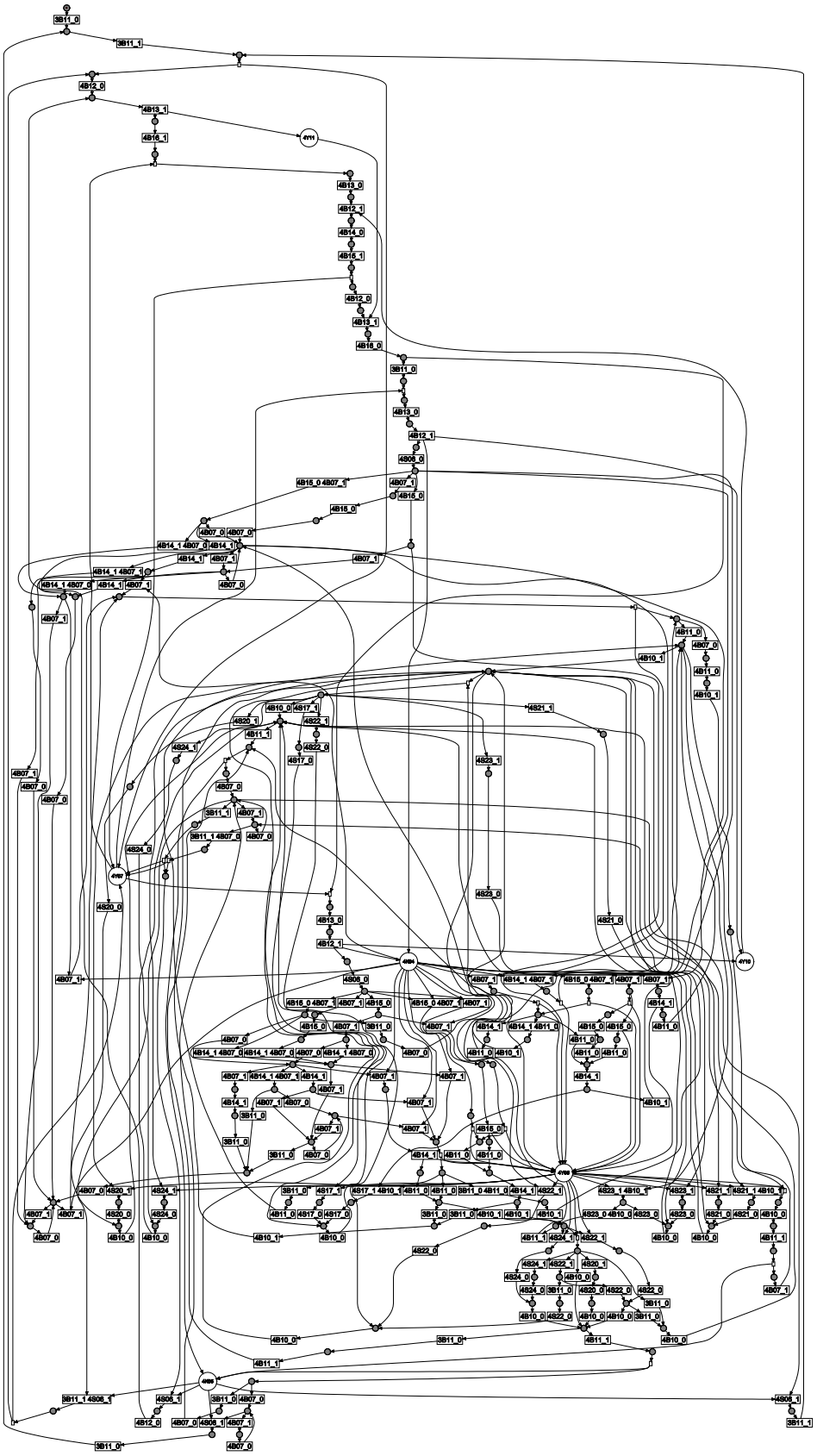


Figure 5.16 Identified model with κ = 2

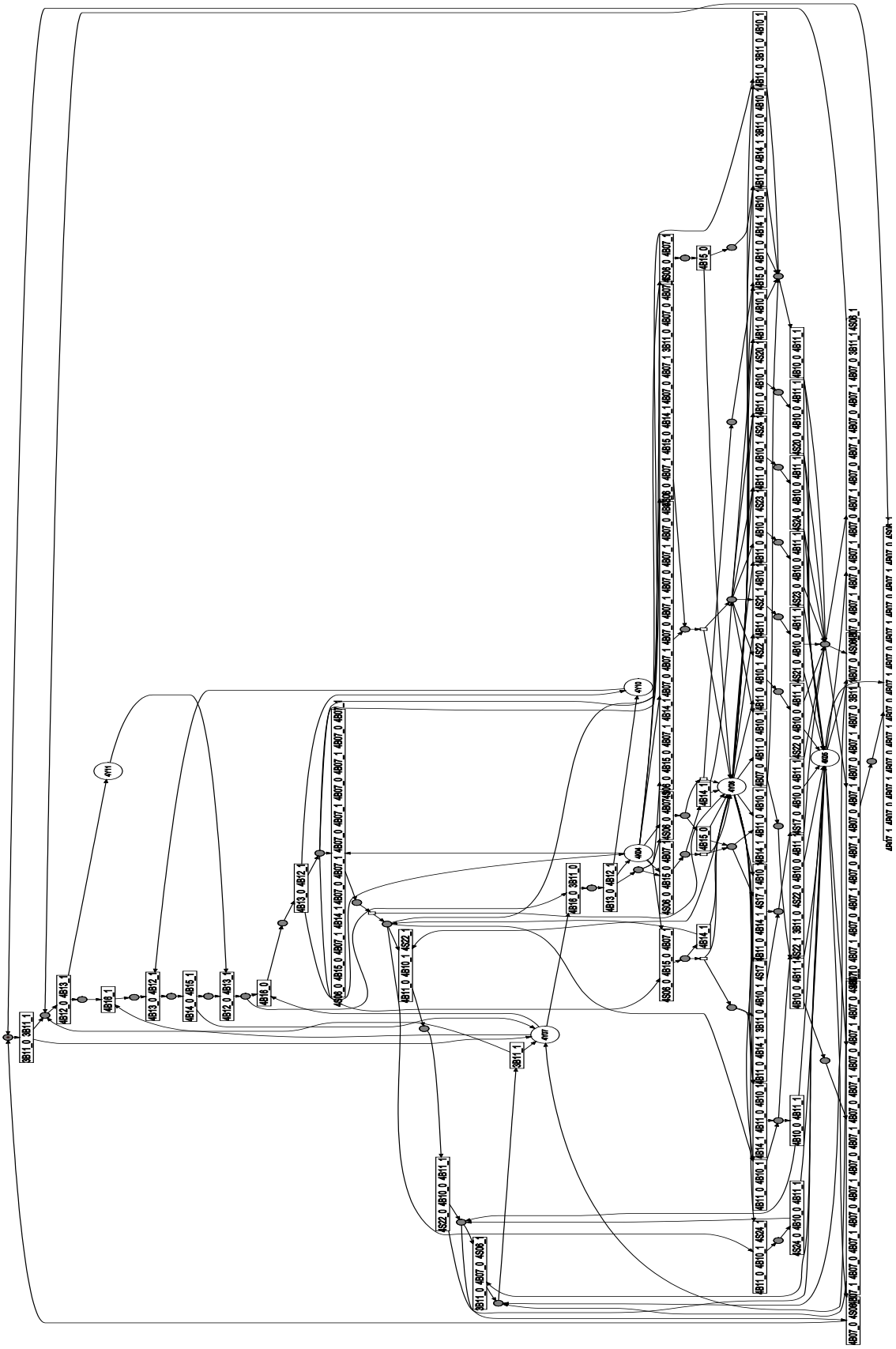


Figure 5.17 Compacted model for the sorting system

5.3.2. Application of the statistical method

For the statistical method, the *DCM* in Figure 5.18 and the *ICM* in Figure 5.19 have been computed.

	4Y11_1	4Y11_0	4Y10_1	4Y10_0	4Y07_1	4Y07_0	4Y06_1	4Y06_0	4K05_1	4K05_0	4K04_1	4K04_0
3B11_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.004	0.000	0.000
3B11_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S24_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.006	0.000	0.000	0.000	0.000
4S24_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S23_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.010	0.000	0.000	0.000	0.000
4S23_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S22_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.006	0.000	0.000	0.000	0.000
4S22_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S21_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.004	0.000	0.000	0.000	0.000
4S21_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S20_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.000	0.000	0.000	0.000
4S20_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S17_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.012	0.000	0.000	0.000	0.000
4S17_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B16_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B16_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B15_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B15_0	0.000	0.000	0.000	0.000	0.000	0.000	0.006	0.000	0.000	0.000	0.000	0.077
4B14_1	0.000	0.000	0.000	0.000	0.000	0.000	0.004	0.000	0.000	0.000	0.000	0.023
4B14_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B13_1	1.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B13_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B12_1	0.000	0.000	1.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
4B12_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B11_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B11_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B10_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.015	0.000	0.000	0.000	0.000
4B10_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B07_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
4B07_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S06_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
4S06_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Figure 5.18 Direct Causality Matrix for the MSS

	4Y11_1	4Y11_0	4Y10_1	4Y10_0	4Y07_1	4Y07_0	4Y06_1	4Y06_0	4K05_1	4K05_0	4K04_1	4K04_0
3B11=1	1.000	1.000	0.098	1.000	1.000	0.549	0.088	0.062	0.044	0.056	0.098	0.088
3B11=0	0.000	0.000	0.902	0.000	0.000	0.451	0.912	0.938	0.956	0.944	0.902	0.912
4S24=1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.006	0.000	0.000	0.000	0.000
4S24=0	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.994	1.000	1.000	1.000	1.000
4S23=1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.010	0.000	0.000	0.000	0.000
4S23=0	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.990	1.000	1.000	1.000	1.000
4S22=1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.006	0.000	0.000	0.000	0.000
4S22=0	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.994	1.000	1.000	1.000	1.000
4S21=1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.004	0.000	0.000	0.000	0.000
4S21=0	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.996	1.000	1.000	1.000	1.000
4S20=1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.000	0.000	0.000	0.000
4S20=0	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.998	1.000	1.000	1.000	1.000
4S17=1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.012	0.000	0.000	0.000	0.000
4S17=0	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.988	1.000	1.000	1.000	1.000
4B16=1	0.000	1.000	0.000	1.000	0.500	0.500	0.000	0.000	0.000	0.000	0.000	0.000
4B16=0	1.000	0.000	1.000	0.000	0.500	0.500	1.000	1.000	1.000	1.000	1.000	1.000
4B15=1	0.000	1.000	1.000	0.000	0.500	0.500	0.002	0.000	0.000	0.000	1.000	0.008
4B15=0	1.000	0.000	0.000	1.000	0.500	0.500	0.998	1.000	1.000	1.000	0.000	0.992
4B14=1	1.000	0.000	0.000	1.000	0.500	0.500	0.833	1.000	1.000	1.000	0.000	0.829
4B14=0	0.000	1.000	1.000	0.000	0.500	0.500	0.167	0.000	0.000	0.000	1.000	0.171
4B13=1	1.000	1.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
4B13=0	0.000	0.000	1.000	1.000	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000
4B12=1	0.000	0.000	1.000	1.000	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000
4B12=0	1.000	1.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
4B11=1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	1.000	1.000	1.000	1.000
4B11=0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
4B10=1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
4B10=0	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000	1.000	1.000	1.000	1.000
4B07=1	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.992	0.992	0.000	0.000	1.000
4B07=0	1.000	1.000	1.000	1.000	1.000	1.000	0.000	0.008	0.008	1.000	1.000	0.000
4S06=1	1.000	1.000	1.000	1.000	0.995	1.000	0.000	0.000	0.000	1.000	1.000	0.000
4S06=0	0.000	0.000	0.000	0.000	0.005	0.000	1.000	1.000	1.000	0.000	0.000	1.000

Figure 5.19 Indirect Causality Matrix for the MSS

Notice that the DCM matrix column corresponding to output event 4Y07_1 is zero and thus the computed firing conditions would be empty. The same occurs for output events 4Y07_0, 4Y06_1 and 4K05_1: the corresponding DCM matrix columns are almost zero and as a consequence they yield to empty firing conditions. This situation is the case specified at the end of section 4.2.5, where instead of considering that the input and output events must occur at the same PLC cycle, we must look at the input events occurring in the previous event vector. We have computed the probability values considering the previous event vector to construct the called One Step Direct Matrix in Figure 5.20.

	4Y11_1	4Y11_0	4Y10_1	4Y10_0	4Y07_1	4Y07_0	4Y06_1	4Y06_0	4K05_1	4K05_0	4K04_1	4K04_0
3B11_1	0.000	0.000	0.000	0.000	0.500	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3B11_0	0.000	0.000	0.000	0.000	0.000	0.451	0.000	0.000	0.000	0.000	0.000	0.000
4S24_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S24_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S23_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S23_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S22_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S22_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S21_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S21_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S20_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S20_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S17_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S17_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B16_1	0.000	0.000	0.000	0.000	0.000	0.500	0.000	0.000	0.000	0.000	0.000	0.000
4B16_0	0.000	0.000	0.000	0.000	0.000	0.049	0.000	0.000	0.000	0.000	0.000	0.000
4B15_1	0.000	0.000	0.000	0.000	0.500	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B15_0	0.000	0.000	0.000	0.000	0.000	0.000	0.077	0.000	0.000	0.000	0.000	0.081
4B14_1	0.000	0.000	0.000	0.000	0.000	0.000	0.023	0.006	0.000	0.000	0.000	0.131
4B14_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B13_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B13_0	0.000	0.000	1.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
4B12_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B12_0	1.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B11_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
4B11_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.008	0.000	0.000	0.000	0.000
4B10_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.985	0.000	0.000	0.000	0.000
4B10_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4B07_1	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
4B07_0	0.000	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.990	0.000	0.748
4S06_1	0.000	0.000	0.000	0.000	0.002	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4S06_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.085

Figure 5.20 One Step Direct Matrix for the MSS

Using such a matrix instead of the *DCM* matrix, we have been able to compute the lacking firing functions and the correspondent fragments of Figure 5.21 have been constructed.

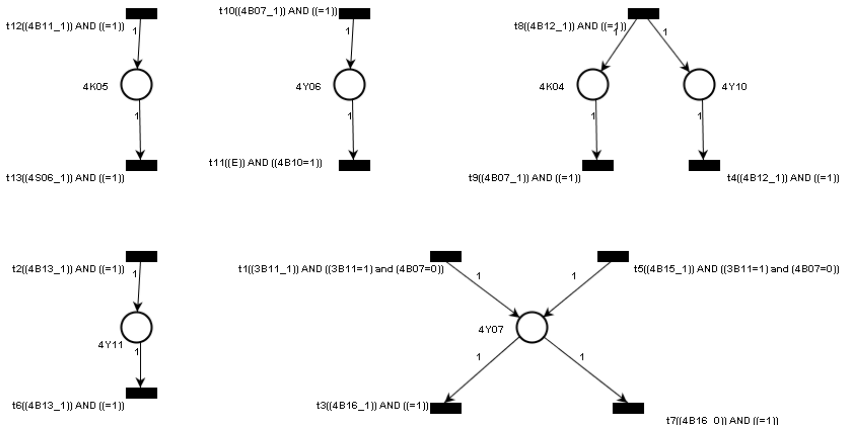


Figure 5.21 Observable behaviour computed for the MSS

The length of the computed sequence *S* is 6,240; this is why it is not showed here. From such a sequence and the computed fragments, the second part of the statistical method has allowed us to build the model in Figure 5.22. The whole identification procedure has taken 7.5s. Notice how the constructed model is more compact and expressive than those constructed by the stepwise method.

We have verified that the model reproduces the observed behaviour: a car arrives with a gearwheel and the rotary gripper goes down to take it. Once the gripper is down, it holds the piece and it starts going up again. Once the gripper up, it starts to swivel to the right. Once arriving to the rightmost position, it goes down again to depose the gearwheel into another car. Once the piece is released, the gripper goes up again and to the left, to return to its initial position. Meanwhile, the car moves until the storage area. There, the car stops and a cylinder is pushed until the gearwheel is in the warehouse.

Then, the cylinder is completely retracted and then the car goes back to its initial position.

Notice how the concurrence between the arriving of a new piece (t1) and the arrangement of the last one (t13) has been captured in the model.

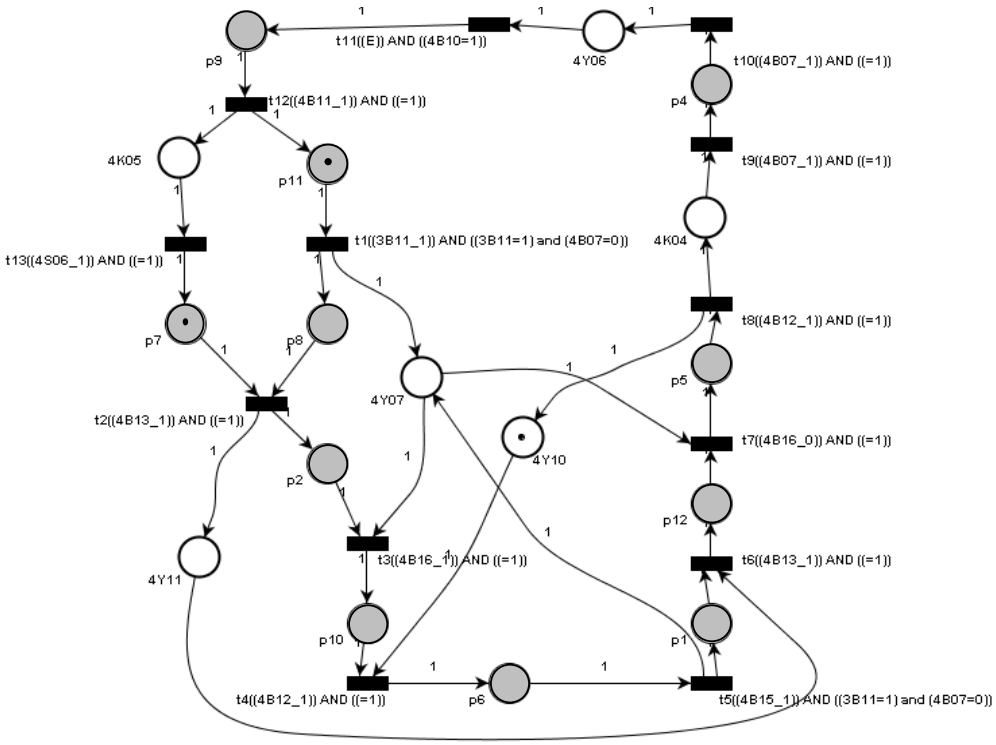


Figure 5.22 Final model for the MSS

5.4. Conclusion

The software tools implementing the algorithms described in this thesis have been tested on two experimental sites with several other case studies. The results have shown that both methods are able to deal with real systems; on the one hand the stepwise method yields detailed models that grows proportionally the system size; on the other hand, the statistical method produces remarkably more clear and expressive models than those synthesized with the stepwise method, because the models are directly expressed in the structure of the IPN. None of approaches considered in the related work of this thesis allow discovering such kind of models.

Conclusions

In this thesis the problem of black-box identification of reactive Discrete Event Systems (DES) has been addressed. Two methods for building Interpreted Petri Net models from a single input-output sequence observed during the DES operation have been conceived and implemented.

The first one is a stepwise method based on the inference of cycles from a state-equivalence definition. This approach allows obtaining IPN models from a single input-output sequence that exhibits the closed loop behaviour of PLC-based controlled plants. The proposed technique builds progressively the IPN approximating the compound controller-plant behaviour, which can be detailed for controller redesign or model-based diagnosis purposes. The resulting procedure operates considering a parameter κ, defining the number of past events to consider, such that in the output language of the obtained IPN only and all observed output sequences of length $\kappa + 1$ are represented.

The second method allows discovering the actual input-output relation of PLC controlled discrete event systems. The technique allows building a concise IPN model in which the transitions are labelled with sufficient conditions on the inputs which represent both the input changed and the inputs execution context. The obtained structure is remarkably more clear and expressive than that synthesized with the first method, because it is directly expressed in the structure of the IPN. Neither the first methodology nor the approaches considered in the related work allow discovering such kind of input conditions.

The methods herein proposed allow dealing with complex automated DES because they take into account technological characteristics of actual controlled systems, and because they are based on efficient algorithms. This feature is not still addressed in current literature on the matter, reviewed in this thesis, in which several features considered in the current stated problem have not been dealt. Both methods are complementary; whilst the first one represents in detail the observed input changes yielding large IPN models, the second method captures in the transition labelling functions the input conditions, producing more compact IPN models that exhibit clearly the concurrent behaviour.

The algorithms issued from these methods have been implemented as software tools and tested on two experimental case studies which are very close to actual industrial discrete event processes. The performed tests reveal the efficiency of the methods when data including thousands of input-output vectors are processed in few seconds.

Due that is a black-box approach the obtained models represent the observed behaviour, consequently when the observation has been made for a long time, the IPN approximates closely the actual behaviour. Afterwards this model can be completed using available knowledge on the process.

Although the presented methods go ahead the previous identification techniques regarding most of the analysed features, several issues can be addressed for extending the work described herein:

- The hypothesis that constraints the inputs and outputs to binary signals could be relaxed.

- The inclusion of timers or counters within the logic of the controller is a feature that appears often in actual industrial discrete manufacturing systems. This feature states interesting problems that implies the identification of non regular languages.

- The inference of other information about the system, like PN invariants, could be done.

Publications

During the thesis research development the following papers have been published and written.

1. A. P. Estrada-Vargas, E. López-Mellado, J.-J. Lesage. “A Comparative Analysis of Recent Identification Approaches for Discrete-Event Systems”, Mathematical Problems in Engineering, Hindawi Pub. Corp. Vol. 2010, Article ID 453254, 21 pages, August, 2010 doi:10.1155/2010/453254.
2. A. P Estrada-Vargas, E. Lopez-Mellado, J.-J. Lesage. "An Identification Method for PLC-based Automated Discrete Event Systems". Proc. of IEEE Int. Conference on Decision and Control, pp.6740-6746. Atlanta E. U. Dec 2010.
3. A. P. Estrada-Vargas, J.-J. Lesage, E. Lopez-Mellado. “Stepwise Identification of Automated Discrete Manufacturing Systems” Proc. of IEEE Int. Conference on Emergency Technologies and Factory Automation, pp. 1-8, Toulouse, France. Sep 2011.
4. A. P. Estrada-Vargas, J.-J. Lesage, E. Lopez-Mellado. “Identification of Industrial Automation Systems: Building Compact and Expressive Petri Net Models from Observable Behaviour”, 2012 American Control Conference, pp. 6095 - 6101, Montréal, Canada, Jun 2012.
5. A. P. Estrada-Vargas, J.-J. Lesage, E. Lopez-Mellado. "Identification de SED au moyen de réseaux de Petri : une approche pour la représentation structurelle des comportements observés", 2012 Conférence Internationale Francophone d’Automatique, pp. 919 - 924, Grenoble, France, Jul 2012.
6. A. P. Estrada-Vargas, E. Lopez-Mellado, J.-J. Lesage. ”Input-Output Identification of Controlled Discrete Manufacturing Systems”, International Journal of Systems Science. Published on-line September 18, 2012, DOI:10.1080/00207721.2012.724098.
7. A. P. Estrada-Vargas, J.-J. Lesage, E. López-Mellado. “A Stepwise Method for Identification of Controlled Discrete Manufacturing Systems”, International Journal of Computer Integrated Manufacturing (submitted).
8. A. P. Estrada-Vargas, E. Lopez-Mellado, J.-J. Lesage. “Automated Modelling of Reactive Discrete Event Systems from External Behavioural Data”. IEEE Int. Conf. on Electronics, Communications and Computers, Cholula, Mexico, Mar 2013 (submitted).

Appendix A. Interpreted Petri Nets

This appendix contains the basic concepts and notation of PN and IPN used in this work.

Definition A.1. An ordinary Petri Net structure G is a bipartite digraph represented by the 4-tuple $G = (P, T, I, O)$ where: $P = \{p_1, p_2, ..., p_{|P|}\}$ and $T = \{t_1, t_2, ..., t_{|T|}\}$ are finite sets of vertices named places and transitions respectively; $I(O) : P \times T \rightarrow \{0,1\}$ is a function representing the arcs going from places to transitions (from transitions to places).

The incidence matrix of G is $C = C^+ - C^-$, where $C^- = [c_{ij}^-]$; $c_{ij}^- = I(p_i, t_j)$; and $C^+ = [c_{ij}^+]$; $c_{ij}^+ = O(p_i, t_j)$ are the pre-incidence and post-incidence matrices respectively.

A marking function $M : P \rightarrow \mathbb{Z}^+$ represents the number of tokens residing inside each place; it is usually expressed as an $|P|$-entry vector. $\mathbb{Z}^+$ is the set of nonnegative integers.

Definition A.2. A Petri Net system or Petri Net (PN) is the pair $N = (G,M_0)$, where G is a PN structure and M_0 is an initial marking.

In a PN system, a transition t_j is *enabled* at marking M_k if $\forall p_i \in P$, $M_k(p_i) \geq I(p_i, t_j)$; an enabled transition t_j can be fired reaching a new marking M_{k+1} . This behaviour is represented as $M_k \xrightarrow{t_j} M_{k+1}$. The new marking can be computed as $M_{k+1} = M_k + Cv_k$, where $v_k(i) = 0$, $i \neq j$, $v_k(j) = 1$; this equation is called the PN state equation. The reachability set of a PN is the set of all possible reachable markings from M_0 firing only enabled transitions; this set is denoted by $R(G,M_0)$.

Now it is defined IPN, an extension to PN that allows associating input and output signals to PN models. Two definitions are used in this work.

Definition A.3. An IPN (Q, M_0) is a net structure $Q = (G, \Sigma, \Phi, \lambda, \varphi)$ with an initial marking M_0 where:

G is a PN structure, $\Sigma = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ is the input alphabet, and $\Phi = \{\phi_1, \phi_2,..., \phi_q\}$ is the output alphabet.

$\lambda : T \rightarrow \Sigma \cup \{\varepsilon\}$ is a labelling function of transitions, where ε represents a system internal event externally uncontrollable; it is not allowed that the symbol ε is associated to more than one $t_j \in p_i^{\bullet}$.

$\varphi : R(Q,M_0) \rightarrow (\mathbb{Z}^+)^q$ is an output function, that associates to each marking in $R(Q,M_0)$ a q-entry output vector; $q=|\Phi|$ is the number of outputs. φ is represented by a $q \times |P|$ matrix, such that if the output symbol ϕ_i is present (turned on) every time that $M(p_j) \geq 1$, then
$\varphi(i, j) = 1$, otherwise $\varphi(i, j) = 0$.

When an enabled transition t_j is fired in a marking M_k, then a new marking M_{k+1} is reached. This behaviour is represented as $M_k \xrightarrow{t_j} M_{k+1}$; the state equation is completed with the marking projection $y_k = \varphi M_k$, where $y_k \in (\mathbb{Z}^+)^q$ is the k-th output vector of the IPN.

Definition A.4. An IPN [David and Alla, 1994] (Q, M_0) is a net structure $Q = (G, V, \Sigma, \Phi, \lambda, \varphi)$ with an initial marking M_0 where:

G is a PN structure, $V = \{v_1, v_2, ..., v_r\}$ is the set of variables, $\Sigma = \{\alpha_1, \alpha_2, ..., \alpha_s\}$ is the set of events, and $\Phi = \{\phi_1, \phi_2, ..., \phi_q\}$ is the output alphabet.

$\lambda : T \rightarrow C \times E$ is a labelling function of transitions, where $C=\{C_1, C_2, \ldots\}$ is the set of variable conditions and $E=\{E_1, E_2, \ldots\}$ is the set of events.

In an *IPN*, a transition will be fired:

- *if* transition t_j is enabled
- and *if* condition $C(T_j)$ is true
- *when* event in $E(T_j)$ occurs

In this thesis, we consider a conjunction of events instead of a single event, due to the PLC technology explained in Chapter 2.

$\varphi : R(Q,M_0) \rightarrow (\mathbb{Z}^+)^q$ is an output function, that associates to each marking in $R(Q,M_0)$ a q-entry output vector; $q=|\Phi|$ is the number of outputs. φ is represented by a $q \times |P|$ matrix, such that if the output symbol ϕ_i is present (turned on) every time that $M(p_j) \geq 1$, then $\varphi(i, j) = 1$, otherwise $\varphi(i, j) = 0$.

The state equation is completed with the marking projection $Y_k = \varphi M_k$, where $Y_k \in (\mathbb{Z}^+)^q$ is the k-th output vector of the *IPN*.

Definition A.5. A place $p_i \in P$ is said to be observable if the i-th column vector of φ is not null, i.e. $\varphi(\bullet,i) \neq 0$. Otherwise it is non-observable. $P = P^m \cup P^u$ where P^m is the set of observable places and P^u is the set of non-observable places.

References

[Angluin, 1988] D. Angluin, "Queries and concept learning", *Machine Learning*, 2(4), pp. 319-342, 1988.

[Biermann and Feldman, 1972] A.W. Biermann, J.A. Feldman, "On the synthesis of finite-state machines from samples of their behaviour", *IEEE Trans. on Computers*, 21(6), pp. 592-597, 1972.

[Booth, 1967] T.L. Booth, "Sequential machines and automata theory", *John Wiley and Sons, Inc. New York, London, Sidney*, 1967.

[Cabasino, 2006a] M. P. Cabasino, A. Giua, C. Seatzu, "Identification of deterministic Petri nets", *Proc. of the 8th Int. Workshop on Discrete Event Systems*, Ann Arbor, Michigan, USA, Jul. 2006.

[Cabasino, 2006b] M. P. Cabasino, A. Giua, C. Seatzu, "Computational complexity analysis of a Petri net identification procedure", *Int. Symp. on Nonlinear Theory and its Applications*, Bologna, Italy, Sep. 2006.

[Cabasino, 2006c] M. P. Cabasino, A. Giua, C. Seatzy, "Identification of unbounded Petri nets from their coverability Graph", *Proc. of the 45th IEEE Conf. on Decision & Control*, San Diego, California, USA, Dec. 2006.

[Cabasino, 2009] M. P. Cabasino, "Diagnosis and identification of discrete event systems using Petri nets", *Ph. D. Thesis*, University of Cagliari, Mar. 2009.

[Cabasino, 2013] M. P. Cabasino, P. Darondeau, M. P. Fanti, C. Seatzu, "Model identification and synthesis of discrete-event systems", *Contemporary Issues in System Science and Engineering*, IEEE/Wiley Press Book Series, M. Zhou, H.-X. Lim M. Weijnen (Eds), 2013.

[Cook, 2004] J. E. Cook, Z. Du, C. Liu, A. Wolf, "Discovering models of behaviour for concurrent workflows", *Computers in Industry*, 53(3), pp. 297-319, 2004. Doi:10.1016/j.compind.2003.10.005.

[David and Alla, 1994] R. David, H. Alla, "Petri nets for modeling of dynamic systems- A survey", *Automatica*, 30(2), pp. 175-202, 1994.

[Dotoli, 2006a] M. Dotoli, M. P. Fanti, A. M. Mangini, "An optimization approach for identification of Petri nets", *Proc. of the 8th Int. Workshop on Discrete Event Systems*, Ann Arbor, Michigan, USA, Jul. 2006.

[Dotoli, 2006b] M. Dotoli, M. P. Fanti, A. M. Mangini, "On-line identification of discrete event systems: a case study", *Proc. of the 2006 IEEE Int. Conf. on Automation Science and Engineering*, Shangai, China, Oct. 2006.

[Dotoli, 2007] M. Dotoli, M. P. Fanti, A. M. Mangini, "On-line identification of discrete event systems via Petri nets: an Application to Monitor Specification", *Proc. of the 3rd Annual IEEE Conf. on Automation Science and Engineering*, Scottsdale, Arizona, USA, Sep. 2007.

[Dotoli, 2008] M. Dotoli, M. P. Fanti, A. M. Mangini, "Real time identification of discrete event systems using Petri nets", *Automatica*, 44(5), pp. 1209-1219, May 2008.

[Estrada, 2009] A. P. Estrada-Vargas, "Identification of concurrent discrete event systems from input-output sequences", *Master of Sciences Thesis*, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Unidad Guadalajara, Aug. 2009.

[Estrada, 2010a] A.P. Estrada-Vargas, E. López-Mellado, J.-J. Lesage. "A comparative analysis of recent identification approaches for discrete-event systems", *Mathematical Problems in Engineering*, 2010, Hindawi. Doi:10.1155/2010/453254

[Estrada, 2010b] A.P. Estrada-Vargas, E. Lopez-Mellado, J-J. Lesage. "An identification method for PLC-based automated discrete event systems". *Proc. of the IEEE Int. Conf. on Decision and Control*, pp.6740-6746, Atlanta, USA, Dec. 2010.

[Estrada, 2011] A. P. Estrada-Vargas, J.-J. Lesage, E. López-Mellado, "Stepwise identification of automated discrete manufacturing systems", *16th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, Toulouse, France, Sep. 2011.

[Estrada, 2012] A. P. Estrada-Vargas, E. Lopez-Mellado, J.-J. Lesage,. "Input-Output Identification of Controlled Discrete Manufacturing Systems", International Journal of Systems Science. Published on-line September 18, 2012, DOI:10.1080/00207721.2012.724098.

[Giua, 2005] A. Giua, C. Seatzu, "Identification of free-labeled Petri nets via integer programming", *Proc. of the 44th IEEE Conf. on Decision and Control, and the European Control Conference 2005*, Seville, Spain, Dec. 2005.

[Gold, 1967] E.M. Gold, "Language identification in the limit", *Information and Control*, 10, pp. 447-474, 1967.

[Hiraishi, 1992] K. Hiraishi, "Construction of safe Petri nets by presenting firing sequences", *Lectures Notes in Computer Sciences*, 616, pp. 244-262, 1992.

[Ishizaka, 1990] H. Ishizaka, "Polynomial time learnability of simple deterministic languages", *Machine Learning*, 5, pp. 151-164, 1990.

[Kella, 1971] J. Kella, "Sequential machine identification", *IEEE Trans. on Computers*, 20, pp. 332-338, 1971.

[Klein, 2005a] S. Klein, L. Litz, J.-J. Lesage, "Fault detection of discrete event systems using an identification approach", *16th IFAC World Congress*, CDROM paper n°02643, 6 pages, Praha, Czech Republic, Jul. 2005.

[Klein, 2005b] S. Klein, "Identification of discrete event systems for fault detection purposes", *Ph. D. Thesis*, Ecole Normale Supérieure de Chachan, Oct. 2005.

[Lampérière-Couffin, 1999] S. Lampérière-Couffin, O. Rossi, J.-M. Roussel, J.-J. Lesage, "Formal Validation of PLC programs : a survey", *European Control Conference*, Karlsruhe, Germany, CD-ROM paper n°741, 6 pages, Sep. 1999

[Levy, 1978] L.S. Levy, A.K. Joshi, "Skeletal structural descriptions", *Information and Control*, 39, pp. 192-211, 1978.

[Meda, 1998] M.E. Meda, "DES identification using interpreted Petri nets", *Int. Symposium on Robotics and Automation*, pp. 353-357, Saltillo, Mexico, Dec. 1998.

[Meda, 2000a] M. Meda, A. Ramírez, E. López "Asymptotic identification for DES", *IEEE Conf. on Decision and Control*, Sydney, Australia, pp. 2266-2271, Dec. 2000.

[Meda, 2000b] M. Meda, A. Ramírez, E. López, "Behavioral properties for the control of discrete event systems modelled by interpreted Petri nets", *IEEE Int. Conf. on Systems, Man, and Cybernetics*, Nashville, USA, pp. 2150-2155, Oct. 2000.

[Meda, 2001] M. Meda, E. López, "A passive method for on-line identification of discrete event systems", *IEEE Int. Conf. on Decision and Control*, Orlando, Florida, USA pp. 4990-4995, Dec. 2001.

[Meda, 2002a] M. E. Meda, "On-line identification of discrete event systems: fundamentals and algorithms for the synthesis of Petri net models", *Ph. D. Thesis*, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Unidad Guadalajara, Nov. 2002.

[Meda, 2002b] M. Meda, E. López, "Incremental synthesis of Petri net models for identification of discrete event systems", *IEEE Conf. on Decision and Control*, Las Vegas, USA, pp.805-810, Dec. 2002.

[Meda, 2003] M. Meda, E. López, "Required transition sequences for DES identification", *IEEE Conf. on Decision and Control*, Maui, Hawaii USA pp. 3778-3782, Dec. 2003.

[Meda, 2005] M. Meda, E. López, "Identification of concurrent discrete event systems using Petri nets", *IMACS 2005 World Congress*, Paris, France, pp. 1-7, Jul. 2005.

[Ould El Medhi, 2006] S. Ould El Medhi, E. Leclercq, D. Lefebvre, **"**Petri nets design and identification for the diagnosis of discrete event systems", *2006 IAR Annual Meeting*, Nancy, France, Nov. 2006.

[Ould El Mehdi, 2012] S. Ould El Mehdi, R. Bekrar, N. Messai, E. Leclercq, D. Lefebvre, B. Riera, "Design and identification of stochastic and deterministic stochastic Petri nets", *IEEE Trans. on Systems, Man and Cybernetics, Part A*, 42(4), pp. 931-946.

[Richetin, 1984] M. Richetin, M. Naranjo, P. Luneau, "Identification of automata by sequential learning", *Pattern Recognition Letters 2*, 2(6), pp. 379-385, 1984.

[Roth, 2010a] M. Roth, "Identification and fault diagnosis of industrial closed-loop discrete event systems", *Ph. D. Thesis*, Technische Universität Kaiserslautern, Ecole Normale Supérieur de Cachan, Oct. 2010.

[Roth, 2010b] M. Roth, J.-J. Lesage, L. Litz, "Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems", *Proc. of the 2010 American Control Conf.*, Baltimore, USA, pp. 2601-2606, Jun.-Jul. 2010.

[Roth, 2010c] M. Roth, J.-J. Lesage, L. Litz, "Identification of discrete event systems, implementation issues and model completeness", 7th Int. Conf. on Informatics in Control Automation and Robotics, Funchal, Portugal, pp. 73-80, Jun. 2010

[Roth, 2011] M. Roth, J.-J. Lesage, L. Litz, "The concept of residuals for fault localization in discrete event systems", *Control Engineering Practice*, 19(9), pp. 978-988.

[Roth, 2012] M. Roth, S. Schneider, J.-J. Lesage, L. Litz, “Fault detection and isolation in manufacturing systems with an identified discrete event model”, *Int. Journal of Systems Science*, 43(10), pp. 1826-1841. Doi**:**10.1080/00207721.2011.649369

[Takada, 1998] Y. Takada, “Grammatical inference for even linear languages based on control sets”, *Information Processing Letters*, 28, pp. 193-199, 1998.

[Valiant, 1984] L.G. Valiant, “A theory of the learnable”, *Communications of the ACM*, 27, pp. 1134-1142, 1984.

[van der Aalst, 2004] W. van der Aalst, T. Weijters, L. Maruster, “Workflow mining: discovering process models from event logs”, *IEEE Trans. on Knowledge and Data Engineering*, 16(9), Sep. 2004.

[Veelenturf, 1978] L.P.J. Veelenturf, “Inference of sequential machines from sample computations”, *IEEE Trans. on Computers*, 27, pp. 167-170, 1978.

[Veelenturf, 1981] L.P.J. Veelenturf, “An automata theoretical approach to developing learning neural networks” *Cybernetics and Systems*, 12, pp. 179-202, 1981.

Identificación Caja-negra de Sistemas de Eventos Discretos Automatizados

Resumen

Esta tesis trata sobre la identificación de sistemas de eventos discretos (SED) automatizados operando en un contexto industrial. En particular el trabajo se enfoca a los sistemas formados por una planta y un controlador lógico programable (CLP) operando en lazo cerrado; la identificación consiste en la obtención de un modelo aproximado expresado en redes de Petri interpretadas (RPI) a partir del comportamiento externo observado en la forma de una única secuencia de vectores entrada-salida del CLP.

En primer lugar, se presenta una revisión de algunos métodos de identificación así como un estudio comparativo de enfoques recientes para la identificación de SED. Enseguida se describe el problema abordado; se detallan características tecnológicas importantes en los sistemas automatizados por CLP. Dichas características deben ser consideradas en la solución del problema, pero no pueden ser tratadas por métodos de identificación anteriores. La principal contribución de esta tesis es la creación de dos métodos de identificación complementarios.

El primer método permite construir de manera sistemática un modelo RPI desde una única secuencia de entrada-salida que representa el comportamiento observable del SED. Los modelos RPI resultantes describen en detalle la evolución de entradas y salidas durante el funcionamiento del sistema.

El segundo método considera DES grandes y complejos; está basado en un enfoque estadístico para crear modelos RPI compactos y expresivos. Consiste en dos etapas; la primera calcula, a partir de la secuencia entrada-salida, la parte reactiva del modelo, la cual está compuesta por lugares observables y transiciones. La segunda etapa construye la parte no observable del modelo incluyendo lugares que aseguran la reproducción de la secuencia de entrada-salida.

Los métodos propuestos, basados en algoritmos de complejidad polinomial, han sido implementados en herramientas software, las cuales han sido probadas con secuencias de entrada-salida obtenidas de sistemas reales en funcionamiento. Las herramientas son descritas y su aplicación es ilustrada mediante dos casos de estudio.

Black-box Identification of Automated Discrete Event Systems

Abstract

This thesis deals with the identification of automated discrete event systems (DES) operating in an industrial context. In particular the work focuses on the systems composed by a plant and a programmable logic controller (PLC) operating in a closed loop; the identification consists in obtaining an approximate model expressed in interpreted Petri nets (IPN) from the observed behaviour given under the form of a single sequence of input-output vectors of the PLC.

First, an overview of previous works on identification of DES is presented as well as a comparative study of the main recent approaches on the matter. Then the addressed problem is stated; important technological characteristics of automated systems and PLC are detailed. Such characteristics must be considered in solving the identification problem, but they cannot be handled by previous identification techniques. The main contribution in this thesis is the creation of two complementary identification methods.

The first method allows constructing systematically an IPN model from a single input-output sequence representing the observable behaviour of the DES. The obtained IPN models describe in detail the evolution of inputs and outputs during the system operation.

The second method has been conceived for addressing large and complex industrial DES; it is based on a statistical approach yielding compact and expressive IPN models. It consists of two stages; the first one obtains, from the input-output sequence, the reactive part of the model composed by observable places and transitions. The second stage builds the non-observable part of the model including places that ensure the reproduction of the observed input-output sequence.

The proposed methods, based on polynomial-time algorithms, have been implemented in software tools, which have been tested with input-output sequences obtained from real systems in operation. The tools are described and their application is illustrated through two case studies.

Identification Boîte-noire de Systèmes Automatisés à Evénements Discrets

Résumé

Cette thèse traite de l'identification des systèmes à événements discrets (SED) automatisés dans un contexte industriel. En particulier, ce travail s'intéresse aux systèmes constitués d'un processus et d'un contrôleur (le plus souvent un automate programmable industriel - API) fonctionnant en boucle fermée. Dans ce contexte, l'identification a pour objectif de construire un modèle (exprimé sous la forme d'un réseau de Petri interprété - RPI) approximant le comportement du SED, à partir de l'observation de son comportement externe, capturé sous la forme d'une séquence de vecteurs d'entrées-sorties de l'API.

Tout d'abord, une analyse des méthodes d'identification existantes est présentée, ainsi qu'une étude comparative des approches les plus récentes dédiées à l'identification des SED. Le problème de l'identification est ensuite reformulé sous son aspect expérimental, de manière à permettre une prise en compte des caractéristiques technologiques des systèmes automatisés, et en particulier de celles de l'API. Il est alors montré que les méthodes existantes d'identification ne peuvent prendre en compte ces contraintes expérimentales. La contribution principale de cette thèse, qui consiste à proposer deux méthodes d'identification complémentaires, est ensuite développée.

La première méthode permet de construire systématiquement un modèle RPI à partir d'une seule séquence d'entrées-sorties représentant le comportement observé du SED. Les modèles RPI construits décrivent en détail l'évolution des entrées et sorties pendant le fonctionnement du système.

La seconde méthode est plus spécifiquement destinée aux SED complexes et de grande taille ; elle est basée sur une approche statistique qui permet la construction de modèles RPI compacts et expressifs. Elle est composée de deux étapes ; la première permet de calculer à partir de la séquence d'entrées-sorties, la partie réactive observable du modèle. La deuxième étape permet l'inférence du comportement non-observable, en ajoutant des places qui permettent la reproduction de la séquence entrée-sortie observée.

Les méthodes proposées, basées sur des algorithmes de complexité polynomiale, ont été implémentées dans un environnement logiciel expérimental et ont été testés avec des séquences d'entrées-sorties obtenues à partir de systèmes réels en fonctionnement. L'ensemble de ces outils expérimentaux est décrit et leur application est illustrée à travers deux études de cas.