HAL
open science

Blackbox Behavioural Identification of Discrete Event Systems by Interpreted Petri Nets

Jérémie Saives

To cite this version:

Jérémie Saives. Blackbox Behavioural Identification of Discrete Event Systems by Interpreted Petri Nets. Automatic. Université Paris-Saclay, 2016. English. NNT : 2016SACLN018 . tel-01358280v1

HAL Id: tel-01358280

https://theses.hal.science/tel-01358280v1

Submitted on 31 Aug 2016 (v1), last revised 1 Sep 2016 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

NNT : 2016SACLN018

THESE DE DOCTORAT
DE L'UNIVERSITE PARIS-SACLAY,
préparée à l'ENS Cachan

ÉCOLE DOCTORALE N° 580
Sciences et Technologies de l'Information et de la Communication

Spécialité *Electronique, Electrotechnique, Automatique*

Par

Monsieur Jérémie SAIVES

Blackbox Behavioural Identification of Discrete Event Systems by Interpreted Petri Nets

Identification Comportementale "Boîte-noire" des Systèmes à Evénements Discrets par Réseaux de Petri Interprétés

Thèse présentée et soutenue à *Cachan*, le *30 juin 2016*.

Composition du Jury :

M. Basile Francesco	Professeur, Univ. di Salerno, Italy	Rapporteur
M. Lopez Luis Ernesto	Professeur, Cinvestav Guadalajara, Mexico	Rapporteur
M. Alla Hassane	Professeur, GIPSA-Lab, France	Président
M. Riera Bernard	Professeur, URCA-CReSTIC, France	Examinateur
M. Lesage Jean-Jacques	Professeur, ENS Cachan – LURPA, France	Directeur de thèse
M. Faraut Gregory	MdC, ENS Cachan – LURPA, France	Co-encadrant

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

"The world is full of obvious things which nobody by any chance ever observes."

-Sherlock Holmes-

Sir Arthur Conan Doyle, *The Hound of the Baskervilles*

Acknowledgments

During these last three years, I was a member of the world of research, developping contributions and pushing further away the limits of science. When I came to ENS Cachan seven years ago, I had teaching as sole purpose; I hope now that I can combine teaching and research in my future career.

Retrospectively, this thesis is an experience to be proud of, and I am grateful to the people who made it possible. Notably, I want to thank first Jean-Jacques Lesage, who took me under his wing in the last years. His teachings and advice helped me become a better scientist. To complete the team, Gregory Faraut was always available to cheer me up, discuss ideas and help me proof test some of them. I wish him an accomplished scientific career. I would also like to thank Jean-Marc Roussel for bringing me to the field of discrete automatic, being actually the first milestone who led to this thesis.

Joyful was the ending, and I would like to thank my jury for reviewing this work. Francesco Basile came all the way from Italia, and I thank him for his enthusiasm regarding DES identification. Ernesto Lopez-Mellado provided helpful advice for the formalization, and I thank him also for inviting me to Mexico two years ago. Finally, Hassane Alla and Bernard Riera showed as well a lot of interest, and I thank them for their sympathy.

Escaping the routine was made possible by colleagues, PhD and Master students, friends and geeks alike. I thank all the LURPA for the environment, and the good mood.

Resting now on the shoulders of my coffee-mate Laureen, the CIVIL is a wonderful committee which makes everyone feel integrated, and the hard days easier. I'm happy to have directed it, see it now in good hands, and thank all its members, past and present.

Especially Julien, my dear neighbour from Office 18, who supports my rantings and my cats. Lorène and Fabien, who arrived with me, and with whom I shared the difficulty of writing. But also Matthias, Kevin, Sylvain and all the younglings.

My roommate, Blandine, who shared a sinusoidal mood with me during this last year. My former roomate, Benjamin, who came from far away to see my defence.

It is said that being a teacher in a classroom is like being an actor on stage. I thank the LiKa, my improv team, among which I developped acting skills and gained confidence. To Marc, Simon, Pat, Mathilde, and a lot of others I might forget.

Ending these acknowledgements could only be done by thanking my parents for their continuous support, and I am afraid there is not enough space here to be exhaustive.

Cachan, July 2016

Contents

Table of contents iii

List of figures vii

List of tables xiii

Introduction 1

1 Identification of Discrete Event Systems 5

1.1 Background on Discrete Event Systems . . . 5

1.1.1 Definition . . . 5

1.1.2 Reactive DES and event generators . . . 7

1.1.3 Formalism: Classical models . . . 8

1.2 Identification of a DES: Problem Statement . . . 14

1.2.1 Systems of Interest . . . 15

1.2.2 Incompleteness of the observation . . . 19

1.2.3 Problem Statement . . . 21

1.2.4 Identification for reverse engineering . . . 21

1.3 Identification in the literature . . . 22

1.3.1 Origin: early computer science approaches . . . 22

1.3.2 Identification by Automata . . . 23

1.3.3 Identification by Petri Nets . . . 27

1.4 Conclusions and positioning . . . 39

2 Blackbox behavioural identification of a reactive automated system 41

2.1 Two behaviours: Observable and Unobservable . . . 41

2.1.1 Event types . . . 41

2.1.2 Framework of the method . . . 43

2.1.3 Illustrative example: Sorting system . . . 44

2.2 Identification of the observable behaviour . . . 45

2.2.1 Building output firing functions . . . 45

2.2.2 Construction of the transitions and observable places . . . 48

2.2.3 Determination of the firing sequence . . . 51

2.3 Inference of the unobservable behaviour . . . 51

2.3.1 Finding Causal and Concurrent Transitions . . . 52

2.3.2 Computing unobservable places 54
2.3.3 Verification of the net 56
2.4 Discussion and proposed improvements 56
2.4.1 Scalability and concurrency 57
2.4.2 Limits of the unobservable behaviour discovery 58

3 Scalability of the observable behaviour construction 61
3.1 Illustrative system: the MSS 61
3.1.1 Presentation 61
3.1.2 Data collection 63
3.2 Resynchronization of asynchronous events in concurrent systems and consequences 64
3.3 Filtering of the causality matrix 67
3.3.1 Design of the filter 67
3.3.2 Application 71
3.4 Transitions reduction 73
3.4.1 Replacement of spurious transitions 73
3.4.2 Application 79
3.5 Conclusions 84

4 Discovery of the unobservable behaviour 85
4.1 Problem statement 85
4.2 Theoretical background 86
4.2.1 From the firing sequence to admissible places 86
4.2.2 Complexity of finding admissible places 90
4.2.3 Intermediate conclusions 93
4.3 Assessing the quality of a net 94
4.3.1 Quality metrics 95
4.3.2 Importance of understandability 97
4.4 Discovery in practice 100
4.4.1 Partitioning of the search space 100
4.4.2 Exploration strategy 103
4.4.3 Stopping criterion 104
4.4.4 Algorithmic application 105
4.5 Extensions 106
4.5.1 Implicit places and consequences 107
4.5.2 Reduction of the algorithmic cost 109
4.6 Practical examples 112
4.6.1 Unobservable behaviour only 112
4.6.2 Complete approach 114
4.7 Discussion 118

4.8 Conclusion 120

5 Automated partitioning for distributed identification 123

5.1 Statement of the partitioning problem 123

5.1.1 Objective of the partitioning 123

5.1.2 Related work 127

5.1.3 Mapping I/Os and observable fragments 130

5.1.4 Final formulation 134

5.2 Partitioning by agglomerative hierarchical clustering 138

5.2.1 Similarity and affinity of subsystems 138

5.2.2 Limited clustering 142

5.2.3 Results and interpretation 144

5.3 Conclusion 149

Conclusion and outlooks 153

Bibliography 157

A Assessing the quality of an identified Petri net 173

A.1 Precision 173

A.2 Complexity metrics 175

A.2.1 Structural Complexity 176

A.2.2 Dynamic Complexity 178

B Proofs 181

List of Figures

1.1 Overview of a closed-loop logical system, with its inputs $\mathbb{U}$, its outputs $\mathbb{Y}$, and its state variables $\mathbb{X}$ 6
1.2 [Cassandras and Lafortune, 2008] An example of a state trajectory of a DES. 7
1.3 A spontaneous event generator (a). A closed-loop system consisting in two reactive systems, interacting through their I/Os (b) 8
1.4 A closed-loop system consisting in two reactive systems, seen as an event generator 8
1.5 An example of a DFA 10
1.6 An example of a PN 12
1.7 An example of an IPN 14
1.8 The cyclic behaviour of a Programmable Logic Controller, including the passive observation 16
1.9 An observation of the process controlled illustrating the functioning of the PLC 17
1.10 An illustration of the synchronization effect. 18
1.11 An illustration of a delayed output event 19
1.12 [Roth et al., 2009a] Evolution of the number of observed words of length n over production cycles h 20
1.13 [Roth, 2010] Observed languages over a hundred production cycles 20
1.14 [Prähofer et al., 2014] Left: A flow chart of a function block of the PLC, with the paths numbered. Right: A trace and its representation as a FSM 24
1.15 [Klein, 2005](a) The identified NDAAO after the treatment of the sequences; (b) The simplified model after merging of equivalent states . . . 25
1.16 From [Hiraishi, 1992]: (a)The finite acceptor and (b) the PN deduced from it. 27
1.17 Illustration of a region, and its elementary net equivalent 29
1.18 [Giua and Seatzu, 2005] Two free-labelled, generalized nets identified from solving ILP 30
1.19 [Cabasino et al., 2014](a)Nominal net; (b) Faulty net with two silent transitions 31
1.20 The matrix B and the Petri net computed from Seq 32

1.21 [Van der Aalst, 2013a] Positioning of the three main types of process mining: *discovery, conformance and enhancement* 33
1.22 [Van der Aalst et al., 2004] The net mined by the α-algorithm from the log $\{ABCD, ACBD, AED\}$ 35
1.23 [Meda-Campana and Lopez-Mellado, 2001] 6 nets incrementally computed from the observed output sequences 36
1.24 [Estrada-Vargas et al., 2014](a) Basic identified model; (b) Model after merging; (c) Model after concurrency simplification; (d) IPN model with observable places 38
1.25 Left: Unobservable part (machine behaviour). Right: Observable part (signals) 39

2.1 Data collection, then construction of an IPN in two steps 44
2.2 Illustrative example: a package sorting system 44
2.3 Matrices computed for the sorting system. Left: Direct Causality Matrix, Right: Indirect Context Matrix 47
2.4 Elementary observable fragments constructed from the firing functions . 49
2.5 Example of the creation of a new transition, by choosing the right input conditions 49
2.6 Example of the creation of a new transition, based on a simultaneous output events observation 50
2.7 Result of the first step for the sorting system: the observable part 50
2.8 Structures that represent $t_a < t_b$: a) shows a causal relationship whereas b) shows a concurrent relationship 52
2.9 a)Choice and b)parallelism after the firing of t_k 55
2.10 The non-observable part of the IPN computed for the sorting system . . 55
2.11 (a) The unobservable part, corrected by the token-flow equation; (b) The final IPN with both interpretation layers 57
2.12 (a) The observable part of the net; (b) The net identified ; (c) A net that satisfies the problem 59
2.13 A non free choice net with two memory places 59

3.1 A picture of the MSS in the LURPA, and of the workpieces (gears and bearings) processed. 62
3.2 Scheme of the MSS, decomposed in 4 stations and 11 subsystems 62
3.3 Observed language of the MSS over 20 production cycles, for $n = 1, 2$ . . 63
3.4 Two concurrent processes observed asynchronously 64
3.5 Two concurrent processes spuriously observed synchronously 65
3.6 (a) Observable part identified for two concurrent processes with spurious transitions (b) Simplified desirable model 66
3.7 MSS transition occurences 66
3.8 Principle of the filter: put zeros in cells corresponding to noisy observations 67

3.9 Result of the Filter on the DCM of the MSS. Grey cells correspond to validated non-empty cells, and black cells to noise. 71
3.10 Efficacity of the filter: better denoising for column sums close to one. . . 72
3.11 Principle of the reduction: remove transitions and replace in S 73
3.12 Increase of the permissivity of transitions by OEFF labelling 74
3.13 Example of an observable fragment, with t_1 reducible by $[t_4, t_5]$. 75
3.14 An observable fragment; only the unrelated input events are shown in the firing functions 76
3.15 Non-unicity of minimal reductions illustrated by t_5 78
3.16 Observable Behaviour of the MSS pre-reduction: one single spaghetti fragment 82
3.17 The result of the reduction for the MSS: 30 observable places and 101 transitions. 13 observable fragments and 24 isolated transitions 83
3.18 Observed language of the MSS expressed on the transitions, for $n = 1, 2, 3$ 84

4.1 An unobservable PN built from S using mutual dependencies. 88
4.2 A PN structure composed of two places p_{ij} and p_{ji} for $\Sigma_i = \{t_i^1, \ldots, t_i^m\}$, $\Sigma_j = \{t_j^1, \ldots, t_j^n\}$. t_i^1 is the first transition fired. 89
4.3 Two net solutions, built with different sets of admissible places 91
4.4 Visualisation of the complexity of the problem 94
4.5 A system consisting of two chariots and a gripper 97
4.6 Observable part of the system. 5 outputs and 9 observable transitions . . 98
4.7 (a) A simple solution, with a lot of exceeding language ; (b) A complex solution, with no exceeding language up to $n = 6$ 99
4.8 Visual representation of the partition of $(2^T - \{\emptyset\})^2_{\not\emptyset}$. 102
4.9 Size of the cell $\mathbb{D}_\delta$ of the search space depending on δ, plotted for different values of $|T|$. Y scale is logarithmic. 104
4.10 The chariots example: (a)Observable part; (b)Initial net for the exploration; (c)Net identified at the end of the limited exploration; (d)Net identified by a full exploration 106
4.11 An example of a net (a), after deletion of implicit places (b) 107
4.12 Results of the identification of Example 4.7 108
4.13 Vertical propagation of the domination relationship 111
4.14 Example 4.9: (a) Net discovered for $\mathbb{D}_2$; (b) Net discovered for $\mathbb{D}_3$ 113
4.15 Example 4.10: the initial marking is unreachable after any firing. 113
4.16 Example 4.11; there are two backward loops in the main process t0 $\longrightarrow$ t1 114
4.17 Example 4.12: 5 fully concurrent processes, synchronized by m 114
4.18 Example 4.13: (a)Observable part; (b)Solution $\mathbb{D}_2$; (c)Solution $\mathbb{D}_4$ 115
4.19 Extended chariots example, with its observable behaviour 115
4.20 The resulting net after the exploration of $\mathbb{D}_2$. 116
4.21 A representation of the counter of the second conveyor, using self-loops . 117

4.22 Monolithic IPN model obtained after the exploration of $\delta = 2$. 5 transitions and two fragments remain unconnected to the remainder of the net . . . 121

5.1 Principle of the distributed approach, based on a partition of the system. 124
5.2 Identified net for SUB_k consisting in either 1 output or 1 input . . . 126
5.3 Shape of the Pareto frontier of optimal solutions of the cover problem . . 126
5.4 (a) The NDAAO and (b) the IPN built from $SUB_k = \{u_1, Y_1\}$ . . . 128
5.5 Overview of the partitioning approach from [Schneider, 2014] . . . 129
5.6 Mapping of an observable fragment and isolated transitions on the different I/O sets . . . 131
5.7 Sharing a connected input between two fragments and two isolated transitions. . . . 132
5.8 Possible unobservable behaviour for a Type 2 Block . . . 134
5.9 Location of the blocks on the MSS . . . 134
5.10 Overview of the distributed approach, partitioning taking place after the construction of the observable model . . . 135
5.11 The 21 blocks computed for the MSS . . . 137
5.12 Similarity table and Affinity graph deduced for Example 5.4 . . . 139
5.13 (a) Evolution of the affinity graph along the clustering; (b) Hierarchical representation . . . 141
5.14 Successive affinity graphs, first run of the clustering, with $|T|_{Lim} = 9$ . . 145
5.15 Successive affinity graphs, second run of the clustering, with $|T|_{Lim} = 18$ 146
5.16 Location of the six computed subsystems on the MSS, $|T|_{Lim} = 18$ . . . 146
5.17 Location of the six computed subsystems on the MSS, $t_{Lim} = 20s$ . . . 147
5.18 Succesive affinity graphs computed for the MSS, $t_{Lim} = 20s$ . . . 148
5.19 Evaluation of different partitions computed with the clustering approach 149
5.20 IPN identified for $SSYS_1$, biggest subsystem of the MSS . . . 150
5.21 IPNs identified for the remaining subsystems of the MSS . . . 151
Fr.1 Principe du filtre : Annuler les cases de la matrice correspondant à des observations parasites . . . 169
Fr.2 Principe de la réduction : Suppresion des transitions, et remplacement dans S . . . 169
Fr.3 Motif projeté caractérisant une mutuelle dépendance, et places associées . 170
Fr.4 Deux RdPI identifiés pour le même système : (a) est simple, mais génère plus de langage excédentaire que (b) . . . 170
Fr.5 Exemple de l'approche par clustering : Evolution d'un graphe d'affinité (a), et construction de la hiérarchie (b) . . . 171
Fr.6 Localisation des six sous-systèmes obtenus par time-clustering sur la MSS, $t_{Lim} = 20s$ . . . 172

A.1 First row: N identified for S, and its exceeding language; Second row: N' identified for S', and its exceeding language 174
A.2 The net N identified and its exceeding language according to Definition A.2 175
A.3 (a) A simple solution, with a lot of exceeding language ; (b) A complex solution, with no exceeding language up to $n = 6$ 176
A.4 Reachability graphs of the nets of Figure A.3 180

List of Tables

1.1 Similarity between closed-loop systems and IPN 40

2.1 Elementary description of the first event vectors 45
2.2 Firing functions computed for the sorting system 48
2.3 Translation into the firing sequence of the first vectors of E 51

3.1 DCM of two concurrent processes 65
3.2 Non empty cells of the column $\mathrm{DCM}_{i,1Y04_0}$ 67

4.1 Evaluation of the precision and simplicity metrics on the discovered nets for the chariots. 117
4.2 Evaluation of the simplicity metrics on the discovered nets for the MSS. . 118

A.1 Comparison of structural complexity metrics on Example 4.5 178
A.2 Comparison of dynamic complexity metrics on Example 4.5 180

Introduction

System modelling has become a preponderant task in engineering, namely in the field of automatic control, where models are extensively used to find optimal control laws. Identification consists in building a mathematical model of a system from its observation; namely, parametric identification is widely spread to build models of continuous-time systems described by differential equations. However, Discrete Event Systems (DES), described by state-based models whose evolutions are triggered by events, are often modelled from expert knowledge only. DES identification has received more and more interest over the last decade. The objective of this thesis is to contribute to its development, in order to make identification a viable alternative to expert modelling.

Identification is an experimental approach by nature, requiring to observe a real system during its operation. The experimental setup implies physical and technological constraints, which must be taken into account by identification algorithms. Namely, the nature of the input/outputs, the cyclicity of the controller or the impossibility to achieve complete observation are challenges to be accepted. Another challenge is to develop scalable algorithms, in order to deal with systems of realistic sizes, despite the state-space explosion issue quite usual for concurrent DES.

Models are designed for a purpose, be it either simulation, performance evaluation, control, diagnosis, dysfunctional analysis... Identified models should be designed likewise, and identification algorithms concieved for a given purpose. Model-based fault diagnosis has been the key motivation for a series of theses ([Klein, 2005][Roth, 2010][Schneider, 2014]). The resulting models, automata, are especially efficient for diagnosis. However, they lack the semantics to offer an explicit vision of complex behaviours such as concurrency, or input/output causalities. This thesis is thus dedicated to reverse-engineering. The aim is to produce compact models, easily readable, in order for an engineer to understand the operations performed by the system.

The focus of this thesis is set on closed-loop systems consisting in a plant and a controller. Sensors and actuators of the plant are respectively the logical inputs and outputs of the controller, typically a Programmable Logic Controller (PLC). These controllers are widely used in manufacturing systems for their robustness to hostile environments, and their reactivity to sensor events. In this thesis, the identification is blackbox, *i.e.* no knowledge about the internal variables of the system is available, and passive, *i.e.* the observed data consists only in the values of the inputs/outputs, recorded during the operation of the system.

The behaviour of the system can be split into an *observable* part, observed through

the direct causal input/output evolutions of the DES, and an *unobservable* part, in which are agregated memory effects, timed behaviours or sequential evolutions of the internal state variables of the system. Interpreted Petri Nets (IPN) offer the semantics to visualize both parts in a single graphical model. For reverse engineering, these models should be compact and as simple as possible.

To achieve this goal, an approach in two steps was proposed in a previous thesis [Estrada-Vargas, 2013]. The observable, then unobservable parts of the model are built successively. The first step is performing, building fragments expressing the input/output causalities. These fragments are then connected in the second step by adding the unobservable part, leading to a monolithic model. The method was succesfully applied to small systems, but several difficulties appeared when bigger, concurrent systems are considered. Direct causalities on one hand, and concurrency in the unobservable behaviour on the other hand, become harder to discover.

The contributions of this thesis aim therefore at improving the scalability of the approach, and are the following:

- Improve the monolithic approach by:
  - Limiting the effects of concurrency in the construction of the observable part.
  - Developping a new approach for the discovery of the unobservable part.
- Propose a distributed approach, to split the system into subsystems when monolithic models become too big to be apprehended or computed.

This document is organized as follows:

Chapter 1 presents the formalisms used through the thesis, namely IPNs, the systems considered, and the problems and challenges related to DES identification. A review of the state of the art is proposed, from the first techniques of computer science to the DES-related works of the last years.

Chapter 2 resumes the approach in two steps developed in [Estrada-Vargas, 2013], in which the remainder of the thesis takes its roots. The construction of the observable behaviour, based on a probabilistic framework, and the discovery of the unobservable behaviour, based on numerous rules, are successively exposed and illustrated. The new challenges offered by concurrency to both steps are exposed; these challenges are taken up by the contributions presented in this thesis.

Chapter 3 develops the improvement of the observable step towards more scalability. An absorbing filter is designed to get rid of spurious correlations between inputs and outputs, observed due to the concurrency of the system coupled with the synchronization of the controller, and falsely interpreted as causalities. Irrelevant transitions created by the same phenomenon are filtered and reduced as well. A real system available at the LURPA is introduced in this chapter. Its reasonable size (73 I/Os) makes it a good illustration of the scalability improvements presented throughout in this thesis.

Chapter 4 presents the new, generic approach developped to enhance the unobservable discovery. A single theorem is required to characterize all unobservable places of the identified model, and ensure its fitness, as the observed behaviour is fully reproduced. Due to state-space explosion, a heuristic is proposed to perform the discovery in practice. This heuristic is designed to reduce the structural complexity of the resulting model, desirable quality of a model dedicated to reverse-engineering.

Finally, Chapter 5 proposes a distributed approach, to be employed when monolithic models become too big to be computed, and hard to understand on top of it. The distribution occurs after the computation of the observable behaviour; the observable fragments are used as an initial partitioning of the system, and a hierarchical clustering algorithm is proposed. The fragments are merged while ensuring that the resulting subsystems are satisfyingly modelled, until a computation time threshold is reached. The result is a set of distributed models, easier to compute, and easier to read due to their reduced size, thus fitting the objective.

Chapter 1

Identification of Discrete Event Systems

Introduction

The research presented in this thesis deals with Identification of Discrete Event Systems, *i.e.* the construction of mathematical models from observed data, collected during the operation of the real system. The aim of this research is more specifically to provide compact models exhibiting the reactive behaviour of the observed system.

First, Discrete Event Systems are defined in Section 1.1, and the modelling formalisms are recalled. Then, the systems of interest, with their technological constraints, are presented, leading to the statement of the problem dealt with in this thesis (Section 1.2). Numerous works dealing with Discret Event Systems Identification are then reviewed in Section 1.3. Finally, Section 1.4 concludes on the choice of model adapted to the objective, and presents the contributions of this thesis.

1.1 Background on Discrete Event Systems

1.1.1 Definition

First, an informal description of the behaviour of systems is given. Generically speaking, a dynamical system can be associated to *input variables* $\mathbb{U} = \{u_1, u_2, \dots, u_{|\mathbb{U}|}\}$ and *output variables* $\mathbb{Y} = \{y_1, y_2, \dots, y_{|\mathbb{Y}|}\}$. Input variables are stimuli that provoke a response through the output variables.

The output variables might depend only on the values of the input variables, *i.e.* $\forall j, y_j = f(u_1, \dots, u_{|\mathbb{U}|})$). Namely, continuous-time systems are often described by ordinary differential equations ($y = f(x, x', x'', \dots)$. Most systems can not be modelled so simply. Pushing a light switch might turn a lightbulb on, or off, depending on the previous state of the lightbulb. Such systems possess therefore one, or multiple *state variables*, $\mathbb{X} = \{x_1, x_2, \dots, x_{|\mathbb{X}|}\}$, that describe the system at a given moment. For instance, the lightbulb can be described by a logical variable x_1 which is 1 (0) when the light is on (off). Figure 1.1 gives a generic overview of a system consisting in a plant

and a controller, where the value of the outputs $\mathbb{Y}$ depends also on the state variables $\mathbb{X}$. The evolution of the system is now state-dependant: for the same input, the outcome depends on the state. The state of the system is also updated depending on the input, which leads to the two following equations valid for any system:

$$\begin{cases} \forall j, y_j = f(u_1, \dots, u_{|\mathbb{U}|}, x_1, \dots, x_{|\mathbb{X}|}) \\ \forall k, \hat{x_k} = g(x_1, \dots, x_{|\mathbb{X}|}, u_1, \dots, u_{|\mathbb{U}|}) \end{cases}$$

where $\hat{x_k}$ represents the future of the state variable x_k. For instance, if x_k is a continuous time-dependant variable, $\hat{x_k}$ is its derivative. The second equation describes the *state evolution* of the system.

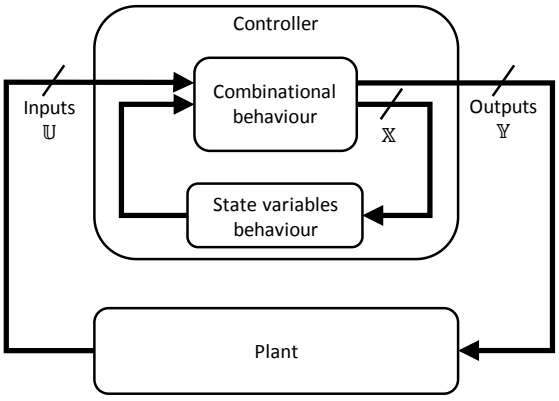


Figure 1.1: Overview of a closed-loop logical system, with its inputs $\mathbb{U}$, its outputs $\mathbb{Y}$, and its state variables $\mathbb{X}$

The interest is now set on a specific class of systems implying state variables: Discrete Event Systems. The definition of this class of systems follows:

Definition 1.1 (Discrete Event System [Cassandras and Lafortune, 2008]). *A Discrete Event System (DES) is a discrete-state, event-driven system; that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time.*

The system is described by a set of states $X = \{s_1, s_2, \dots\}$. Each state corresponds to a unique combination of values for the internal variables $\mathbb{X} = \{x_1, x_2, \dots x_{\mathbb{X}}\}$. For instance, a single lightbulb is described by one logical state variable, and has two states. A couple of lightbulbs are described by two logical state variables, and the resulting system has four states.

The inputs of the systems are discrete *events*, denoted e_i, which are occuring instantaneously, and asynchrounously. The state evolution function becomes:

$$\forall k, X(k+1) = g(X(k), e_i)$$

A *state trajectory* is a succession of states reached by the system when a succession of events is given, as illustrated by Figure 1.2. In this system described by 6 states,

occurence of event e_1 at time t_1 causes a state evolution from state s_2 to state s_5, *i.e.* $X(2) = g(X(1), e_1)$.

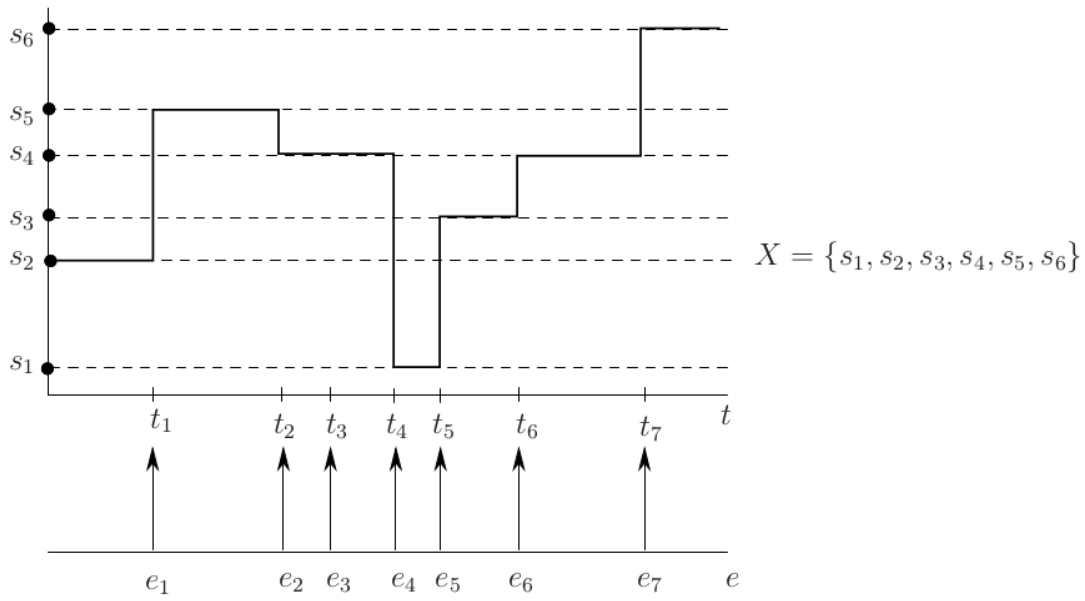


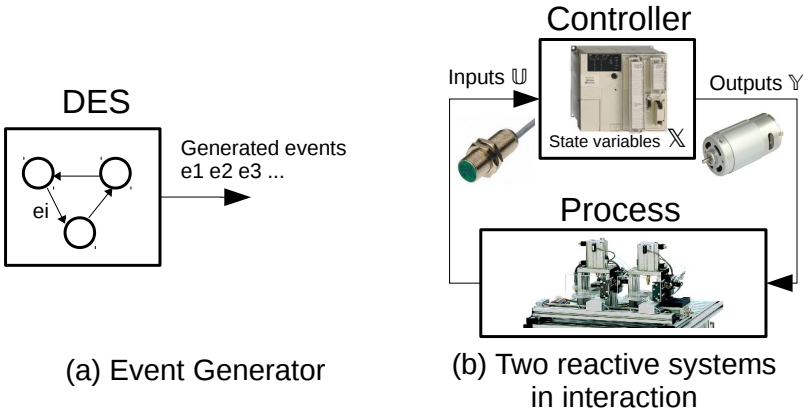
Figure 1.2: [Cassandras and Lafortune, 2008] An example of a state trajectory of a DES.

1.1.2 Reactive DES and event generators

The evolution of a DES is described by state trajectories, associated to event sequences. A DES can therefore be seen as a *spontaneous event generator*: the DES follows a state trajectory, and generates the events corresponding to said trajectory. The DES presented in Figure 1.2 generated successively $e_1, e_2, e_3, e_4, e_5, e_6$ and e_7 while following the trajectory $(s_2, s_5, s_4, s_1, s_3, s_4, s_6)$. Viewed as an event generator (Figure 1.3(a)), the DES does not interact with its environment.

However, some systems receive events generated by the environment, the internal state is updated according to the state transition function, and actions are performed by the system, transforming the environment.Such systems are called *reactive*. A reactive system always maintains an interaction with its environment, to react accordingly to inputs. DES might be reactive, in which case an adequate model must express in its semantics the reactions of the outputs of the system to the stimuli of the inputs.

As introduction to the remainder of this thesis, consider a closed-loop system consisting of a controller and a process (Figure 1.3(b)). The controller is a reactive DES, since it receives input signals from the physical sensors of the process and delivers orders (output signals) to the physical actuators of the process. The reactive behaviour is the triggering of outputs depending on both the inputs and the state variables. On the other hand, the plant is also a reactive DES, although the reactions are slower, due to mechanical movements, instead of just computation durations. Inputs (resp. outputs) of the plant are outputs (resp. inputs) of the controller.



(a) Event Generator

(b) Two reactive systems in interaction

Figure 1.3: A spontaneous event generator (a). A closed-loop system consisting in two reactive systems, interacting through their I/Os (b)

Given a reactive DES, input and output signals are its *external behaviour*, *i.e.* the way the DES interacts with its external environment. On the other hand, the state variables and the state evolution function represent its *internal behaviour*, which is not accessible to an external observer. Since only the external behaviour is accessible to an external observer, a reactive system is often perceived as an *event generator* (Figure 1.4): the variations of input and output signals become events. A reactive DES can be interpreted as an event generator, while the reverse is not always true.

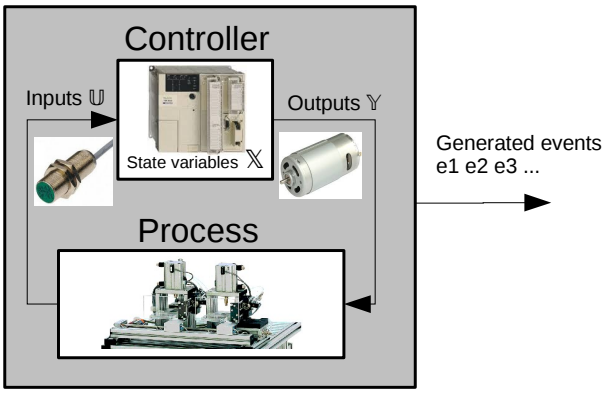


Figure 1.4: A closed-loop system consisting in two reactive systems, seen as an event generator

1.1.3 Formalism: Classical models

Modelling a system consists in designing its internal behaviour. The formalisms used for DES are briefly recalled in this section.

1.1.3.1 Languages and finite-state automata

When the DES is perceived as an event generator, its external behaviour is a set of strings of events $e_1e_2e_3\dots$, specifying in which order the events occur during the life of the system. A formal way of studying such a behaviour is by interpreting it as a language.

First, the set of all events that can be generated by the system is denoted $E = \{e_1, e_2, \dots\}$, and is analogous to the *alphabet* of a language. Then a language is formally defined over E:

Definition 1.2 (Language [Cassandras and Lafortune, 2008]). *A language defined over an event set E is a set of finite-length strings formed from events in E.*

A language might contain the empty word, denoted ε. A language is possibly infinite. For a string s, if there exists (u, v) such that $s = u.v$, u is called a *prefix* of s. A language L is *prefix-closed* if, given any string $s \in L$, all prefixes of s also belong to L. If a string is generated by a DES, then all prefixes have also been generated; DES are often modelled by prefix-closed languages.

An automaton is a state-machine capable of representing languages, and is a formalism widely used to represent the possible states X and the state transition function of a DES. Formally:

Definition 1.3 (Deterministic Finite-State Automaton (DFA)[Cassandras and Lafortune, 2008]). *A Deterministic Finite-State Automaton, denoted by G, is a six-tuple $G = (X, E, f, \Gamma, x_0, X_m)$ where:*

- *X is the finite set of states*
- *E is the finite set of events associated with G*
- *$f : X \times E \to X$ is the transition function:$f(x, e) = y$ means that there is a transition labeled by event e from state x to state y; in general, f is a partial function on its domain.*
- *$\Gamma : X \to 2^E$ is the active event function (or feasible event function); $\Gamma(x)$ is the set of all events e for which $f(x, e)$ is defined and it is called the active event set (or feasible event set) of G at x*
- *x_0 is the initial state*
- *$X_m \subseteq X$ is the set of marked states.*

The hypothesis of determinism can be lifted by replacing the output domain of f by 2^X, in which case, the machine is a simple *Finite-State Automaton or Machine* (FSM). If the initial and marked states are not given, the machine is called a *Transition system.* An example of a DFA is shown in Figure 1.5; it is defined on the event set $E = \{e_1, \dots e_9\}$,

and the states $X = \{X0, \ldots X11\}$. $X_m = \{X1, X4, X5, X9, X10\}$. Notice that all states from $X3$ to $X11$ represent concurrency between the executions of the strings e_4e_5 and e_6e_7, which can be interleaved in every possible way. Concurrency is not explicit in the structure of the automaton, as every interleaving is represented by a different path, multiplying the number of states and transitions.

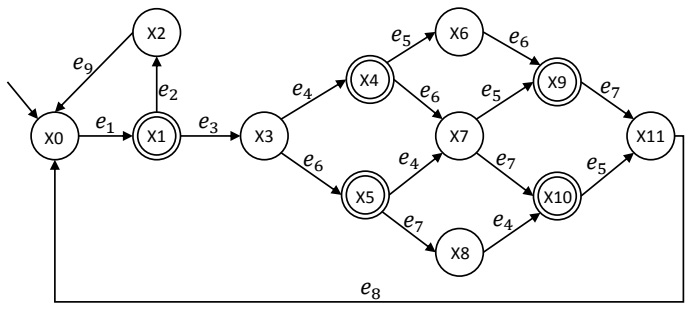


Figure 1.5: An example of a DFA

DFAs are able to generate languages, defined as following:

Definition 1.4 ([Cassandras and Lafortune, 2008])**.** *The language generated by* $G = (X, E, f, \Gamma, x_0, X_m)$ *is:*

$$L(G) := \{s \in E^* : f(x0, s) \in X\}$$

The language marked by G *is:*

$$L_m(G) := \{s \in L(G) : f(x0, s) \in X_m\}$$

E^* is the Kleene-closure of the set of events E, *i.e.* the infinite set of all possible finite-strings of elements of E (including the empty string ε). f here is the extended version of the transition function $f : X \times E^* \to X$. The language generated by the net is the set of all strings generated when following all paths starting in the initial state; the marked language is a restriction to paths ending in a marked state.

Additional semantics are required to differenciate inputs and outputs of a reactive DES. Moore ([Moore, 1956]) and Mealy ([Mealy, 1955]) machines offer this level of interpretation. Two event sets are considered: Σ as the set of input events, and Λ as the set of output events.

Definition 1.5. *A Moore machine is a DFA with the following modifications:*

- *the transition function only considers input events* $f : X \times \Sigma \to X$
- $g : X \to \Lambda$ *is an output function mapping each state to an output symbol.*

A Mealy machine is a DFA with the following modification:

- *the transition function* $f : X \times \Sigma \to X \times \Lambda$ *additionally maps output symbols to the transitions*

Each Moore machine can be converted to an equivalent Mealy machine (and reciprocally). As pointed out in [Cassandras and Lafortune, 2008], the couple input/output associated to a transition in a Mealy machine can be considered as a single event. Mealy machines can therefore be reduced to DFAs with event sets built on input/output couples; all properties of DFAs are valid for Mealy machines as well. The external behaviour of a DES modelled by a Mealy machine is therefore a string $s = (ie_1, oe_1)(ie_2, oe_2)(ie_3, oe_3)\ldots$, where each input symbol is associated to an output symbol. The event set becomes $E = \{(ie_i, oe_j), ie_i \in \Sigma, oe_j \in \Lambda\}$.

1.1.3.2 Petri Nets

Petri Nets (PNs) have been developped in the 70s, following the first investigations by Carl Adam Petri in his thesis [Petri, 1962]. They are especially useful to compactly represent dynamic, concurrent and undeterministic systems. They form with automata the main formalisms used for DES modeling. An advantage of PNs is their linear representation, which enables the use of linear algebra methods to verify properties of the nets.

Definition 1.6 (Petri Net structure)**.** *A generalized Petri Net structure G is a bipartite digraph represented by the 5-tuple $G = (P, T, I, O)$ where: $P = \{p_1, p_2, \ldots, p_{|P|}\}$ and $T = \{t_1, t_2, \ldots, t_{|T|}\}$ are finite sets of vertices named places and transitions respectively; $I(O) : P \times T \to \mathbb{N}$ is a function representing the edges going from places to transitions (from transitions to places), and associating a weight (or multiplicity) to each edge.*

$I(p_i, t_j) = 0$ means that the edge $p_i \to t_j$ does not exist. If the destination of $I(O)$ is restricted to $\{0, 1\}$, the structure is called *ordinary*. For a place p_i, the set of pre(post)-transitions $\{t_j \in T, I(O)(p_i, t_j) = 1\}$ will be written ${}^\bullet p_i (p_i^\bullet)$. Its *in-degree* (resp. *out-degree*) is the number of pre(post)-transitions: ${}^\bullet p_i$ (resp. $p_i^\bullet$). The degree δ is the sum of in- and out- degrees.

Definition 1.7 (Linear representation)**.** *A Petri Net structure is represented by its incidence matrix: $C = C^+ - C^-$, where $C^- = [c_{ij}^-]$; $c_{ij}^- = I(p_i, t_j)$ and $C^+ = [c_{ij}^+]$; $c_{ij}^+ = O(p_i, t_j)$ are respectively the pre-incidence and post-incidence matrices, also often noted Pre and $Post$.*

The structure is the static part of the net, where the transitions represent the events driving the system, and the places the conditions under which these events can occur. To describe the dynamic part, the notions of marking and firing are introduced.

Definition 1.8 (Marking)**.** *A marking function $M : P \to \mathbb{N}$ represents the number of tokens residing inside each place; it is usually expressed as a $|P|$-entry vector. $\mathbb{N}$ is the set of nonnegative integers. If $\mathbb{N}$ is replaced by $\{0, 1\}$, there is at most one token residing in any place, and the net is 1-bounded.*

Definition 1.9 (Petri Net system)**.** *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.*

A marking M represents a state of the system. In a Petri Net system, the state evolution is driven by the firing of transitions, given that the marking enables said firing. An example of a PN is shown in Figure 1.6, with $T = \{t_1, \ldots, t_9\}$, $P = \{P0, \ldots, P8\}$, and $M_0 = {}^t\left[100000000\right]$.

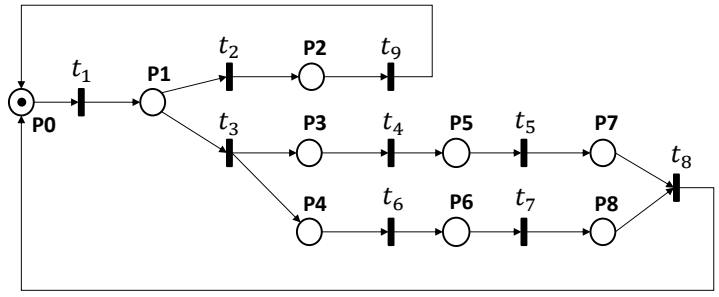


Figure 1.6: An example of a PN

Definition 1.10 (Enabled transitions and firing)**.** *In a PN system, a transition t_j is enabled at marking M_k if $\forall p_i \in P, M_k(p_i) \geq I(p_i, t_j)$, written $M_k \xrightarrow{t_j}$; an enabled transition t_j can be fired reaching a new marking M_{k+1}, written $M_k \xrightarrow{t_j} M_{k+1}$. It can be computed using the PN state equation: $M_{k+1} = M_k + C.\upsilon_k$ where $u_k(j) = 1; \upsilon_k(i) = 0, i \neq j$.*

If multiple transitions are enabled in a given marking M_k, any can be fired, but only one is actually fired, hence the possibility of modeling undeterminism. For instance, in Figure 1.6, t_1 is the only transition enabled at the initial marking. Then, after its firing t_2 and t_3 are enabled. Notice the representation of the concurrency after the firing of t_3; the firings of $t_5 t_7$ and $t_6 t_8$ are explicitly concurrent in this model.

The PN state equation, analogously to the state transition function of DFAs, can be extended to firing sequences instead of single firings.

Definition 1.11 (Firing Sequence)**.** *If $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} \ldots \xrightarrow{t_k} M_k$, then $\sigma = t_1 t_2 t_3 \ldots t_k$ is a firing sequence leading to marking M_k, and written $M_0 \xrightarrow{\sigma} M_k$. It can be computed using the PN state equation: $M_{k+1} = M_0 + C.\Upsilon$ where $\Upsilon(j)$ is equal to the number of firings of t_j in S.*

If a single transition firing is equivalent to an event generated by the net, a firing sequence is equivalent to a word. Different languages have been defined for Petri Nets ([Jantzen, 1987]), depending on markings or deadlocks; only the most generic, P-type language is recalled here:

Definition 1.12 (Language generated by a PN)**.** *The language generated by a PN (G, M_0) is the set of all firing sequences σ enabled from the initial marking M_0. The alphabet associated to this language is the set of transitions of the net, i.e. $L(G, M_0) = \{\sigma \in T^*, M_0 \xrightarrow{\sigma}\}$.*

Definition 1.13 (Reachability set)**.** *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 $R(G, M_0)$.*

The reachability set can be represented as a reachability graph, each marking corresponding to a node, and a transition firing to an edge. The language generated by the reachability graph viewed as a state machine is the language generated by the PN. For instance, the reachability graph of the net of Figure 1.6 is isomorphic to the automaton depicted in Figure 1.5.

However, the languages defined up to now assume a bijection between the alphabet and the set of transitions (*i.e.* each transition name is a symbol). Labelled Petri Nets are defined without this assumption, and use a labelling function instead:

Definition 1.14 (Labelling function)**.** *Let E be a set of symbols. A labelling function is a function $\lambda : T \to E$, and a Labelled Petri Net is a PN coupled with a labelling function. Its language becomes:*

$$L(G, M_0) = \{\lambda(\sigma) \in E^*, M_0 \xrightarrow{\sigma}\}$$

If λ is a bijection (i.e. an event can not be associated to multiple transitions), the language is called free. Otherwise, if the empty-word ε is not used, the language is called λ-free; otherwise, it is called arbitrary.

LPNs offer the possibility to add input information to the transitions. To model reactive systems, it remains to add also output information, namely to places. This is the purpose of Interpreted Petri Nets [David and Alla, 1994]:

Definition 1.15 (Interpreted Petri Nets)**.** *An Interpreted Petri Net system (IPN) Q, $Q = (G, M_0, \mathbb{U}, \Sigma, \lambda, \mathbb{Y}, \varphi)$, is based on an ordinary PN system (G, M_0) to which are added:*

- $\mathbb{U} = \{u_1, u_2, \ldots, u_{|\mathbb{U}|}\}$ *the known input alphabet*
- $\Sigma = \{\uparrow u_i, \downarrow u_i \mid u_i \in \mathbb{U}\}$ *the set of events.*
- $\lambda : T \to \{0, 1\}$ *the labelling function of transitions.*
  $\forall t_i \in T, \lambda(t_i) = F_i(\mathbb{U}) \bullet G_i(\Sigma)$ *where:*
  - $F_i : \mathbb{U} \to \{0, 1\}$ *is a boolean function depicting the conditions on the levels of the inputs to fire t_i*
  - $G_i : \Sigma \to \{0, 1\}$ *is a boolean function depicting the conditions on the input events to fire t_i*

  $\lambda(t_i) = 1$ *iff* $F_i(\mathbb{U}) = 1 \wedge G_i(\Sigma) = 1$
- $\mathbb{Y} = \{y_1, y_2, \ldots, y_{|\mathbb{Y}|}\}$ *the known output alphabet*

- $\varphi : R(G, M_0) \to \{0,1\}^{|\mathbb{Y}|}$ *the output function that returns the value of the outputs given a marking of the net.*

Notice that G_i can contain multiple input events, which is suitable for adaptation to technology, as will be shown in Section 1.2. In the framework of Interpreted Petri Nets, the firing of a transition is no longer autonomous, but constrained by its labelling function. A transition t_j is fired when the following conditions are verified:

- t_j is enabled (by the marking)
- $\lambda(t_j) = 1$ (both F_i and G_i are True)

The output function can be restricted to $\varphi : P \to (Y) \cup \varepsilon$, *i.e.* outputs are associated to places instead of markings and a place can be associated to only one output. Places associated to an output are called *measurable* or *observable*, while places associated to ε, *i.e.* no output, are called *non measurable/unobservable*. The set of places P is then partitionned into two sets of observable and unobservable places, $P = P^{Obs} \cup P^{Unobs}$.

An example of IPN is shown in Figure 1.7; it is the net of Figure 1.6 with an additional layer of interpretation. Three observable places have been linked to the outputs $\mathbb{Y} = \{Y1, Y2, Y3\}$, all other places are unobservable. Various event and level conditions on the inputs $\mathbb{U} = \{u0, \ldots, u4\}$ are assigned to the transitions, t_5 and t_6 namely have both, whereas t_7 and t_8 have none (*i.e* ε for no event required, and $(= 1)$ for a condition always true).

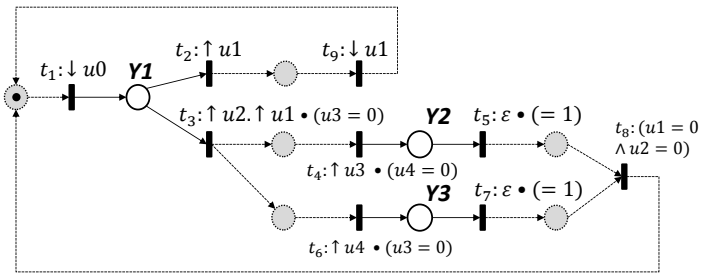


Figure 1.7: An example of an IPN

1.2 Identification of a DES: Problem Statement

System identification consists in building a mathematical model of the behaviour of a system from a finite observation of said system. On the opposite of manual design based on expert knowledge, an identification approach is experimental; the system is observed during its operation, and a model is built with the objective of fitting the observation as well as possible, according to an approximation criterion.

In continuous system identification, the problem consists in choosing a model class, and adjust the parameters to best fit the observed data, in which case it is *parametric*. For instance, record the rotation speed of a motor who is delivered a step of tension, and

choose a first order model, whose parameters are the amplification factor and the time-constant. The task consists in choosing the best combination of the two parameters, such that the response output of the model best fits the real response, according to the Least Squares Estimator.

Regarding DES, the task of identification consists in building a model of both the internal and external behaviours, *i.e.* the state space, state evolution function and output function, based on the observation of the I/O signals only. If information on the structure of the internal behaviour is known, the identification is *whitebox*. When only the external behaviour is accessible, the identification is *blackbox*.

System identification is a method widely spread in the context of continuous-time systems; however, most DES models are still manually built by experts. Numerous methods and guidelines for efficient modelling have been proposed in the literature ([Girault and Valk, 2003]). The aim of this thesis is instead to contribute to the development of identification methods. The focus is set on reactive DES, under blackbox hypothesis.

1.2.1 Systems of Interest

In this section, the systems of interest of the whole thesis are described. Namely, the technology of the components and the experimental conditions are constraints that are to be considered by any identification algorithm.

We are interested in real closed-loop systems, consisting in a controller and a process (Figure 1.3). The point of view of the controller is adopted. The sensors (optical, inductive, switches, ...) deliver input signals $\mathbb{U}$, and the controller delivers output signals $\mathbb{Y}$ to actuators (pneumatic or hydraulic actuators, contactors, ...). All these signals are logical: their values are either 0 or 1.

The aim is to discover the relationships between the controller and its environment, so we have no interest in isolating the controller from the process. Therefore the observation is *passive*. In an active identification approach, outputs of the controller could be forced to see the reaction of the system; without strong safety constraints, this could result in component damage.

The controller can be assumed deterministic, since it is programmed to always act the same way in the same conditions. The process is however non-deterministic, due to numerous mechanical or physical factors. If two cylinders start their extension simultaneously, the order of the respective ends of their extensions might vary, depending on fluid pressure, gripping, mechanical defects,... The system as a whole is therefore non-deterministic.

The focus is set on Industrial Programmable Logic Controllers (PLC), which are widely used in manufacturing industries. Architecture and programming of these specific controllers are extensively detailed in the [IEC 61131-1..IEC 61131-8, 2010] norm. The main characteristic of these controllers is the cyclicity of their behaviour, as recalled in Figure 1.8. A PLC cycle typically lasts a few milliseconds. It consists in three successive

actions:

Input reading (I) The controller gets the values of the inputs stored in the input modules, *i.e.* the statuses of the sensors of the process. These logical informations are stored in the variable table of the controller, and left unchanged until the next reading.

Program Execution (PEX) The implemented program is executed one time. The values of all internal variables and outputs are computed and stored in the variable table.

Output Writing (O) The values of output variables stored in the variable table are sent to the actuators of the process via the output modules. They are left unchanged until the next writing.

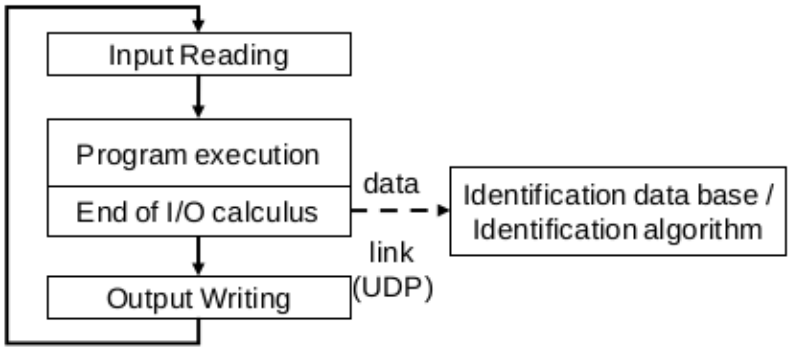


Figure 1.8: The cyclic behaviour of a Programmable Logic Controller, including the passive observation

The controller can be described as a sequential system. It is in a given state $X(k)$ at every cycle k. The state is updated, and outputs are generated at every cycle of the controller, using the state evolution function and output functions:

$$\forall k \begin{cases} X(k+1) = g(X(k), U(k+1)) \\ Y(k+1) = f(X(k+1), U(k+1)) \end{cases}$$

Due to the blackbox approach, the states $X(k)$ are unknown *a priori*. To observe the system, an experimental, non invasive protocol is required to record the values of the input and output signals. Such a protocol is proposed in [Roth et al., 2010b]. The observation is conducted at the end of the program execution. The values of all external variables, *i.e.* inputs and outputs, are sent via a UDP connexion to an external computer. Therefore, an observation is performed during each cycle of the system; it consists in the input values read at the start of the cycle, and the output values computed at the end of the program execution. Formally, given a cycle k, an observation $w(k)$ is a vector

defined by:

$$w(k) \in \{0,1\}^{1\times(|\mathbb{U}|+|\mathbb{Y}|)}; w(k) = {}^t[u_1(k), \dots u_{|\mathbb{U}|}(k), y_1(k), \dots y_{|\mathbb{Y}|}(k)]$$

with $u_i(k)$ (resp. $y_j(k)$) the value of the input u_i (resp. output y_j) in the cycle k.

The succession of all observed vectors $w(1)w(2)\dots w(k)\dots$ is an *observed vector sequence* w. From two successive observations, an *event vector* $E(k)$ is computed to represent the elementary changes of inputs and outputs that occurred during the two observations. Formally, $E(k) = w(k+1) - w(k)$, with:

$$E_i(k) = \begin{cases} 1 \text{ if a rising edge of i/o } i \text{ occurred} \\ -1 \text{ if a falling edge of i/o } i \text{ occurred} \\ 0 \text{ if the value of i/o } i \text{ is unchanged} \end{cases}$$

If $\forall i, E_i(k) = 0$, nothing occurred, denoted $[\varepsilon]$. The succession of all event vectors $E(1)E(2)\dots E(k)\dots$ is an *event vector sequence* E.

To better understand the constraints and the computation of the observed sequences, have a look at the evolution in chronogram (a) of Figure 1.9, based on an observed sequence $w_{(a)}$. The program ensures that the output y_1 (y_2) has the same value as the input u_1 (u_2). Initially, all inputs and outputs are 0. u_1 is set during the first PLC cycle, but its value is not read until the input reading step of the second cycle. During the PEX step of the second cycle, the program updates the value of y_1 in its variable table since u_1 has changed, leading to the set of y_1 during the output writing step. Same behaviour for u_2 and y_2 during the third cycle.

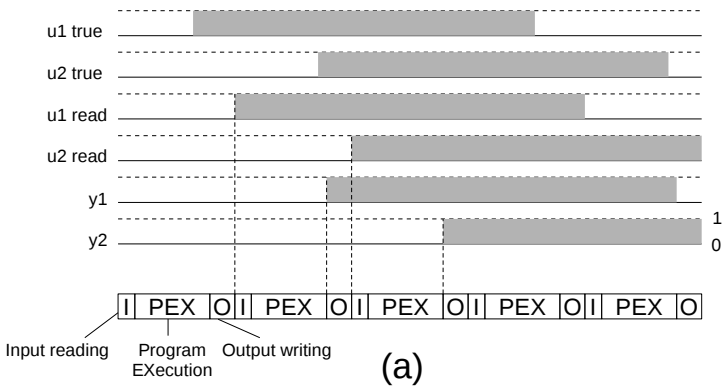


Figure 1.9: An observation of the process controlled illustrating the functioning of the PLC

The observed vector sequence $w_{(a)}$ and the event vector sequence $E_{(a)}$ are then:

$$w_{(a)} = \begin{matrix} u_1 \\ u_2 \\ y_1 \\ y_2 \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \quad E_{(a)} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ -1 \\ 0 \end{bmatrix}$$

$$E_{(a)} = [\uparrow u_1 \uparrow y_1]; [\uparrow u_2 \uparrow y_2]; [\varepsilon]; [\downarrow u_1 \downarrow y_1]$$

However, the phase of input reading occurs only once per cycle. Two asynchronous input changes can be perceived as synchronous, as illustrated by chronogram (b) of Figure 1.10, based on the following vector sequence $w_{(b)}$ and event vector sequence $E_{(b)}$:

$$w_{(b)} = \begin{array}{c} u_1 \\ u_2 \\ y_1 \\ y_2 \end{array} \begin{bmatrix}0\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\end{bmatrix} \begin{bmatrix}1\\1\\1\\1\end{bmatrix} \begin{bmatrix}1\\1\\1\\1\end{bmatrix} \begin{bmatrix}0\\1\\0\\1\end{bmatrix} \quad E_{(b)} = \begin{bmatrix}0\\0\\0\\0\end{bmatrix} \begin{bmatrix}1\\1\\1\\1\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\end{bmatrix} \begin{bmatrix}-1\\0\\-1\\0\end{bmatrix}$$

$$E_{(b)} = [\varepsilon]; [\uparrow u_1 \uparrow u_2 \uparrow y_1 \uparrow y_2]; [\varepsilon]; [\downarrow u_1 \downarrow y_1]$$

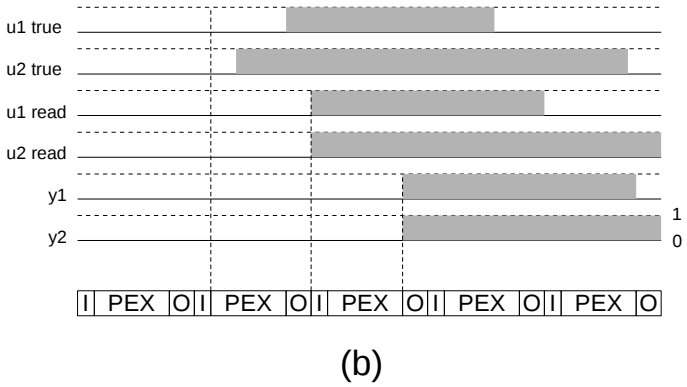


Figure 1.10: An illustration of the synchronization effect.

Generically speaking, multiple inputs and outputs can change between two observations; hence the introduction of event vectors instead of using single events. It is possible to create an alphabet by using the event vectors as symbols; the event vector sequence becomes a string of the language, and classical DES theory applies then.

Finally, PLC often include timers, or memory variables, which are unknown in a blackbox approach. Therefore, input events often occur without any output change in the same cycle, having a delayed action on the outputs. Similarly, the delayed ouput can occur without any concurring input event. An instance is proposed in the chronogramm of Figure 1.11; one PLC cycle is blank before the output reacts to the input event, the event vector sequence being:

$$E_{(c)} = [\uparrow u_1]; [\varepsilon]; [\uparrow y_1]; [\downarrow u_1 \downarrow y_1]$$

To sum things up, an observation is a sequence of vectors, and multiple values (indiferrentially inputs or outputs) can change between two vectors. Due to the synchronisation of the controller, input and output events observed simultaneously might nevertheless not be related.

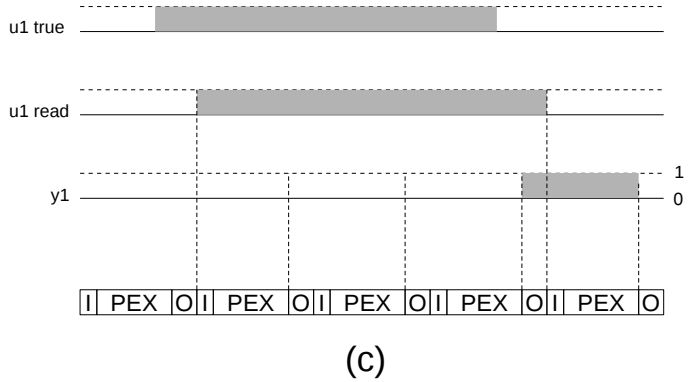


Figure 1.11: An illustration of a delayed output event

1.2.2 Incompleteness of the observation

Generically speaking, the more data on the system is available, the closer to reality the model built on this data is. If the observation was infinite, then all possible behaviours could have been observed. However, massive concurrency of systems hinders this possibility as shown in [Roth et al., 2009a][Roth, 2010].

The DES is therein considered as an event generator, hence has an output alphabet Ω. Each different observed vector w(k) is a letter of this alphabet. The system is observed during p production cycles (observed vector sequences $w_1, \ldots w_p$, of respective length $l_1, \ldots, l_p$), and the set of observed words of length q is introduced:

Definition 1.16. *[Roth et al., 2009a] The observed words of length q are denoted as:*

$$W^q_{Obs} = \bigcup_{i=1}^{p} (\bigcup_{k=1}^{l_i-q+1} (w_i(k), w_i(k+1), \ldots, w_i(k+q-1))$$

Then, the behaviour of the system is defined by its observed language of length n, *i.e.* the language containing all observed words of length up to n:

Definition 1.17. *[Roth et al., 2009a] The observed language of length n is*

$$L^n_{Obs} = \bigcup_{i=1}^{n} W^i_{Obs}$$

By observing more production cycles, new words are observed, and the sizes of the observed languages L^n_{Obs} grow with the duration of the observation. For systems with low concurrency, it is possible to observe an asymptote, since the ordering of the output events remains the same during multiple observations of the same processing. On the contrary, highly concurrent systems exhibit lots of possible orderings of events; the number of observed words keeps raising with the duration of observation, as illustrated in Figure 1.12.

For low values of n, it is possible to make the assumption that every word that could be emitted by the system was observed, but not for higher values of n. An example of

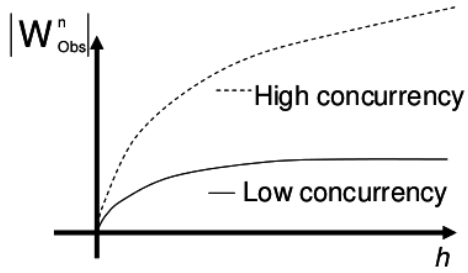


Figure 1.12: [Roth et al., 2009a] Evolution of the number of observed words of length n over production cycles h

a real system (described in [Roth, 2010]) whose convergence is valid only for $n = 1$ over a hundred production cycles is presented in Figure 1.13.

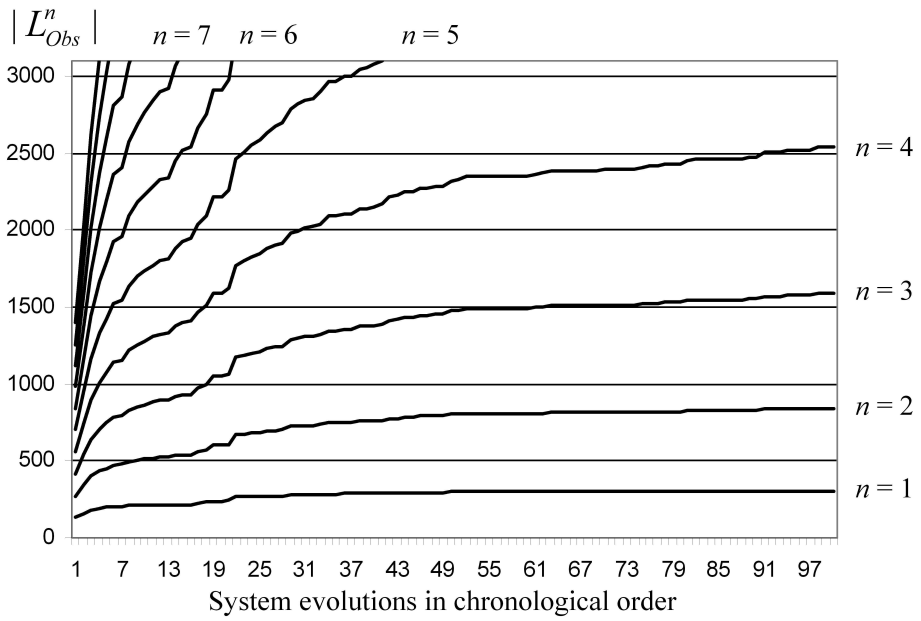


Figure 1.13: [Roth, 2010] Observed languages over a hundred production cycles

Reasonably, the hypothesis is therefore made that the observation, being finite, is always incomplete. The belonging of a non-observed word to the original language of the system is undecidable. Therefore, no counter-examples can be used in the context of identification of real systems. This caracteristic is highly important, as some related works are based instead on the design and use of counter-examples ([Giua and Seatzu, 2005][Cabasino et al., 2015]). The authors of [Basile et al., 2016c] notably propose to compute timed counter-examples from observed timed sequences, assuming that lower and upper bounds of execution times are known; they are however unavailable in our blackbox approach.

1.2.3 Problem Statement

The following constraints and hypotheses on the systems of interest have been exhibited in the previous section:

- Logical input and output signals
- Multiple input-output change between two observations
- Inputs might have delayed actions on outputs
- The couple controller/process is non deterministic
- The system can exhibit massive concurrency
- The identification is blackbox, and passive
- No knowledge of counter-examples

We wish to build, from a vector sequence w observed under the above conditions, a model that non only reproduces w, but also exhibits the reactive behaviour of the system in its structure. The main problem of this thesis is stated:

Problem Statement *Consider a logical closed-loop system controlled by a PLC. Identify, from an observed vector sequence w, a model that exhibits the reactive behaviour and reproduces w.*

In this problem, no explicit representation of time is required in the model; timed phenomena are present, but the identification of time bounds and their explicit addition in the model are not required. Cyclicity is not required as well; observations do not require to start and end at the same point.

An additional objective is to get an understandable, compact behavioural model, to satisfy a goal of reverse engineering as presented in next section.

1.2.4 Identification for reverse engineering

Reverse engineering is defined in [Chikofsky and Cross II, 1990] as *the process of analyzing a subject system to identify the systems components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.* Identification is a natural method to obtain these representations of the system. Instead of formulating theoretical models, they are constructed from the observation of the real system in its environment.

Potential applications are certification, reimplementation, or debug. In the latter case, an example is [Prähofer et al., 2013], where the authors are interested in the reverse engineering of the program of the PLC. The behaviour called reactive in this article is a change of the path followed by the program depending on the inputs values. The idea is to obtain a model of the paths followed, for analysis and visualization.

For a reactive DES, a good reverse-engineered model should provide insight on the characteristics of the system, such as concurrency, non-determinism or input/output causalities. These caracteristics should be readable in the model, and easily understandable by an engineer. Readable models will facilitate the job of program certification, or reimplementation in a controller.

The choice of the model class should therefore be conditionned by the purpose deserved by the model. Hence, the upcoming literature review exhibitis different methods and models, and discusses their adequation to reverse-engineering. The choice of IPNs as a satisfying model class ensues from this review.

1.3 Identification in the literature

1.3.1 Origin: early computer science approaches

1.3.1.1 Language inference

The problem of DES identification can be retraced to the 60s, and the beginning of artificial intelligence, that motivated the study of language identification [Gold, 1967], also called language inference [Angluin, 1982]. An accurate history is presented in these two surveys on identification: [Estrada-Vargas et al., 2010] and [Cabasino et al., 2015].

The idea of [Gold, 1967] is to present information, as a set of strings s_1, s_2, ..., to a learner which has to guess the language L; this is quite similar to the observation obtained from a DES (as event generator), where the identification algorithm plays the role of learner. The language L is said *identifiable in the limit*, if after some time, the learner always guesses the correct language. Two ways of presenting information are considered: either a *text* containing every string from L (hypothesis of full knowledge of the language), or an *informant*, which has absolute knowledge on the membership of strings s_i to L.

This notion of informant, oracle, or *teacher* was explored in [Angluin, 1987], where the learner can submit queries to the teacher, which can also exhibit counter-examples (strings not belonging to the language). The learning algorithm L^* proposed is able to build a deterministic finite-state acceptor (*i.e.* a DFA) from the information given by the teacher. Full knowledge of the language is a recurrent characteristic of language inference methods, which is not accessible in the case of DES identification.

Angluin's L^* algorithm is still used in recent works in the field of computer science, such as [Shahbaz and Groz, 2009], where the complexity of the processing of counterexamples is reduced.

1.3.1.2 Moore and Mealy machines

Other methods have focused on the inference of Moore or Mealy machines from input/output sequences.

In [Kella, 1971], a Mealy machine with $n + 1$ states is built from one input/output

event sequence of length n: $(i_1, o_1)(i_2, o_2)\ldots(i_n, o_n)$. The problem is then to find all possible reduced machines by state merging operations. The resulting machines are not *'completely specified'*, in that they exhibit exceeding behaviour.

[Gill, 1966], then [Heun and Vairavan, 1976] have invested required properties of a set of finite input/output strings in order to be realizable by Mealy machines. [Gill, 1966] characterizes *compatibility*, extended by [Heun and Vairavan, 1976] into *consistency*. An algorithm constructing the Mealy machine is provided as well.

In [Biermann, 1972], the maximal number of states of the Mealy machine to be identified is set. The input/output pairs of the sequence are incrementally dealt with, and guesses consistent with the sequence are made on the structure. If no machine can reproduce the behaviour, the procedure is repeated with an increased number of states. Consequently, the final solution has the fewer states of all machines whose behaviour comprises the sample set (*i.e.* is minimal).

The same author has also investigated non deterministic Moore machines in [Biermann and Feldman, 1972]. A set of i/o sequences is given, where the output is only emitted with the last symbol of the sequence; these sequences are assumed to start from the same initial state. An initial machine is contructed as a tree, then reduced by state merging rules.

Finally, [Veelenturf, 1978] has proposed an iterative algorithm to successively build Mealy machines from a set of observed input/output sequences, such that the behaviour of each machine is included in the result of the previous iteration. New machines are built by adding transition and/or states to the previous one. The sequences are always treated as a whole, contrary to [Biermann, 1972] that deals incrementally with input/output pairs.

The same author has proposed a learning algorithm to identify a Moore machine in [Veelenturf, 1981]. The algorithm guesses structural properties, detects eventual contradictions with the observed i/o sequences, and corrects by removing transition. The inferred automata is minimal.

In all these methods, the notion of input/output is not exploited; the couple (i_k, o_k) is interpreted as a symbol. Namely, all the algorithms discovering Mealy machines could be applied on problems without interpretation; they are not specific to represent reactive behaviour.

After this short historical tour, recent approaches in the field of Discrete Event Systems are presented.

1.3.2 Identification by Automata

1.3.2.1 Identification of closed-loop processes

These approaches consider systems close to the ones presented in Section 1.2.1.

A whitebox approach for PLC-controlled processes is presented in [Prähofer et al., 2014]. An observation consists in recording the inputs, outputs and taking snapshots of

the program state, from which the path followed by the program in a functional block in a given cycle is deduced (Figure 1.14). Knowledge of the internal state of the controller is therefore known. An FSM is built from these traces, to which are ultimately added time windows. The FSM is a high-level model, representing paths, but the low-level relations of inputs and outputs are not considered.

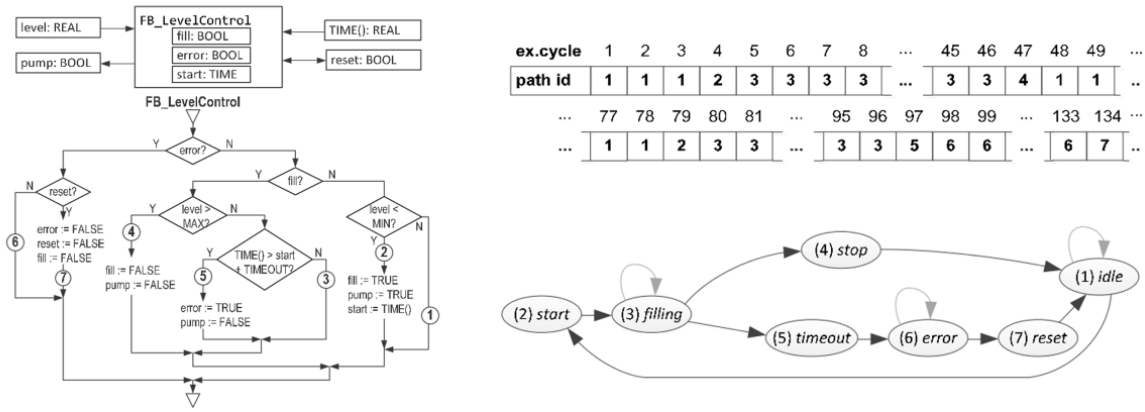


Figure 1.14: [Prähofer et al., 2014] Left: A flow chart of a function block of the PLC, with the paths numbered. Right: A trace and its representation as a FSM

Identification of PLC-controlled processes have been considered in a series of works. [Klein, 2005][Roth, 2010][Schneider et al., 2012]. The methods proposed in these works have been conceived for fault diagnosis applications. In [Klein, 2005], the identified models are Non Deterministic Autonomous Automata with Outputs (NDAAO). Formally:

Definition 1.18 ([Klein, 2005]NDAAO)**.** *A non-deterministic autonomous automaton with output, is a five-tuple* $NDAAO = (X, \Omega, f_{nd}, \lambda, x_0)$ *where* X *is the finite set of states,* Ω *is the finite set of output symbols,* $f_{nd} : X \to 2^X$ *is the non-deterministic transition relation,* $x_0 \in X$ *is the initial state, and* $\Lambda : X \to \Omega$ *is the output function.*

In the proposed identification approach, the closed-loop system is considered as an event generator; the choice has been made to build a model that can spontaneously generate events as well. The transition relation is not conditionned by events, hence the autonomy of the model. When a state is reached after a firing, an element of the output alphabet is emitted (as in a Moore machine). Since the observation of the system consists in vectors $w(k)$ (see Section 1.2.1), each different observed vector is mapped to an output symbol. Thus, the model can generate output sequences homogeneous to the observed vectors.

An example of the identification procedure is given below, for a plant with two inputs and one output. Notice that the observed sequences start and end with the same vector, which assumes cyclicity of the process observed, an hypothesis not made in our problem.

The observed data is:

$$\sigma_1 = \underset{A}{\begin{bmatrix}0\\0\\0\end{bmatrix}}\underset{B}{\begin{bmatrix}0\\1\\0\end{bmatrix}}\underset{C}{\begin{bmatrix}1\\1\\1\end{bmatrix}}\underset{D}{\begin{bmatrix}0\\1\\1\end{bmatrix}}\underset{E}{\begin{bmatrix}0\\0\\1\end{bmatrix}}\underset{A}{\begin{bmatrix}0\\0\\0\end{bmatrix}}\sigma_2 = \underset{A}{\begin{bmatrix}0\\0\\0\end{bmatrix}}\underset{D}{\begin{bmatrix}0\\1\\1\end{bmatrix}}\underset{B}{\begin{bmatrix}0\\1\\0\end{bmatrix}}\underset{C}{\begin{bmatrix}1\\1\\1\end{bmatrix}}\underset{D}{\begin{bmatrix}0\\1\\1\end{bmatrix}}\underset{A}{\begin{bmatrix}0\\0\\0\end{bmatrix}}\underset{C}{\begin{bmatrix}1\\1\\1\end{bmatrix}}\underset{A}{\begin{bmatrix}0\\0\\0\end{bmatrix}}$$

A length parameter k is fixed to transform the observed strings into sequences of substrings of length k. Hence, for $k = 2$:

$$\sigma_1^2 = ((A, A)(A, B)(B, C)(C, D)(D, E)(E, A)(A, A))$$

$$\sigma_2^2 = ((A, A)(A, D)(D, B)(B, C)(C, D)(D, A)(A, C)(C, A)(A, A))$$

Each substring is associated to one state of the identified NDAAO (Figure 1.15(a)). Then, each state is renamed with the last letter of the asociated substring, and equivalent states are merged (two states being equivalent if they have the same output letter, and the same set of post states). Consistently with the hypothesis of cyclicity, the first and last states are considered equivalent as well. The result is presented in Figure 1.15(b).

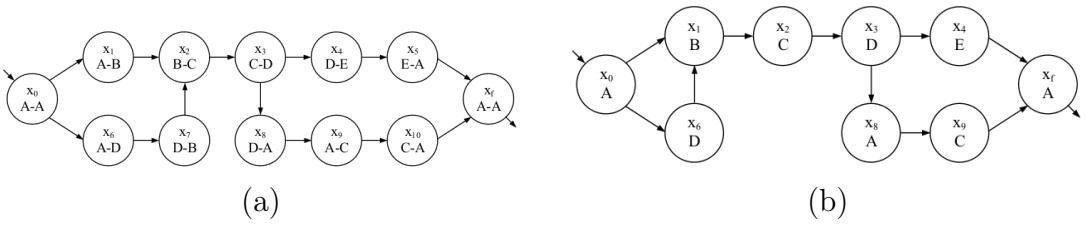


Figure 1.15: [Klein, 2005](a) The identified NDAAO after the treatment of the sequences; (b) The simplified model after merging of equivalent states

Such models have been used for fault diagnosis in [Roth et al., 2009b]. The idea is to test, during the execution of the process, whether the i/o behaviour is reproducible by the automaton, and compute the residuals, *i.e.* differences between the observed and the repoducible behaviour. Notice that the inputs and outputs are however undifferentiated, which is fine for fault diagnosis approaches, but not for reverse-engineering.

The parameter k must be finely tuned. It was proven in [Klein, 2005] that for all k, $L_{Obs}^{k+1} = L_{Iden}^{k+1}$, *i.e.* the identified automaton reproduces all and only all substrings of length $k + 1$ observed in the sequences σ_i. From a diagnosis point of view, the higher k, the less faults will be missed. In addition, it was proven in [Roth, 2010] that if $L_{Orig}^{k+1} = L_{Obs}^{k+1}$, then $L_{Orig}^{k+n} \subset L_{Iden}^{k+n}$, *i.e.* if it can be assumed that all words of length $k+1$ have been observed, then the identified net can reproduce every string from the original behaviour. For diagnosis, this corresponds to a minimization of false alerts. The choice of k is therefore based on the convergence of the size of observed languages, such as presented in Section 1.2.2.

The method was then extended to include timed transitions [Schneider et al., 2012], and to distributed identification of automata networks ([Roth et al., 2009a]). The prob-

lem of partitioning of a system for distributed identification will be reviewed in Chapter 5.

An approach similar to [Schneider et al., 2012] was proposed in [Maier, 2014], with similar hypotheses. An event is a vector of I/Os who are not differentiated, automata with timed transitions are identified, and the model is to be used for diagnosis. [Schneider et al., 2012] first computes the logical structure of the automaton, then adds time intervals to transitions, when [Maier, 2014] computes a timed prefix tree, then merges equivalent states (approach similar to [Biermann and Feldman, 1972]).

1.3.2.2 Miscellaneous

The following approaches are disconnected from the physical inputs and outputs of a real system, but propose original methods for identifying automata.

[Aguilar, 2011] proposed to use evolutionary computing to identify a Mealy machine. During the evolution of the population, the best candidates are those who can reproduce all presented i/o sequences, and eventually have the smallest set of states. Since the approach is a metaheuristic, there is no guarantee that the best models reproduce the sequences.

Active identification methods have been considered to identify timed automata. The authors of [Grinchtein et al., 2005] use a teacher and a learner, which can be assimilated to active identification, therefore hard to reproduce on real systems. A very similar approach was developped later in [Jarvis, 2010] where a variant of Angluin's L^* algorithm is designed. The result is not a timed automata, but a Mealy machine where i/os are timelines, called Timed Event Systems. The method seems adapted for deterministic systems only, as the timelines are defined by single values instead of intervals.

The work presented in [Saives and Faraut, 2014] takes place in a different context, which is Activity Discovery of inhabitants in Smart Homes. Data mining approaches are used to build an Extended Finite Automaton (an automaton with guards and variables) from a set of observed sensor events. The resulting model is an automaton that represents the sequences of events mirroring the frequent habits of the inhabitant. The method was extended in [Saives et al., 2015b] to include a diagnosis method, to detect deviations in the behaviour of the inhabitants. The method is unapplicable to reactive systems, since only sensor events are considered. Besides, sequence mining approaches are statistical approaches which extract relations between the events in sequences. Even if most of the behaviour contained in the sequence w is reproducible, any data mining approach can not guarantee fitness, *i.e.* that the whole sequence is firable.

1.3.2.3 Positioning on identification by Automata

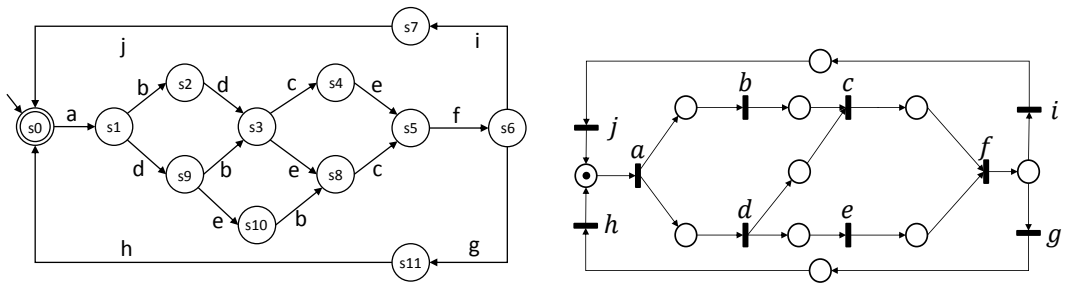
The huge drawback of automata for our reverse-engineering problem is their simplicity: impossible to represent compactly concurrent phenomena. Nevertheless, in cases where the model is not destined to be read by human operators, for instance diagnosis,

simplicity of automata makes them good models, who can be computed and manipulated at low cost.

1.3.3 Identification by Petri Nets

One of the first works in PN identification is [Hiraishi, 1992]. The method firstly builds a DFA with one final state from a set of sequences, then builds a PN whose language is equal to the one of the DFA.

Consider the set of sequences $\{\varepsilon, abdcefij, abdecfij, adbcefgh, adebcfgh\}$. The identified DFA is shown in Figure 1.16(a). States s0 to s7 are created with $abdcefij$, then s8 with $abdecfij$, then s9 with $adbcefgh$ and finally s10 and s11 with $adebcfgh$.



(a) (b)

Figure 1.16: From [Hiraishi, 1992]: (a)The finite acceptor and (b) the PN deduced from it.

Then, one transition is created for each event, and language dependencies *Ldep* are constructed from the DFA:

$$Ldep =$$
$$\{(h,a),(j,a),(a,b),(a,d),(b,c),(d,c),(d,e),(c,f),(e,f),(f,i),(f,g),(i,j),(g,h)\}$$

These dependencies are homogeneous to strings of length 2. Transition dependencies are built from the DFA. If two transitions have been consecutively observed in both orders, they are therefore considered independent (concurrent).

$$D^-(t) = \{t', (t',t) \in Ldep \wedge (t,t') \notin Ldep\}$$
$$D^+(t) = \{t", (t,t") \in Ldep \wedge (t",t) \notin Ldep\}$$

Then one place is created for each dependency, leading to the PN of Figure 1.16(b). Its language is equal to the language of the DFA of Figure 1.16(a).

This method is not applicable without the hypothesis of cyclicity, or if an event occurs twice in the same observed sequence. This approach requires the building of an intermediate DFA, hence of a language, which notably includes exceeding strings (for instance, $adbecfij$). This first step is homogenous to numerous approaches presented in the previous section.

Recent PN identification approaches can be split in two classes: the ones homogenous to the second step of [Hiraishi, 1992], which aim at building a net from a complete

description of its behaviour (given as a language or a transition system), and the ones who build nets directly from an incomplete collection of observed sequences.

1.3.3.1 Building PN from completely observed languages

The problem of building a net from a language has received some attention in the literature. The approaches presented here however deviate from identification as presented in Section 1.2, as there is no experimental background; however they are perfectly adapted for building nets from specifications given as languages.

The problem here consists in creating a PN from a given original language L. A very good review of this problem can be found in [Cabasino et al., 2015], where the authors distinguish two problems. They call *net synthesis* the problem of building a Petri net N such that $L = L(N)$, whereas the problem of *identification* aims for an inclusion $L \subseteq L(N)$. This section presents results related to the first case, in which a full set of counter-examples is available, since every string not belonging to the original language must be forbidden in the net. Recall that counter-examples are never available in identification of real systems.

Region theory

Region theory has been developped in order to synthesize a Petri Net N from a specification given as a DFA, such that the reachability graph of N is isomorphic to the original DFA, hence no exceeding language is generated by the net. Extensive details on research in this field can be found in a recently published book [Badouel et al., 2015].

Polynomial algorithms exist to solve this problem for bounded Petri Nets [Badouel et al., 1995]. However, the same problem for the specific subclass of Elementary Nets is proven to be NP-complete [Badouel et al., 1997]. An Elementary Net is a 1-bounded PN, the marking being defined as a subset of the places. Additionally, it can neither contain self-loops, nor duplicate places, nor duplicate transitions, nor isolated transitions.

The idea of the synthesis [Desel and Reisig, 1996] is to compute regions from the DFA, and associate a place to each of these regions.

Definition 1.19 (Region)**.** *A region of a DFA is a subset of its states $r \subset S$ such that each event $e \in E$ verifies exactly one of the three mutually exclusive conditions:*

- *e enters r, i.e. $\forall (s, s'), s \xrightarrow{e} s' \Rightarrow (s \notin r \wedge s' \in r)$*
- *e exits r, i.e. $\forall (s, s'), s \xrightarrow{e} s' \Rightarrow (s \in r \wedge s' \notin r)$*
- *e crosses no border,i.e. $\forall (s, s'), s \xrightarrow{e} s' \Rightarrow (s \in r \Leftrightarrow s' \in r)$*

Figure 1.17 illustrates this notion: $r = \{s1, s2, s3\}$ is a region; b enters r, c exits r, and a, d do not cross the border. Verifying if a subset of states is a region can be done in polynomial time, but finding all regions in a given graph is NP-complete. Once a region is found, it is associated to a place whose pre-transitions are the events entering the region, and post-transitions the events exiting the region.

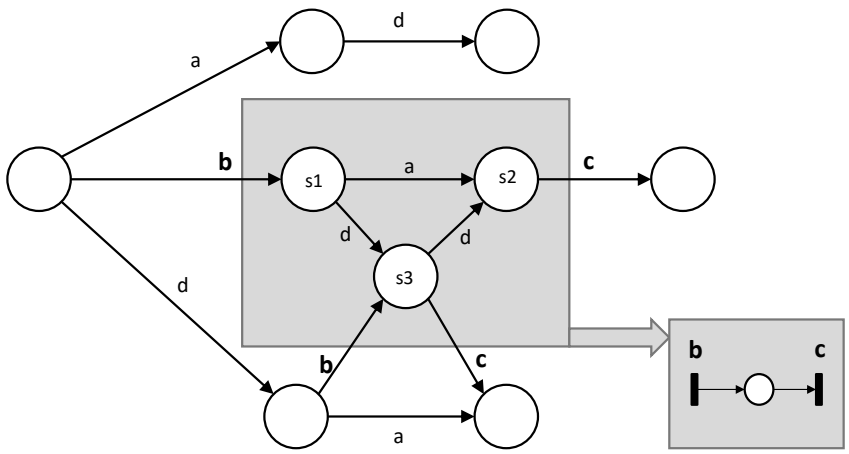


Figure 1.17: Illustration of a region, and its elementary net equivalent

Resolution by Integer Linear Programming

While region theory takes a transition system as input, the studies presented in this section directly consider languages.

Given a finite, prefix-closed language L, and introducing k as the maximal length of the strings in L, the authors of [Giua and Seatzu, 2005] consider a synthesis problem: building a PN (N, M_0) whose generated language up to length k is exactly L ($L_k(N, M_0) = L$). The method proposed by the authors consists in solving an Integer Linear Program (ILP) to find the incidence matrices (Pre,$Post$) and the initial marking(M_0) of the net; the number of places m is fixed, and the net being free-labelled, the n transitions are already known. The set of positive samples and inferred counter-examples are respectively $\mathcal{E}$ and $\mathcal{D}$:

$$\mathcal{E} = \{(\sigma, t_j) | \sigma \in L, |\sigma| < k, \sigma t_j \in L\}$$

$$\mathcal{D} = \{(\sigma, t_j) | \sigma \in L, |\sigma| < k, \sigma t_j \notin L\}$$

A linear characterization is then provided:

$$\mathcal{G}(\mathcal{E}, \mathcal{D}) \triangleq \begin{cases} M_0 + Post.\overrightarrow{\sigma} - Pre.(\overrightarrow{\sigma} + \overrightarrow{\varepsilon_j}) \geq \overrightarrow{0}, \forall(\sigma, t_j) \in \mathcal{E} \\ -KS_{\sigma,j} + M_0 + Post.\overrightarrow{\sigma} - Pre.(\overrightarrow{\sigma} + \overrightarrow{\varepsilon_j}) \leq -\overrightarrow{1_m}, \forall(\sigma, t_j) \in \mathcal{D} \\ {}^t\overrightarrow{1} S_{\sigma,j} \leq m - 1, \forall(\sigma, t_j) \in \mathcal{D} \\ M_0 \in \mathbb{N}^m \\ Pre, Post \in \mathbb{N}^{m \times n} \\ S_{\sigma,j} \in \{0, 1\}^m \end{cases}$$

And the solution can be computed by solving the following ILP, where $f(M_0, Pre, Post)$ is a cost function to minimize. For instance, ${}^t\overrightarrow{1_m}.M_0 + {}^t\overrightarrow{1_m}.(Pre + Post)\overrightarrow{1_m}$ to minimize

the sum of tokens and weights of edges.

$$\begin{cases} \min f(M_0, Pre, Post) \\ \qquad\qquad \text{s.t.} \mathcal{G}(\mathcal{E}, \mathcal{D}) \end{cases}$$

For instance, given $L = \{t_1, t_1t_1, t_1t_2, t_1t_1t_2, t_1t_2t_1\}$, the identified net is presented in Figure 1.18(a). By adding a marking constraint $m_1 + m_2 = \text{const}$, the net of Figure 1.18(b) is obtained. Notice that the second net verifies $L_k(N, M_0) = L$, but possesses exceeding language since $L_{k+1}(N, M_0) \neq \emptyset$.

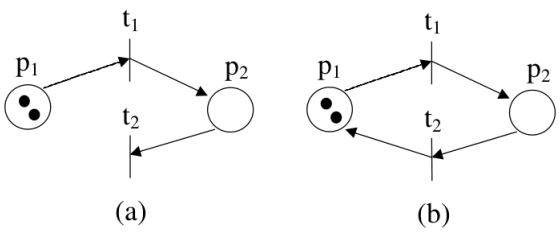


Figure 1.18: [Giua and Seatzu, 2005] Two free-labelled, generalized nets identified from solving ILP

The knowledge of the number of places is reduced to the knowledge of an upper bound in [Cabasino et al., 2007], and an extension of the procedure to λ-free nets is also proposed (*i.e.* nets whose events can label multiple transitions). In these approaches, the language is assumed to be completely known, making the construction and use of counterexamples $\mathcal{D}$ possible. These counterexamples are not available in our hypotheses due to the incompleteness of the observation, as shown in Section 1.2.2. Besides, inputs or outputs are unrelated to events.

1.3.3.2 Building PN from incomplete observation

In this section, different methods using one (a set of) observed sequence(s), and assuming incomplete observation, are reviewed. A set of sequences can be interpreted as an observed language L, the aim being then to build a net N such that $L \subseteq L(N)$.

Resolution by Integer Linear Programming

This principle, firstly used in [Giua and Seatzu, 2005] with complete languages, has then been used in numerous works.

Instead of considering full knowledge of the language in the beginning of the procedure, the authors of [Dotoli et al., 2008] propose an incremental approach, where an ILP is solved after each new observation. An observation consists in recording an event and the markings of places, so knowledge on the structure is already accessible. A new λ-free PN is computed after each observation. However, the longer the observation, the harder it is to solve the ILP due to the increase of the maximal length of the language (the complexity of the resolution being exponential with the number of variables, dependant

on k). Besides, the idea of iteratively computing a model is questionable; whichever the use of the model, it can be computed offline, with access to the whole data record.

Similarly, a theoretical approach of active identification based on ILP resolution is proposed in [Basile et al., 2012]. Events labelling transitions are split into controllable and uncontrollable events, the idea being to send controllable events (*i.e.* fire controllable transitions) and observe the response of the systems in terms of uncontrollable events. The obervation is stopped when cycles are detected, and an ILP is solved thereafter to infer a net structure consistent with the response.

The addition of time has been considered in [Basile et al., 2011] where Deterministic Timed PN are built, the transitions being assigned a fixed firing time value. An extension to time intervals was later proposed in [Basile et al., 2016c], where Timed PN are identified. The knowledge of upper and lower bounds on the time intervals helps designing timed counterexamples to include in the ILP. An extension to labelled Timed PN was also proposed in [Basile et al., 2016b].

In the presented approaches, PN identification is always coupled to fault diagnosis purposes [Fanti and Seatzu, 2008]. A series of extensions have been considered to add faulty behaviour to an already known nominal PN. These whitebox approaches proceed by adding faulty, silent (ε-labelled) transitions to a nominal model, thus extending the previous results to arbitrary PNs. The authors of [Dotoli et al., 2010] use an incremental approach, and assume the knowledge of the markings; the authors of [Cabasino et al., 2014] assume the knowledge of both the nominal and the faulty behaviours of the system as languages; finally, the authors of [Basile et al., 2015] add time to the identified faulty transitions (extended to labelled Timed PN in [Basile et al., 2016a]). An example from [Cabasino et al., 2014] is presented in Figure 1.19; the nominal language was $L = \{\varepsilon, t_1, t_1t_2, t_1t_2t_3\}$, and the faulty language of length 4 is $L_4^F = L \cup \{t_1t_2t_2, t_1t_2t_2t_2\}$.

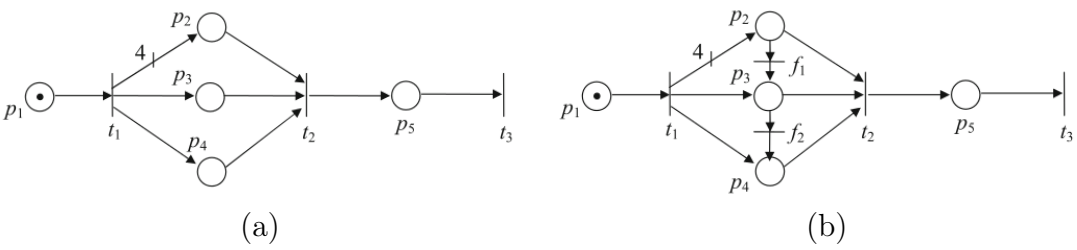


Figure 1.19: [Cabasino et al., 2014](a)Nominal net; (b) Faulty net with two silent transitions

Identification of ordinary and stochastic nets

A theoretical approach to build a PN from a sequence of events is proposed in [Ould El Medhi et al., 2006]. The events are not related to a real system. The idea is to build a direct event propagation matrix B from an observed sequence of events, such that $B_{ij} = 1$ iff e_i and e_j were observed consecutively in that order in the sequence. For instance, given $Seq = e_1e_2e_1e_3e_4e_1$, the matrix B of Figure 1.20 is built:

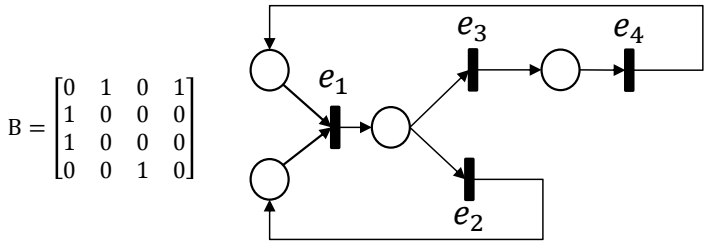


Figure 1.20: The matrix B and the Petri net computed from Seq

Then, for each column j of B, a place p_j is created such that ${}^\bullet p_j = e_j$ and $B_{ij} \neq 0 \Rightarrow e_i \in p_j^\bullet$. The net of Figure 1.20 is obtained by this construction. Additional rules are given in [Lefebvre and Leclerq, 2011] to merge places with the same output, duplicate transitions and compute an initial marking. A learning algorithm based on Neural Networks was also proposed in [Ould El Medhi et al., 2006] to find PNs with a minimal number of places.

An inherent hypothesis of this work is that all causalities, as in succession of related events, were observed. Besides, the use of only sequential relationships forbids the discovery of concurrency and of long-term dependencies. The proposed method was extended to stochastic models in [Lefebvre and Leclerq, 2011], in a context of fault diagnosis; faulty transitions are identified and associated with exponential time distributions, whereas normal distributions are associated to nominal transitions.

The sequentiality rule used in the approach building the nominal model is close to rules defined and used in the α-alogrithm [Van der Aalst et al., 2004], which is the pioneer algorithm of the field of Process Mining, detailled thereafter.

Process Mining approaches

Huge business processes generate flows and flows of data, which are recorded in databases of *event logs*. *More and more events are recorded providing detailed information about the history of processes. Despite the omnipresence of event data, most organizations diagnose problems based on fiction rather than facts . Process mining is an emerging discipline providing comprehensive sets of tools to provide fact-based insights and to support process improvements. This new discipline builds on process model-driven approaches and data mining.*([Van der Aalst, 2013a]). Unlike real systems, the logs are passive recordings of activities, exhibiting absolutely no reactive behaviour.

As presented in Figure 1.21, Process mining approaches are used for three main applications. *Discovery* methods produce models from event logs without *apriori* information. *Conformance* methods compare an event log to an existing process model, to detect, locate and explain deviations. Finally, *enhancement* consists in improving existing models by using information recorded in an event log. While conformance issues would be related to fault diagnosis for DES, enhancement and discovery can respectively be seen as whitebox and blackbox identification approaches.

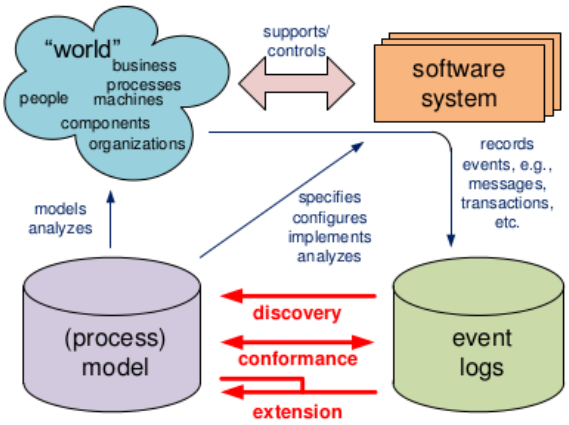


Figure 1.21: [Van der Aalst, 2013a] Positioning of the three main types of process mining: *discovery, conformance and enhancement*

Notable examples of business processes are insurance agencies, banks, hospitals, manufacturers, government agencies, municipalities, ... The interested reader is refered to the reference book [Van der Aalst, 2011b] for further information. In this section, only some discovery methods are presented.

The first algorithm introduced for process (or workflow) mining is the α algorithm [Van der Aalst et al., 2004]. To model the processes, the specific subclass of workflow nets is considered:

Definition 1.20 (Workflow nets)**.** *A workflow net is a Petri net $N = (P, T, W)$ with the following three additional properties:*

1. *Object creation: P contains a place i such that ${}^\bullet i = \emptyset$*
2. *Object completion: P contains a place o such that $o^\bullet = \emptyset$*
3. *$\hat{N} = (P, T \cup \hat{t}, W)$, where ${}^\bullet\hat{t} = o$ and $\hat{t}^\bullet = i$ is strongly connected*

Initially, only the input place is marked. This definition is extended to *sound* workflow nets (SW-nets):

Definition 1.21 (Sound Workflow nets)**.** *A workflow net $N = (P, T, W)$ is sound iff the following properties are verified:*

1. *N is 1-bounded*
2. *Proper completion: if o is marked, then it is the only place marked*
3. *Option to complete: the marking of the output place is always reachable from any reachable marking*
4. *No dead transitions*

The idea of the algorithm is to discover relationships between the activities recorded in the log, based on their order in the log.

Definition 1.22 (Relationships between activities)**.** *Consider two activities A and B. :*

1. *B directly follows A, written $A > B$, iff AB is observed in the log*
2. *A directly causes B, written $A \rightarrow B$, iff $A > B$ and $B \not> A$*
3. *A and B are indifferent, written $A \# B$, iff $A \not> B$ and $B \not> A$*
4. *A and B are parallel, written $A \parallel B$, iff $A > B$ and $B > A$*

For instance, consider the log $\mathcal{L} = \{ABCD, ACBD, AED\}$, on the activity set $T = \{A, B, C, D, E\}$. It comes A $\rightarrow$ B, A $\rightarrow$ C, A $\rightarrow$ E, B $\rightarrow$ D, C $\rightarrow$ D, E $\rightarrow$ D, and B $\parallel$ C. The definition of the parallel relationship implies completeness of the log, since two parallel activities must have been sequentially observed in both orders.

The α-algorithm extracts these relations from the log, and computes a set X of relationships between activities sets:

$$X = \{(\mathcal{A}, \mathcal{B}) \in 2^T \times 2^T | \forall_{A \in \mathcal{A}}, \forall_{B \in \mathcal{B}}, A \rightarrow B \wedge \forall_{A_1, A_2 \in \mathcal{A}^2}, A_1 \# A_2 \wedge \forall_{B_1, B_2 \in \mathcal{B}^2}, B_1 \# B_2\}$$

The construction of X runs in exponential time with the number of activities. Then, the set Y of the maximal relationships from X is computed, and is translated into places. The application of α to L leads to the following construction:

$T = \{A, B, C, D, E\}$ the set of activities
$T_i = \{A\}$ the set of initial activities
$T_o = \{D\}$ the set of final activities
$X = \{(\{A\}, \{B\}), (\{A\}, \{C\}), (\{A\}, \{E\}), (\{B\}, \{D\}),$
$(\{C\}, \{D\}), (\{E\}, \{D\}), (\{A\}, \{B, E\}), (\{A\}, \{C, E\}),$
$(\{B, E\}, \{D\}), (\{C, E\}, \{D\})\}$ the set of relationships
$Y = \{(\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}), (\{C, E\}, \{D\})\}$ the maximal set
$P = \{i, o, p(\{A\}, \{B, E\}), p(\{A\}, \{C, E\}),$
$p(\{B, E\}, \{D\}), p(\{C, E\}, \{D\})\}$ the set of places
$W = \{(i, A), (A, p(\{A\}, \{B, E\})), (p(\{A\}, \{B, E\}), B), \dots\}$ the set of edges

The resulting net is presented in Figure 1.22.

Numerous extensions to α have been proposed through the years, some being summarized in [Van Dongen et al., 2009]. Like α, some rule-based approaches have been developped to deal with different classes of nets (for instance non free-choice nets with α++ [Wen et al., 2007] or T-invariants discovery [Tapia-Flores et al., 2014], block-structured workflow nets [Leemans et al., 2013], ...). Other approaches took inspiration in languages, proposing learning [Esparza et al., 2010], region-based [Bergenthum et al., 2007], or partial-order based [Lorenz et al., 2007] algorithms. Heuristics have also been proposed [Van der Aalst et al., 2005], as well as probabilistic models to deal with incomplete logs ([Van Hee et al., 2011],[Leemans et al., 2014]).

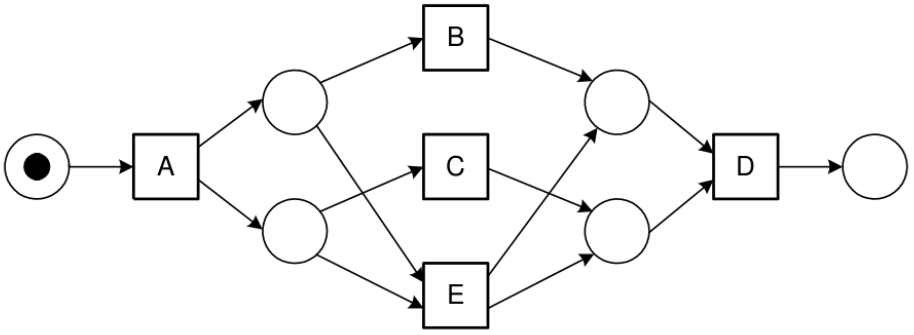


Figure 1.22: [Van der Aalst et al., 2004] The net mined by the α-algorithm from the log $\{ABCD, ACBD, AED\}$

Most, if not all of these algorithms, are available in ProM[1], an open-source framework for implementing process mining tools.

Finally, this tour of PN identification is concluded by the section closest to the hypotheses set in Section 1.2, where IPNs are identified from input/output sequences.

Identification of Interpreted Petri Nets

Historically, the first approach identifying Interpreted Petri Nets from a sequence is the one presented in [Meda-Campana, 2002]. The procedure proceeds in two steps ([Meda et al., 1998] and [Meda-Campana and Lopez-Mellado, 2001]): first it computes the observable places, such that there exists exactly one place for each output signal; then the non observable places of the net are inferred from T-semiflows. A T-semiflow is a firing sequence σ such that $M \xrightarrow{\sigma} M$. Considering that only some places are observable, the notion of T-semiflow is reduced to m-words. A m-word is a firing sequence σ such that $M_{Obs} \xrightarrow{\sigma} M_{Obs}$, where M_{Obs} is the restriction of the marking to observable places.

To understand the method, it is illustrated on the following example extracted from [Meda-Campana and Lopez-Mellado, 2001]. The system considered has 7 outputs, and the first observed output vectors are:

$$w = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\end{bmatrix}\dots$$

The idea is to discover cycles within w. The approach is incremental, as w will be read, and the net updated every time a new cycle is discovered. Here, for the first cycle

[1] http://www.processmining.org/prom/start

detected:

$$w(1) = w(3) = {}^{t}\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix} \qquad \text{Cycle discovered}$$

$$^{\bullet}p_1(o_1) = t_1, p_1(o_1)^{\bullet} = t_2 \qquad \text{Building transitions}$$

$$m_1 = t_1 t_2 \qquad \text{A m-word is computed}$$

$$W_1 = m_1 \qquad \text{A T-semiflow is inferred}$$

The resulting net is presented in Figure 1.23(a). Then, a new cycle is discovered from $w(3) = w(5)$, a new m-word $m_2 = t_3 t_4$ is computed, and the T-semiflow W_1 is extended to $W_1 = m_1 m_2$; the resulting net can be seen in Figure 1.23(b). Same for m-word $m_3 = t_7 t_{11}$, which leads to the net in Figure 1.23(c). The observation of m-word $m_4 = t_8 t_9 t_{10} t_{13}$ leads to $W_1 = m_1 m_2 m_3 m_4$ and the net of Figure 1.23(d). m_1 is then observed again, then $m_5 = t_5 t_6$. A new T-semiflow $W_5 = m_1 m_5$ is inferred, leading to the net of Figure 1.23(e). Finally, the observation of $m_6 = t_7 t_8 t_9 t_{11}$ leads to modifications of the net visible in Figure 1.23(f): dependency $[t_6, t_7]$, concurrency of t_8 and t_{11}. The rules behind the modifications are detailed in [Meda-Campana and Lopez-Mellado, 2001].

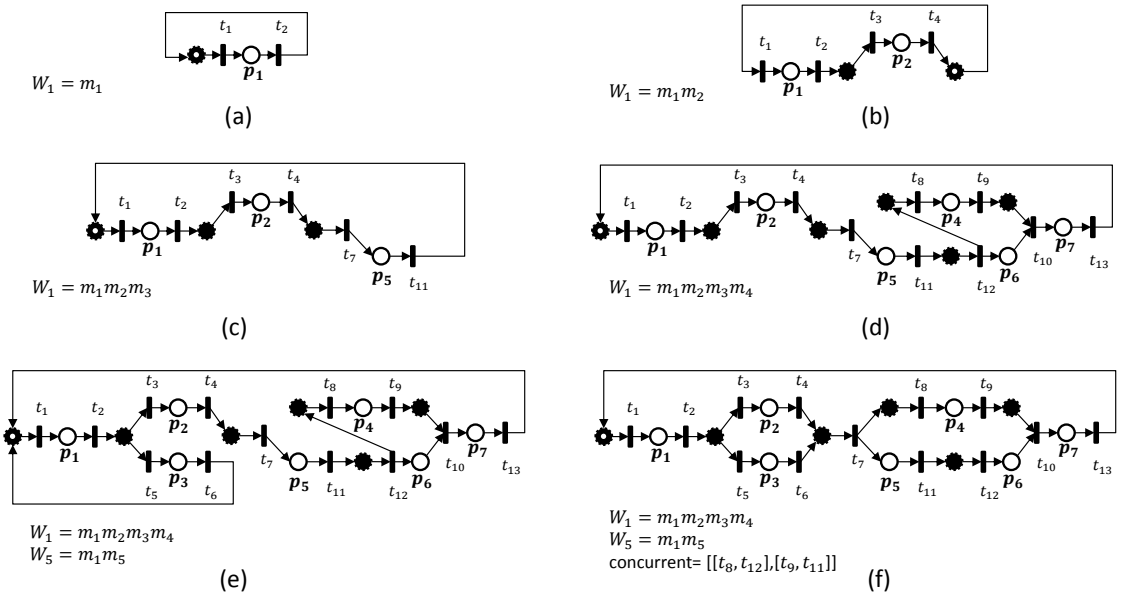


Figure 1.23: [Meda-Campana and Lopez-Mellado, 2001] 6 nets incrementally computed from the observed output sequences

Notice that the approach only considers output signals. The function λ associating input conditions to transitions is not computed, so this approach is not sufficient enough to express the whole behaviour of a reactive DES. However, the identification algorithm runs in polynomial time with the length and the number of observed m-words, which is a good quality.

A similar approach is presented in [Estrada-Vargas et al., 2014], where inputs and outputs are both considered. This approach computes first the unobservable places by

inferring input event cycles, then adds the observable places. It is illustrated by the following example: a system with three binary inputs $\{a, b, c\}$ and four binary outputs $\{A, B, C, D\}$. The system is represented by the vector ${}^t\begin{bmatrix} a & b & c & | & A & B & C & D \end{bmatrix}$, and the observed vector sequences are:

$$w_1 = \begin{bmatrix} 0 \\ 0 \\ \frac{0}{1} \\ 0 \\ 0 \\ 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ \frac{0}{0} \\ 0 \\ 0 \\ 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ \frac{0}{0} \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad w_2 = \begin{bmatrix} 0 \\ 0 \\ \frac{0}{1} \\ 0 \\ 0 \\ 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ \frac{0}{0} \\ 0 \\ 0 \\ 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ \frac{1}{0} \\ 0 \\ 1 \\ 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ \frac{1}{0} \\ 0 \\ 1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ \frac{0}{0} \\ 0 \\ 0 \\ 1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ \frac{0}{1} \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad w_3 = \begin{bmatrix} 0 \\ 0 \\ \frac{0}{1} \\ 0 \\ 0 \\ 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ \frac{0}{0} \\ 0 \\ 0 \\ 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ \frac{1}{0} \\ 0 \\ 1 \\ 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ \frac{0}{0} \\ 0 \\ 0 \\ 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ \frac{0}{0} \\ 0 \\ 0 \\ 1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ \frac{0}{1} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Notice that all sequences are supposed to start from the same vector. Then, only the inputs are considered for the events building, From w_1 is computed an event sequence $\tau_1 = e_1e_2$, such that $\lambda(e_1) = a_1$ and $\lambda(e_2) = \varepsilon$. Indeed, only the rising edge of a was observed on the inputs between $w_1(1)$ and $w_1(2)$, and no input event between $w_1(2)$ and $w_1(3)$. The three observed vector sequences become:

$$\tau_1 = e_1e_2$$
$$\tau_2 = e_1e_3e_4e_5e_6$$
$$\tau_3 = e_1e_3e_5e_4e_6$$

with

$$\lambda(e_1) = a_1,\ \lambda(e_2) = \varepsilon,\ \lambda(e_3) = b_1c_1,\ \lambda(e_4) = b_0,\ \lambda(e_5) = c_0,\ \lambda(e_6) = a_0$$

A parameter κ homogoneous to the parameter k of [Klein, 2005] is introduced (the works also share the hypothesis of sequences starting from the same vector). Similarly, κ determines the length of substrings to be considered, and is a parameter to be tuned regarding the completeness of the observation, for accuracy purposes (see Section 1.3.2.1). The substrings of length κ are here associated to transitions instead of states.

For $\kappa = 2$, the following substrings are computed:

$$\tau_1^2 = \varepsilon e_1, e_1e_2$$
$$\tau_2^2 = \varepsilon e_1, e_1e_3, e_3e_4, e_4e_5, e_5e_6$$
$$\tau_3^2 = \varepsilon e_1, e_1e_3, e_3e_5, e_5e_4, e_4e_6$$

From these substrings is computed the net of Figure 1.24(a). Notice that two transitions have been created for events e_4, e_5 and e_6. Since t_6 and t_9 are labelled by the same event and share the same future (post-place), they are merged (Figure 1.24(b)). The Petri Net looks heavily like an automaton, the concurrency between e_4 and e_5 being separated. Since the substrings e_4e_5 and e_5e_4 have both been observed, concurrency is inferred in the next step, leading to the net of Figure 1.24(c).

Finally, interpretation is added by replacing the events by the labelling function λ, adding the observable places and removing the implicit unobservable places. The

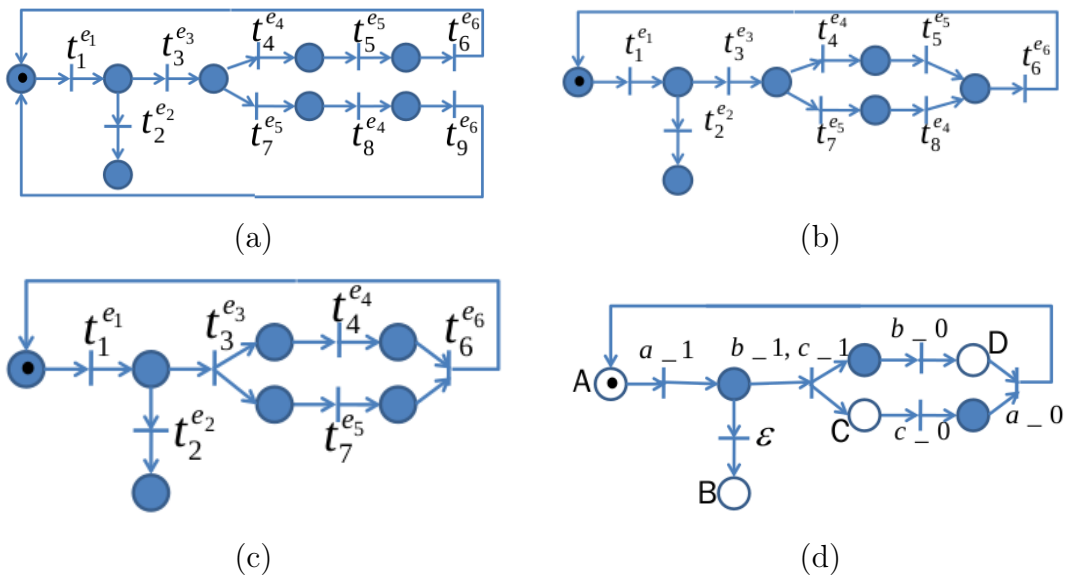


Figure 1.24: [Estrada-Vargas et al., 2014](a) Basic identified model; (b) Model after merging; (c) Model after concurrency simplification; (d) IPN model with observable places

final result is the net of Figure 1.24(d). The construction of the net, except for the concurrency simplification, runs in polynomial time with the length and number of observed sequences. However, to conclude that n transitions are conccurent, and procede to a simplification, it must be checked that $n!$ paths representing all permutations are present. Therefore, exhibiting concurrency is hard (exponential complexity), and the condition is often not verified (due to the incompleteness of the observed data). The IPNs offer a good reading of the reactive behaviour, but are most often state-machines and do not compactly represent concurrency.

The discovery and explicit representation of concurrency is often troublesome. Some recent works identifying IPNs in the context of closed-loop reactive systems suffer from the same problem.

Namely, the authors of [Ladiges et al., 2015] use exactly the same method as the authors of [Lefebvre and Leclerq, 2011] to compute first the unobservable places of the net, called machine behaviour. Then, the signal part, *i.e.* the observable places, is added, and consists in a couple of places corresponding respectively to the high and low levels of a sensor. The resulting net is called Machine-State Petri Net. An example of a MSPN is given in Figure 1.25.

The approach has the same drawbacks as [Lefebvre and Leclerq, 2011], such as the lack of explicit concurrency in the model. Besides, there is also no distinction between input and output signals, as transition are labelled with signal events related to events associated to observable places.

In an approach very similar to [Estrada-Vargas et al., 2014], an incremental IPN identification approach is proposed in [Munoz et al., 2014], with an extension to stochastic-time IPNs. The algorithm identifying the structure is essentially the same, without the

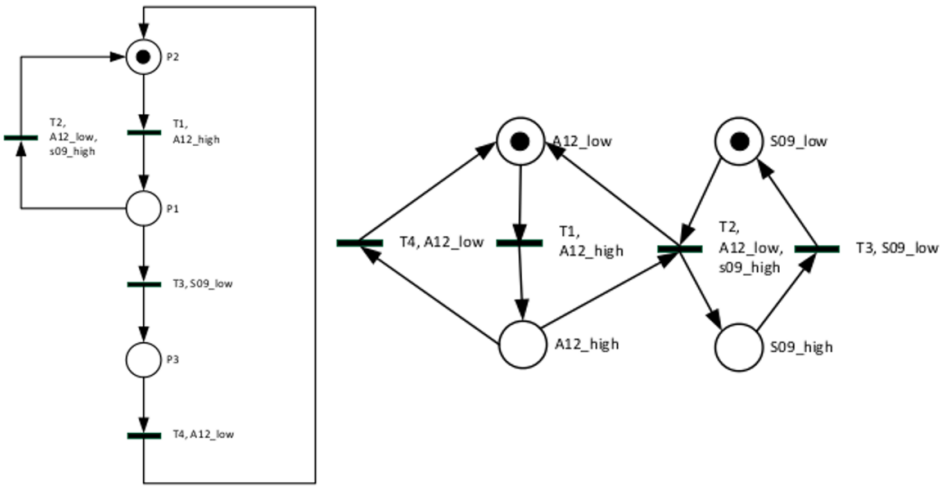


Figure 1.25: Left: Unobservable part (machine behaviour). Right: Observable part (signals)

steps of model simplification. Namely, the PNs identified are state-machines, and no procedure is proposed to deal with concurrency. Besides, the notion of observable places is missing; each place is assigned an output vector instead, degrading the understandability.

Finally, [Estrada-Vargas et al., 2015] proposed a method computing firstly compact observable models, and then inferring unobservable places. The models are easily readable and explicit the reactive behaviour and concurrency. Since the contributions of this thesis are based on the method proposed in this work, it is extensively detailed in Chapter 2.

1.4 Conclusions and positioning

The identification problem stated in Section 1.2 deals with closed-loop logical systems. The technology of the PLC makes the observation of simultaneous input and output events possible, meaning that events are replaced by event vectors. Furthermore, counter-examples are unavailable when a real system is observed, namely when the system observed is highly concurrent. These hypotheses differ from those made in most of the related works. Namely, some language, region based or synthesis methods are impossible to use in our context due to the hypothesis of incompleteness and the lack of counterexamples; active or whitebox approaches are also set aside.

Regarding the resulting model, finite automata lack the semantics to represent concurrency, even in the form of Mealy or Moore machines. To model a non deterministic system, which exhibits concurrency, Petri Nets have the adapted semantics.

Process mining approaches are restrictive in the subclass of nets considered (fixed starting and ending activities), and do not consider inputs and outputs, therefore are inadapted to represent reactive behaviour. Using Interpreted Petri Nets is however appropriate to model the reactive behaviour, as the input conditions added to the tran-

sitions are expressively the cause of the outputs associated to places. Table 1.4 resumes the similarity between IPNs and the reactive closed-loop systems which are of interest in this thesis.

	Closed-loop DES	IPN
State variables	$\mathbb{X}$	P
State of cycle k	$X(k)$	M_k
State evolution function	$X(k+1) = g(U(k), X(k))$	$M_{k+1} = M_k + C.t_i$ if $M_k(P_j, t_i) \geq I(P_j, t_i) \forall P_j \in P$ and $\lambda_{t_j}(k) = 1$
Output function	$Y(k) = g(U(k), X(k))$	$Y(k) = \varphi(M_k)$

Table 1.1: Similarity between closed-loop systems and IPN

The chosen model class is introduced in the problem of this thesis:

(Reformulated) Problem Statement *Consider a logical closed-loop system controlled by a PLC, under the hypotheses of Section 1.2. Identify, from an observed vector sequence w,* ***an IPN*** *that exhibits the reactive behaviour and reproduces w.*

An approach satisfying the problem must be appropriate for blackbox, passive identification, and take into account the logical I/Os, the technology of the PLC (cyclicity and synchronization), and can not use counter-examples due to the incompleteness of the observation. A previous method was developed in [Estrada-Vargas et al., 2015], where all these hypotheses are respected, and the resulting models are satisfyingly compact and expressive for reverse-engineering. The present thesis is based on these results; they are therefore resumed in next chapter. Some challenges regarding scalability or genericity of the approach will be pointed out, leading to the development of the contributions of this thesis in the following chapters.

Chapter 2

Blackbox behavioural identification of a reactive automated system

Introduction

The problem of identification of a reactive DES has been formulated in section 1.2. Previous inquiries([Estrada-Vargas et al., 2015]) have led to the design of a method in two steps to build an Interpreted Petri Net from the observation of the system during its functioning. The aim of this chapter is to recall and illustrate this method, not only to show its relevance, but also to point out difficulties that will be alleviated by the contributions of this thesis. Section 2.1 presents the motivation for the design of the method, based on the observed vector sequence. Section 2.2 presents the computation of the observable behaviour, and Section 2.3 the rule-based inference of the unobservable behaviour.

2.1 Two behaviours: Observable and Unobservable

2.1.1 Event types

A system with m logical inputs and n logical outputs is considered. The starting point of the method is a sequence of observed vectors w. $\forall k, w(k) \in \{0,1\}^{n+m,1}$, each row of w representing the level of the input/output at the end of the PLC cycle k. Formally,

$$w(k) = \begin{bmatrix} U(k) \\ - \\ Y(k) \end{bmatrix} \text{ with } U(k) = \begin{bmatrix} u_1(k) \\ u_2(k) \\ \dots \\ u_m(k) \end{bmatrix} \text{ and } Y(k) = \begin{bmatrix} y_1(k) \\ y_2(k) \\ \dots \\ y_n(k) \end{bmatrix}$$

Then, the event vector sequence E is computed from w as $\forall k, E(k) = w(k+1) - w(k)$ (Section 1.2.1). Similarly, $\forall k, E(k) \in \{-1,0,1\}^{n+m,1}$, each row of E representing events associated to inputs and outputs. The rows of E are split into elementary input events

$IE_{\{1...m\}}$ and elementary output events $OE_{\{1...n\}}$. Formally,

$$E(k) = \begin{bmatrix} IE(k) \\ - \\ OE(k) \end{bmatrix} \text{ with } IE(k) = \begin{bmatrix} IE_1(k) \\ IE_2(k) \\ \dots \\ IE_m(k) \end{bmatrix} \text{ and } OE(k) = \begin{bmatrix} OE_1(k) \\ OE_2(k) \\ \dots \\ OE_n(k) \end{bmatrix}$$

For each input $u_i \in \mathbb{U}$, two elementary input events exist: its rising edge noted u_i_1 or $\uparrow u_i$, and its falling edge noted u_i_0 or $\downarrow u_i$. Given k, a 1-cell of IE(k) corresponds to the rising edge of the corresponding input ($IE_i(k) = 1 \Leftrightarrow u_i_1$), whereas a (-1)-cell corresponds to its falling edge($IE_i(k) = -1 \Leftrightarrow u_i_0$). A null cell means that the input has not changed between the two successive observations $w(k+1)$ and $w(k)$, *i.e.* between the two cycles of the controller. The same definitions stand for output events $OE_i(k)$ and outputs $y_i \in \mathbb{Y}$. Recall that multiple cells can be non-zero in each E(k), due to the technology.

Depending on the values of $IE(k)$ and $OE(k)$, there are four different possibilities for $E(k)$, which can be interpreted according to the technology of the PLC:

Type1. $IE(k) \neq 0 \wedge OE(k) \neq 0$ At least one input change has probably directly caused at least one output change. This reactive behaviour is observed in the same PLC cycle.

Type2. $IE(k) = 0 \wedge OE(k) \neq 0$ An output change is observed despite the lack of input change. The controller has reached a state $X(k)$, where the current input levels enabled the output change.

Type3. $IE(k) \neq 0 \wedge OE(k) = 0$ Two possibilities:

- $a) X(k) \neq X(k-1)$. The input change has provoked a non-observable change in the state variables.
- $b) X(k) = X(k-1)$. The input change is irrelevant for the controller in its current state.

Type4. $IE(k) = 0 \wedge OE(k) = 0$ Two possibilities:

- $a) X(k) \neq X(k-1)$. The controller follows autonomously a state trajectory without input or output changes
- $b) X(k) = X(k-1)$. The controller remains in a stable state.

Type1 is the behaviour closest to DES theory: an input event has caused an output event. The causality extracted is associated as an event to a transition in the IPN.

Type2 represents on the other hand causalities of input levels on outputs. The input event that put the controller in the state trajectory provoking the output event occurred

some cycles ago. The level of the causal input however remains until the output change. This causality is associated as a level condition associated to a transition.

Type3.a is an input change that changed at least one of the state variables of the controller, typically counters or memories. However, these variables being unknown, it is impossible to compute precisely such an internal evolution. By studying the context of the input firing, it might however be possible to infer the state evolution. In this case, since no output event occurred, unobservable places represent the states reached. Type4.a represent also internal state evolutions, without input changes, such as timers.

Finally, there is nothing to be guessed for Types3.b and 4.b, since nothing happened in the controlled. Furthermore, Type4 evolutions are not even recorded, since nothing was observed on the I/O signals ($w(k+1) = w(k)$, or equivalently $E(k) = 0$).

Type1 and 2 represent observable behaviour; whereas types3a and 4a represent unobservable behaviour. The first one is expressed by observable transitions, labelled by the event and conditions on the inputs, connected to observable places, labelled by the outputs. The second one is represented by unobservable places, agregating the evolutions of unknown state variables, and connecting the transitions.

2.1.2 Framework of the method

From the distinction between observable and unobservable behaviour pointed in the last section, the proposed method consists firstly in building the whole observable behaviour, then to infer unobservable state evolutions. The framework is presented in Figure 2.1. In short:

- The sequence w is built from one (or multiple) observation(s) of the system, long enough to have observed some production cycles. As presented in Section 1.2.2, the observation remains however incomplete, and it is not required to observe all possible interleavings of concurrent subprocesses. (First box of Figure 2.1)

- The observable parts of the PN, *i.e.* transitions and observable places, are built. Firstly, the sequence w is analyzed to build causality matrices, from which the input causes of output events are extracted, and formalized as output firing functions. Then, one observable place is created for each output, and transitions are built to represent the output events; they are labelled with the consistent output firing functions. At the end of this step, all transitions and observable places are computed. Finally, the sequence w is projected on the just computed transitions, and translated into a firing sequence S. (Second box of Figure 2.1, detailed in Section 2.2)

- It remains to infer and add the unobservable places of the identified PN, such that the sequence S becomes firable. Causal and concurrency relations are defined on transitions, and decided according to rules. These relations are associated to unobservable PN fragments, that are added to the net. The structure is then

verified by the token flow, and corrected if required. (Third box of Figure 2.1, detailed in Section 2.3)

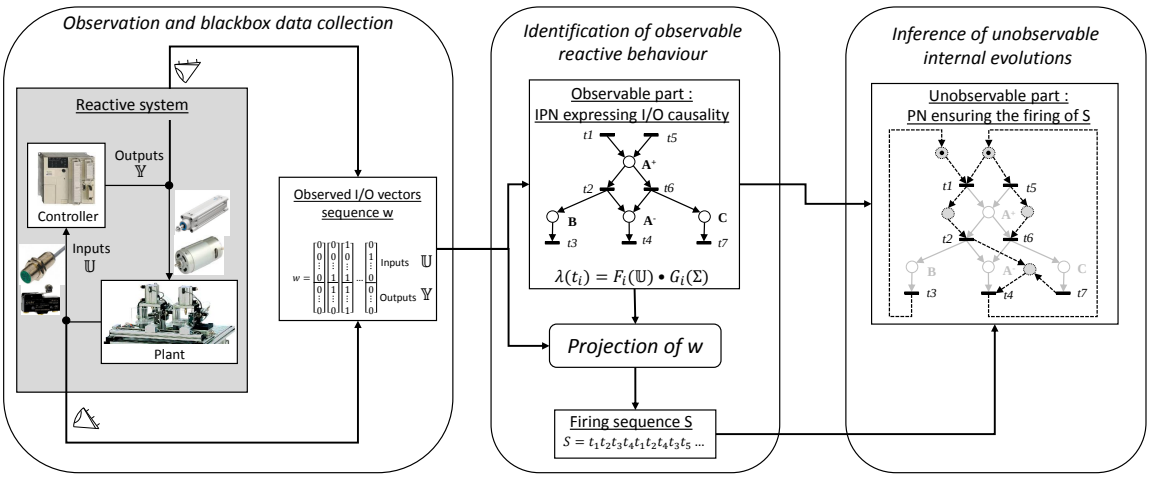


Figure 2.1: Data collection, then construction of an IPN in two steps

Before entering into details of each step, an example is presented, on which each step of the method will be applied.

2.1.3 Illustrative example: Sorting system

This example is extracted from [Estrada-Vargas et al., 2015]. The considered system consists in 4 outputs and 9 inputs, and is designed to sort parcels depending on their size (Figure 2.2). When a parcel arrives on conveyor 1, its height is detected by either k1 (small parcel) or k2 (big parcel). The small (resp. big) parcel is then pushed by the double-acting cylinder A, who can be extended by A+ and retracted by A-, in front of the single-acting cylinder B (resp. C) that pushes it on conveyor 2 (resp. 3). A new parcel may arrive and be detected while another one is being treated by B or C. The inputs a0,a1,a2 are three proximity sensors detecting the position of cylinder A. The same goes for inputs b0,b1 (resp. c0,c1) and cylinder B (resp. C).

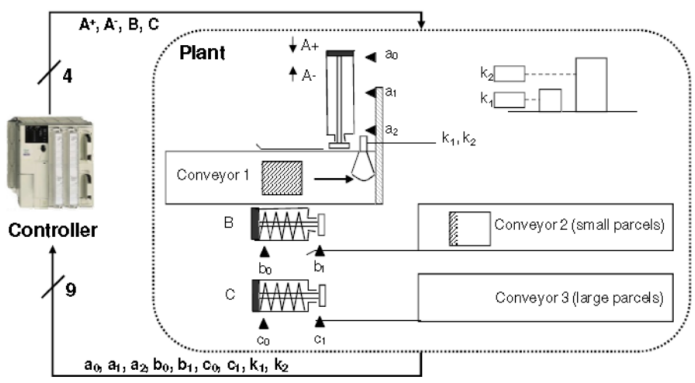


Figure 2.2: Illustrative example: a package sorting system

The observation was conducted during the treatment of 20 parcels, the first vectors of the observed sequence w are shown below, corresponding to the arrival and sorting of a small parcel, as well as the first vectors of the event sequence E computed from w. The elementary events deduced from E are presented in Table 2.1.

$$w = \begin{array}{c} \\ k_1 \\ k_2 \\ a_0 \\ a_1 \\ a_2 \\ b_0 \\ b_1 \\ c_0 \\ c_1 \\ A+ \\ A- \\ B \\ C \end{array} \begin{array}{c} \begin{array}{cccccccc} w(1) & w(2) & w(3) & w(4) & w(5) & w(6) & w(7) & w(8) \end{array} \\ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \end{array} \quad E = \begin{array}{c} \\ k_1 \\ k_2 \\ a_0 \\ a_1 \\ a_2 \\ b_0 \\ b_1 \\ c_0 \\ c_1 \\ A+ \\ A- \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccccccc} E(1) & E(2) & E(3) & E(4) & E(5) & E(6) & E(7) \end{array} \\ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \end{bmatrix} \end{array}$$

2.2 Identification of the observable behaviour

2.2.1 Building output firing functions

The first step in the building of the observable behaviour consists in finding direct causalities linking inputs and output events, which are the conditions and events who label the transitions in the identified IPN.

Each occurence of an elementary output event OE_j was caused either by one or several input events occurring in the same PLC cycle (Type1), and by a condition expressed on the levels of the inputs observed in the same cycle (Type2). The firing function of the output event is defined accordingly by combining the two origins:

Definition 2.1 (Output Event Firing Function (OEFF))**.** *An output event firing func-*

Event vector	Elementary Input Events	Elementary output events
E(1)	$IE(1) = \uparrow k_1$	$OE(1) = \uparrow (A-)$
E(2)	$IE(2) = \downarrow a_0$	$OE(2) = \varepsilon$
E(3)	$IE(3) = \downarrow k_1$	$OE(3) = \varepsilon$
E(4)	$IE(4) = \uparrow a_1$	$OE(4) = \downarrow (A-) \wedge \uparrow (A+) \wedge \uparrow B$
E(5)	$IE(5) = \downarrow b_0$	$OE(5) = \varepsilon$
E(6)	$IE(6) = \downarrow a_1$	$OE(6) = \varepsilon$
E(7)	$IE(7) = \downarrow b_1$	$OE(7) = \downarrow B$

Table 2.1: Elementary description of the first event vectors

tion χ states necessary and sufficient conditions for the occurence of the output event OE_j. It is defined as:

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

where $G : OE \rightarrow 2^{IE}$ and $F : OE \rightarrow 2^{\mathbb{U}}$ are functions on respectively the input events and the input levels that express the necessary and sufficient conditions to trigger the output event OE_j.

Some inputs are observed in the same cycle as OE_j, but might nevertheless not be a cause; this fact happens often in systems exhibiting massive concurrency. To compute $G(OE_j)$, the causes have to be extracted from all the observations of OE_j. The Direct Causality Matrix (DCM) is introduced for that purpose: it stores frequency informations of all elementary input events that occured simultaneously with OE_j. This information can be interpreted as a probability:

$$DCM_{ij}(IE_i, OE_j) = \frac{\#(OE_j \wedge IE_i)}{\#(OE_j)} = Prob(IE_i|OE_j)$$

where $\#(OE_j \wedge IE_i)$ is the number of PLC cycles in which OE_j and IE_i occur simultaneously (therefore are observed in the same event vector). The value of DCM_{ij} naturally represents the probability that IE_i is a necessary condition of OE_j.

Similarly, the Indirect Context Matrix (ICM) is introducted to store the frequencies of input levels present when OE_i occurred:

$$ICM_{2i,j}(u_i = 1, OE_j) = \frac{\#(OE_j \wedge u_i = 1)}{\#(OE_j)} = Prob(u_i = 1|OE_j)$$
$$ICM_{2i+1,j}(u_i = 0, OE_j) = \frac{\#(OE_j \wedge u_i = 0)}{\#(OE_j)} = Prob(u_i = 0|OE_j)$$

Naturally, $ICM_{2i,j} + ICM_{2i+1,j} = 1$. The resulting matrices for the sorting system are presented in Figure 2.3; there are 8 columns for the 8 elementary output events ($|\mathbb{Y}| = 4$), and 18 rows for the 18 elementary input events in the DCM, 18 possible input levels in the ICM ($|\mathbb{U}| = 9$).

When taking a closer look at the DCM, interesting values can be extracted. Namely, one can see **(1)** that $Prob(a1_1|B_1) = 1$, which means that $a1_1$ is an input event always present when output event B_1 occurred, hence a necessary condition. Notice that no other input event was observed with B_1, hence $a1_1$ is a necessary and sufficient condition, so $G(B_1) = a1_1$. Event conditions can be more complex **(2)**, as $Prob(a1_1|A+_0) + Prob(a2_1|A+_0) = 1$, which means that $A+_0$ is provoked either by $a1_1$ or $a2_1$, so $G(A+_0) = a1_1 \oplus a2_1$.

The generic idea for computing the OEFF χ of an output event OE_j is therefore to seek probabilities equal to 1 in the columns of the matrices, to have the certainty that a necessary condition is discovered. Regarding the event part, $G(OE_j)$ is computed as

	A+_1	A+_0	A-_1	A-_0	B_1	B_0	C_1	C_0
k1_1	0.111	0.111	0.111	0.111	0.000	0.200	0.000	0.000
k1_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
k2_1	0.222	0.000	0.000	0.000	0.000	0.000	0.000	0.000
k2_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
a0_1	0.222	0.000	0.000	1.000	0.000	0.000	0.000	0.000
a0_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
a1_1	0.000	0.444	0.444	0.000	1.000	0.000	0.000	0.000
a1_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
a2_1	0.000	0.556	0.556	0.000	0.000	0.000	1.000	0.000
a2_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
b0_1	0.333	0.000	0.000	0.000	0.000	0.000	0.000	0.000
b0_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
b1_1	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
b1_0	0.000	0.000	0.000	0.111	0.000	0.000	0.000	0.000
c0_1	0.111	0.000	0.000	0.000	0.000	0.000	0.000	0.000
c0_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
c1_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
c1_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

	A+_1	A+_0	A-_1	A-_0	B_1	B_0	C_1	C_0
k1=1	0.444	0.111	0.111	0.333	0.000	0.250	0.200	0.200
k1=0	0.556	0.889	0.889	0.667	1.000	0.750	0.800	0.800
k2=1	0.556	0.000	0.000	0.333	0.000	0.250	0.000	0.200
k2=0	0.444	1.000	1.000	0.667	1.000	0.750	1.000	0.800
a0=1	1.000	0.000	0.000	1.000	0.000	0.500	0.000	0.000
a0=0	0.000	1.000	1.000	0.000	1.000	0.500	1.000	1.000
a1=1	0.000	0.444	0.444	0.000	1.000	0.000	0.000	0.000
a1=0	1.000	0.556	0.556	1.000	0.000	1.000	1.000	1.000
a2=1	0.000	0.556	0.556	0.000	0.000	0.000	1.000	0.000
a2=0	1.000	0.444	0.444	1.000	1.000	1.000	0.000	1.000
b0=1	1.000	1.000	1.000	0.556	0.000	0.000	1.000	1.000
b0=0	0.000	0.000	0.000	0.444	1.000	1.000	0.000	0.000
b1=1	0.000	0.000	0.000	0.111	1.000	1.000	0.000	0.000
b1=0	1.000	1.000	1.000	0.889	0.000	0.000	1.000	1.000
c0=1	1.000	1.000	1.000	0.889	0.000	1.000	1.000	0.000
c0=0	0.000	0.000	0.000	0.111	1.000	0.000	0.000	1.000
c1=1	0.000	0.000	0.000	0.000	1.000	0.000	0.000	1.000
c1=0	1.000	1.000	1.000	1.000	0.000	1.000	1.000	0.000

Figure 2.3: Matrices computed for the sorting system. Left: Direct Causality Matrix, Right: Indirect Context Matrix

a conjunction of event disjunctions:

$$G(OE_j) = \prod_k DisjE_k$$

where $DisjE_k = \bigoplus_i IE_i$ involves elementary input events whose probabilities sum to 1, *i.e.*

$$\forall i, DCM_{ij} \neq 0$$

$$\sum_i DCM_{ij} = 1$$

The hypothesis is made that two distinct input events which can cause the firing of OE_j do not occur together, which explains the exclusive OR in the disjunction. Each disjunction is a necessary condition for the triggering of the output event, and their conjuction becomes a necessary and sufficient condition. This hypothesis was made to avoid using the inclusion-exclusion principle, and therefore having to count all possible simultaneous input event occurrences. This hypothesis is realistic when considering real systems. The only case observed on a real system implies an input dedicated to safety: the cause of the output event was either a functional input event, or a safety input event (like an emergency stop button). It can occur that the safety is triggered simultaneously with the functional input, but this remains a scarce behaviour.

A similar approach is conducted to compute the level part. Have a look at the column B_1 of the ICM. Everytime B_1 occurred, the system was in the same state, input wise $(\overline{k1}.\overline{k2}.\overline{a0}.a1.\overline{a2}.\overline{b0}.b1.\overline{c0}.c1)$ This information is correct, but useless, since the values of most sensors are irrelevant to the status of B, and do not have a direct influence on it. Therefore, only inputs whose associated events were at least observed once with the output event are considered influent. The function $F(OE_j)$ is therefore computed as a conjuction of level disjunctions:

$$F(OE_j) = \prod_k DisjL_k$$

where $DisjL_k = \bigoplus_i u_i$ with

$$\forall i, DCM_{ij} \neq 0$$
$$\sum_i ICM_{ij} = 1$$

For instance, when looking at $A+_1$ **(3)**, non-zero entries in the DCM are k1_1, k2_1, a0_1, b0_1, c0_1; in the ICM, $Pr(k1 = 1|A+_1) + Pr(k2 = 1|A+_1) = 1$,$Pr(a0 = 1|A+_1) = 1$,$Pr(b0 = 1|A+_1) = 1$ and $Pr(c0 = 1|A+_1) = 1$, so the computed level function is $G(A+_0) = (k1 = 1 \oplus k1 = 2) \wedge (a0 = 1) \wedge (b0 = 1) \wedge (c0 = 1)$. The computed OEFFs for the example are summed up in Table 2.2.

OE_j	$G(OE_j)$	$F(OE_j)$
A+_1	ε	$((k1 \oplus k2) \wedge a0 \wedge b0 \wedge c0)$
A+_0	$a1_1 \oplus a2_1$	(=1)
A-_1	$a1_1 \oplus a2_1$	(=1)
A-_0	$a0_1$	(=1)
B_1	$a1_1$	(=1)
B_0	$b1_1$	(=1)
C_1	$a2_1$	(=1)
C_0	$c1_1$	(=1)

Table 2.2: Firing functions computed for the sorting system

It is possible that some inputs do not appear in the computed OEFFs, which means that their events or levels could not be causally connected to an output. In which case, a set of unassigned inputs D is created. In the case of the sorting system, $D = \emptyset$.

The problem of finding firing functions consists, for each column in the DCM, in choosing the k non-zero cells, and find in these k cells all possible combinations that sum to 1. To find all combinations, all subsets must be tested, hence 2^k. Since there are at most m inputs, hence $2m$ input events or levels, the complexity of finding firing functions is at worst $\mathcal{O}(4^m)$. In practice, in sequential systems, the number of inputs simultaneously observed with a given output event remains low; however, it can reach untractable values when systems grow bigger and exhibit more concurrency.

2.2.2 Construction of the transitions and observable places

From the OEFFs computed in Table 2.2, elementary IPN fragments are easily created. One observable place is attributed to each output y_j, then one pre-transition is created, labelled with $\chi(\uparrow y_j)$, and one post-transition is created, labelled with $\chi(\downarrow y_j)$. In the case of a non-assigned input $u_d \in D$, two isolated transitions, labelled with $\uparrow u_d$ and $\downarrow u_d$ are created. The result for the sorting system is presented in Figure 2.4.

However, recall that multiple output events can occur in the same vector. In that case, a single transition, generating the observed output events alltogether must be created, and labelled with a firing function consistent with the OEFFs of all observed output events. Therefore, for each observed vector $E(k)$ such that there is at least one

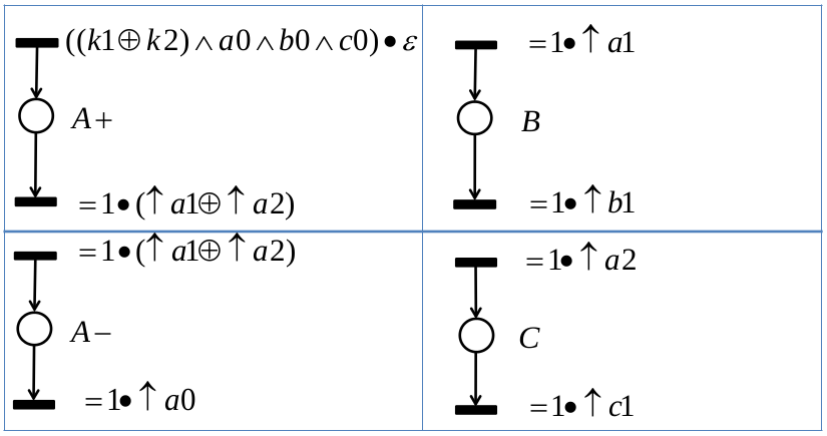


Figure 2.4: Elementary observable fragments constructed from the firing functions

elementary output event ($OE(k) \neq 0$):

1. Consider all elementary output events OE_j occurring in OE(k)
2. For each OE_j, consider $G(OE_j)$
3. Find in each disjunction of $G(OE_j)$ the input event IE_j which occured in IE(k)
4. Build $G = \prod_j IE_j$, where IE_j are the input events chosen for all observed output events OE_j.
5. Repeat the procedure to build $F = \prod_j u_j$, where u_j are the input levels chosen for all observed output events OE_j
6. Create a transition, if not already existant, labelled by $G \bullet F$, and adequately connected to the observable places related to all OE_j.

Two event vectors from the sorting system are considered to illustrate this construction. Consider E(1) in Table 2.1, verifying $IE(1) = \uparrow k1$ and $OE(1) = \uparrow (A+)$. From $\chi(\uparrow A+)$ in Table 2.2, there is only one disjunction to be considered $(k1 \oplus k2)$, from which the first term is chosen. A transition is created as pre-transition of the observable place $A+$, and labelled with $(\varepsilon) \bullet (k1 \wedge a0 \wedge b0 \wedge c0)$. This is illustrated in Figure 2.5.

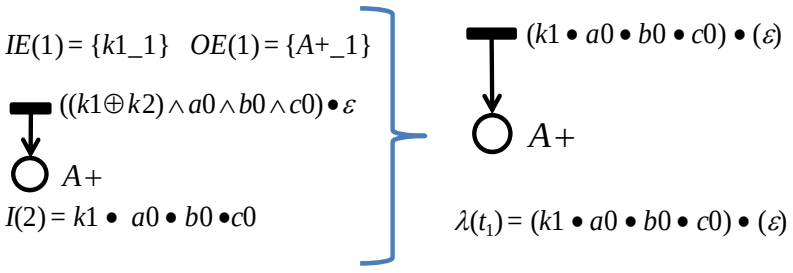


Figure 2.5: Example of the creation of a new transition, by choosing the right input conditions

Consider now E(4), where $OE(4) = \downarrow (A+) \wedge \uparrow (A-) \wedge \uparrow B$, and $IE(4) = \uparrow a1$. The event $\uparrow a1$ is chosen in the disjunctions of $G(\downarrow A+)$ and $G(\uparrow A-)$. A transition is created with $A+$ as pre-places and $A-$ and B as post-places, labelled with $\uparrow a1$. This is illustrated in Figure 2.6.

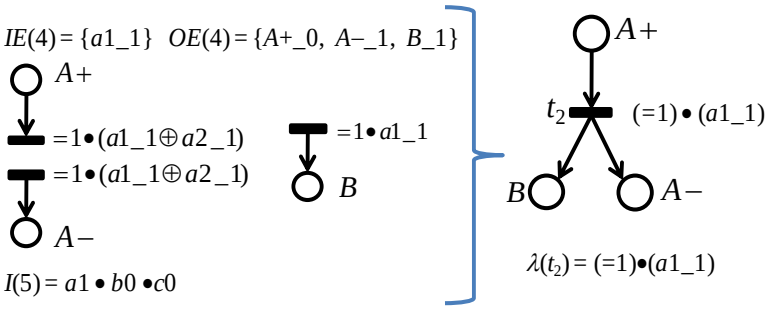


Figure 2.6: Example of the creation of a new transition, based on a simultaneous output events observation

Notice that some Event vectors such as E(2) or E(3) do not have elementary output events; since the associated elementary input events do not belong to D, no transition is created. The behaviour expressed by these events is not represented in the observable part of the IPN. Indeed, these are sensor events who do not provoke a reaction of the controller; they do not belong to the reactive behaviour of the system, and are therefore not represented in the model, leading to a more compact model. The net computed is shown in Figure 2.7. Notice that a single connected fragment was built in this case; in the generic case, multiple more or less connected fragments and isolated transitions are built.

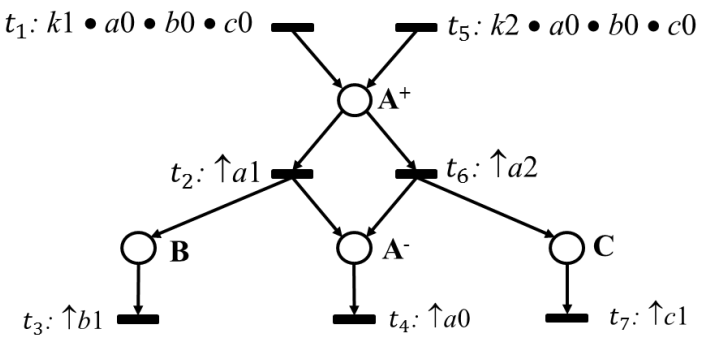


Figure 2.7: Result of the first step for the sorting system: the observable part

It is worth noting that at least one transition is created for each different output vector OE_j observed. For highly concurrent systems, due to the synchronisation of the controller, it is not uncommon to observe in the same event vector multiple output events who belong to concurrent subprocesses, and are therefore unrelated. A transition is created nevertheless in such a case. Suppose there are n concurrent subprocesses, each consisting of 1 output, such that any combinations of output events is possible. There are $\sum_{k=1}^{n} C_n^k 2^k$ potential transitions to build, if every combination is observed. Most of

these transitions are meaningless, and degrade the compacity and understandability of the net.

2.2.3 Determination of the firing sequence

E is then projected on the freshly built transitions and converted into a firing sequence S. See Table 2.3

Event vector	Elementary input events	Elementary output events	Equivalent transition
E(1)	$IE(1) = \uparrow k_1$	$OE(1) = \uparrow (A-)$	t1
E(2)	$IE(2) = \downarrow a_0$	$OE(2) = \varepsilon$	
E(3)	$IE(3) = \downarrow k_1$	$OE(3) = \varepsilon$	
E(4)	$IE(4) = \uparrow a_1$	$OE(4) = \downarrow (A-) \wedge \uparrow (A+) \wedge \uparrow B$	t2
E(5)	$IE(5) = \downarrow b_0$	$OE(5) = \varepsilon$	
E(6)	$IE(6) = \downarrow a_1$	$OE(6) = \varepsilon$	
E(7)	$IE(7) = \downarrow b_1$	$OE(7) = \downarrow B$	t3

Table 2.3: Translation into the firing sequence of the first vectors of E

This operation can be conducted simultaneously with the construction of transitions; E is parsed, the transitions are created if required at every new event vector, and S is updated on the fly. The full firing sequence computed is:

$$S = t_1t_2t_3t_4\ t_1t_2t_4t_3\ t_5t_6t_7t_4\ t_1t_2t_3t_4\ t_5t_6t_7t_4\ t_1t_2t_3t_4\ t_1t_2t_3t_4\ t_5t_6t_7t_4\ t_5t_6t_7t_4\ t_1t_2t_3t_4$$
$$t_5t_6t_7t_4\ t_5t_6t_7t_4\ t_5t_6t_7t_4\ t_5t_6t_7t_4\ t_5t_6t_7t_4\ t_1t_2t_3t_4\ t_1t_2t_3t_4\ t_1t_2t_3t_4\ t_1t_2t_4t_3\ t_1t_2t_3t_4$$

At the end of this step, the observable behaviour is completely identified. All observable places and transitions are built, the labelling of the transitions expresses the reactive behaviour, the model is compact, and the observed behaviour is expressed as a firing sequence S.

2.3 Inference of the unobservable behaviour

It remains to infer unobservable places to express the internal, non-observable dynamics of the system (Type3.a). To keep the model compact, the idea is not to extensively detail each internal evolution, which would require the creation of additional transitions, but instead to agregate the evolutions into unobservable places. The places act as connectors that connect the observable fragments, in order to express the whole behaviour of the system and be able to fire the sequence S.

Given an observable IPN $G^{Obs} = (P^{Obs}, T, I^{Obs}, O^{Obs})$, and a transition sequence $S = t_1t_2 \cdots \in T^*$, the aim is to build an ordinary PN $G^{Nobs} = (P^{Nobs}, T, I^{Nobs}, O^{Nobs})$ with an initial marking M_0 such that $(G = G^{Obs} \cup G^{Nobs}, M_0)$ is a complete IPN model reproducing S, and 1-bounded.

The 1-boundedness condition ensures that the marking of the observable places re-

mains binary, to adequately model the binary status of the associated outputs. Only one sequence S, often very long, without knowledge of cycles, is available. No counter-examples can be used (due to the incompleteness of the observation).

2.3.1 Finding Causal and Concurrent Transitions

The approach proposed is to infer relationships between transitions observed consequently in S. Namely, suppose that $t_a t_b$ is a substring of S. Two possibilities can explain this observation. The firing of t_a might be a requirement for the firing of t_b, in which case the relationship is *causal*. Otherwise, t_a and t_b might belong to different, concurrent subprocesses, and the firing of one does not affect the other. In this case, the relationship is *concurrent*, and $t_b t_a$ might also be a substring of S. Figure 2.8 illustrates these two possibilities.

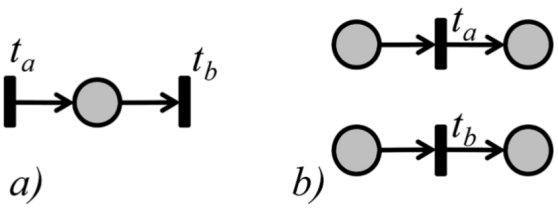


Figure 2.8: Structures that represent $t_a < t_b$: a) shows a causal relationship whereas b) shows a concurrent relationship

A few definitions are introduced to formalize this idea:

Definition 2.2 (Consecutive Observation)**.** *Let $Seq \subseteq T \times T$ be the relation that defines transitions observed consecutively in S.*

$$Seq = \{(t_i, t_{j+1}) | 1 \leq j \leq |S| - 1\}.$$

If $(t_a, t_b) \in Seq$, it is denoted $t_a < t_b$. If $t_a t_b t_a$ (or $t_b t_a t_b$) is a substring of S, then (t_a, t_b) are said to be in a two-cycle (cycle of length two) TC.

For instance, the firing sequence of Section 2.2.3 $S = t_1 t_2 t_3 t_4\ t_1 t_2 t_4 t_3\ t_5 t_6 t_7 t_4\ t_1 t_2 t_3 t_4\ t_5 t_6 \ldots$ leads to:

$$\begin{aligned} Seq = \{&(t_1, t_2), (t_2, t_3), (t_3, t_4), (t_4, t_1), (t_2, t_4), (t_4, t_3), \\ &(t_3, t_5), (t_5, t_6), (t_6, t_7), (t_7, t_4), (t_4, t_5), (t_3, t_1)\} \end{aligned}$$

No transitions are in a two-cycle.

Definition 2.3 (Causal and Concurrent transitions)**.** *Each couple $(t_a, t_b) \in Seq$ is either a causal relationship, in which case there exist a place between t_a and t_b, or a concurrent relationship, in which case there can not exist a place between them.*

These definitions are homogeneous to the ones used in rule-based approaches for Process Mining, namely the α-algorithm. All the remaining developments are focused

on determining for each couple in Seq which relationship to choose. The notion of systematic precedence is introduced.

Definition 2.4 (Systematic precedence)**.** *A transition t_a is preceded systematically by t_b, denoted $t_b \angle t_a$, iff t_b is always observed betweeen two occurrences of t_a in S. By convention, $t_b \angle t_b$ if t_b was observed at least twice in S. The Systematical Precedence Set of a transition t_b is the set of all transitions systematically preceding t_b, hence the transitions which must be fired to re-enable the firing of t_b. $SP(t_b) = \{t_a | t_a \angle t_b\}$.*

From S are computed the following SP sets:

$$SP(t_1) = SP(t_2) = \{t_1, t_2, t_3, t_4\}, SP(t_3) = \{t_1, t_2, t_3\}$$
$$SP(t_4) = \{t_4\}, SP(t_5) = SP(t_6) = SP(t_7) = \{t_4, t_5, t_6, t_7\}$$

2.3.1.1 Causal relationship

The idea behind a SP set is that if a transition systematically precedes another one, there must exist an oriented path in the Petri net connecting the two transitions. Besides, if the two transitions have been consecutively observed in S, then there must exist a place between the two transitions. Transitions in a two-cycle notably verify this property. The propositions and proofs can be found in [Estrada-Vargas et al., 2015]. Here is only recalled the caracterization of causal transitions:

Definition 2.5 (Causal transitions)**.** *The causal relationship set $CausalR$ is defined as:*

$$CausalR = \{(t_a, t_b) | ((t_a < t_b) \wedge (t_a \angle t_b)) \vee (t_a, t_b) \in TC\}$$

From Seq and the SPs computed for the sorting system, the causal set is computed:

$$\begin{aligned} CausalR = \{&(t_1, t_2), (t_2, t_3), (t_4, t_1), (t_2, t_4), \\ &(t_5, t_6), (t_6, t_7), (t_7, t_4), (t_4, t_5), (t_3, t_1)\} \end{aligned}$$

Namely, $(t_3, t_4), (t_4, t_3)$ and (t_3, t_5) were not classified as causal transitions. It remains to prove they are concurrent.

2.3.1.2 Concurrent relationship

Concurrent transitions imply that the firing of one does not affect the firing of the other. If two transitions (t_a, t_b) are concurrent, there exists no place between them, which also suggests that their firing can occur in any order. Allegedly, the substrings $t_a t_b$ and $t_b t_a$ may occur in the firing sequence S. A first characterization emerges:

Definition 2.6 (Concurrent transitions)**.** *The set of all pairs of concurrent transitions is called $ConcR = \{(t_a, t_b) | t_a \parallel t_b\}$*

Proposition 2.1. *Let t_a, t_b be two transitions which have been observed consecutively in a complete sequence S in both orders, i.e. $(t_a, t_b), (t_b, t_a) \in Seq^2$. Then $(t_a, t_b) \notin CausalR$ and $(t_b, t_a) \notin CausalR$ if, and only if $t_a \parallel t_b$*

However, due to the incompleteness of the observation, the two transitions can be concurrent without both substrings being present in S. Notice that only substrings of length 2 are considered to define the Seq set, hence all relationships. Therefore, if a convergence of the observed language size for length $n = 2$ is detected, as defined in Section 1.2.2, this proposition could be used. Notice however that this is not the case in Figure 1.13. In S, the new substring t_3, t_1 was observed in the few last vectors, which means that no convergence was achieved as well.

To circumvent this issue of incompleteness, a few rules have been proposed to characterize concurrent transitions. For instance, transitions t_a and t_b might belong to concurrent threads, synchronized by a transition t_k. The following rules ensues:

Proposition 2.2. *Let t_a and t_b be two transitions that have been observed consecutively in both orders. If:*

1. $t_a \notin SP(t_b)$ *and* $t_b \notin SP(t_a)$ *(Sequentially Independent)*
2. $\exists t_k$ *such that* $t_a \in SP(t_k)$ *and* $t_b \in SP(t_k)$

Then $t_a \parallel t_b$

This rule notably rules (t_3, t_4) as concurrent, since $(t_3, t_4) \in SP(t_1)^2$. The other rules and the proofs can be found in [Estrada-Vargas et al., 2015].

After the application of the rules to the sorting system, it comes:

$$\begin{aligned} CausalR &= \{(t_1,t_2),(t_2,t_3),(t_4,t_1),(t_2,t_4),(t_5,t_6),(t_6,t_7),(t_7,t_4),(t_4,t_5),(t_3,t_1)\} \\ ConcR &= \{(t_3,t_4),(t_4,t_3)\} \\ Seq'R &= SeqR - (CausalR \cup ConcR) = \{(t_3,t_5)\} \end{aligned}$$

There remains a relation that could not be decided, despite the use of multiple rules. A good quality of the rules is that their computation runs in polynomial time with the number of transitions and the length of the sequence. From the computed relations, the unobservable places can now be inferred.

2.3.2 Computing unobservable places

Each causal relation discovered must lead to the addition of a place between the two transitions concerned. However, a transition might be involved in multiple causal relations. Suppose that (t_k, t_{ai}) and (t_k, t_{aj}) are both causal relations. There are two cases:

- t_{ai} and t_{aj} are not concurrent and have not been observed consecutively. This is a case of choice, and one place is created for both relations, illustrated by Figure 2.9a).
- t_{ai} and t_{aj} are concurrent or have been observed consecutively. This is a case of concurrency, and one place is created for each relation, illustrated by Figure 2.9.

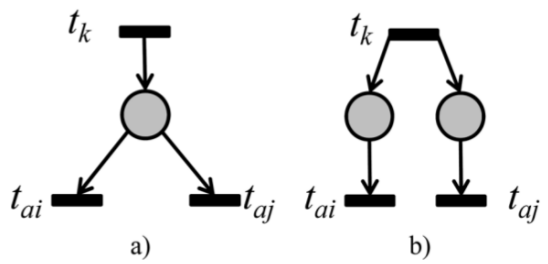


Figure 2.9: a)Choice and b)parallelism after the firing of t_k

So, if the first case occurs, places $[t_k, t_{ai}]$ and $[t_k, t_{aj}]$ are merged into one choice place $[t_k, t_{ai}t_{aj}]$. This operation is a merging of post-transitions. Similarly, $[t_{ki}, t_{ai}]$ and $[t_{kj}, t_{ai}]$ can be merged into $[t_{ki}t_{kj}, t_{ai}]$, in a merging operation of pre-transitions.

The unobservable part of the net is therefore built:

1. Create 1 unobservable place per causal relationship. In the case of a non-empty Seq' set, which means that some pairs could not be classified, they are assumed to be causal for now.

2. Compute the post-merging operation

3. Compute the pre-merging operation

The initial marking is then computed by reverse playing the sequence S in the PN structure:

- If output places are marked, the tokens are removed

- A token is added in each unmarked input place

Notice that this procedure only computes an initial marking M_0, but does not guarantee that S is firable in (N, M_0). The result of the unobservable behaviour for the sorting system is presented in Figure 2.10. Notice the capture of the concurrency between transitions t_3 and t_4.

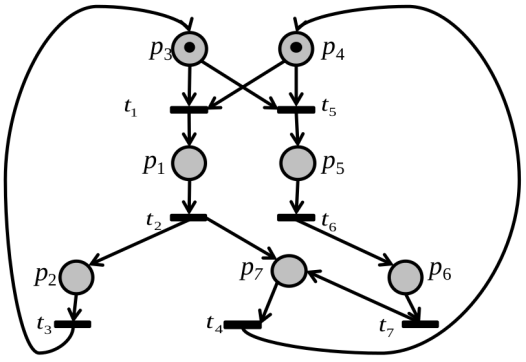


Figure 2.10: The non-observable part of the IPN computed for the sorting system

2.3.3 Verification of the net

This model might however not exactly reproduce S, namely because some relations in Seq' might have been uncorrectly determined. It remains to remove places or arcs that restrict too much the behaviour of the net. Namely, by trying to replay $S = t_1t_2t_3t_4\ t_1t_2t_4t_3\ t_5t_6t_7t_4\ t_1\ldots$ in the net of Figure 2.10, it occurs that that last firing of t_1 can not be executed, due to a missing token in p_3.

To verify the net, the token-flow equation is considered.

Proposition 2.3. *If the IPN model is correct, every non-observable place p must fulfill the token-flow equation:*

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

where Occ is a function returning the number of firings of a transition (t_i) in the given firing sequence S.

The equality ensues from the 1-boundedness condition on the net, and the ± 1 is required in case the final marking reached after the firing of S is not the initial one. In the case the equality is not respected, a correction rule is proposed:

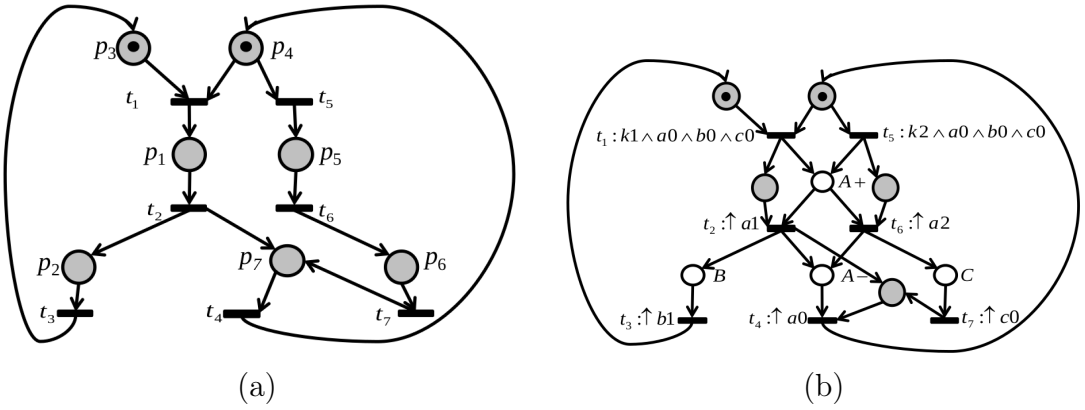
Proposition 2.4 (Correction rule)**.** *If a place does not satisfy the token flow equation, arcs relating transitions which are not in $CausalR$ are removed. If there are not $CausalR$ represented, the place is simply deleted*

For the sorting system, $Occ(t_1) = 12$, $Occ(t_2) = 11$, $Occ(t_3) = 11$, $Occ(t_4) = 20$, $Occ(t_5) = 9$, $Occ(t_6) = 9$, $Occ(t_7) = 9$. The token-flow equation is not verified for place p_3, as $(Occ(t_3) = 11) \neq (Occ(t_1) + Occ(t_5) = 21) \pm 1$. Since $(t_3, t_5) \in Seq'$, this relation was misinterpreted as causal; the edge between p_3 and t_5 is removed. The transitions are actually concurrent, and the observation is incomplete. With this correction, S becomes firable. The corrected net is shown in Figure 2.11(a), and the final IPN with both behaviours is shown in Figure 2.11(b).

2.4 Discussion and proposed improvements

The proposed method builds an IPN from a sequence w observed issued of the observation of a real closed-loop system with logical inputs and outputs.

The first step of the method extracts the reactive behaviour, *i.e.* the direct relationships between elementary input and output events, and translate them into the observable part of the Petri Net. The reactive behaviour is directly readable from the net. Apart from the computation of the firing functions, the procedure runs in polynomial time with the size of the system (number of inputs and outputs), and the length of the observed sequence w. The difficulty of computing the firing functions is theoretically



(a) (b)

Figure 2.11: (a) The unobservable part, corrected by the token-flow equation; (b) The final IPN with both interpretation layers

exponential with the number of inputs, but remains tractable for systems exhibiting low concurrency.

The second step of the method infers unobservable places to complete the net and assemble the observable fragments. Relationships between transitions are extracted from the firing sequence S, based on numerous rules. Unobservable places are built on these relationships, and the resulting net is verified. This rule-based approach has the advantage of polynomial complexity with the number of transitions and the length of the firing sequence S. The final model is compact models, exhibits concurrency, and huge state-machine-like IPNs are avoided (like the one proposed in [Estrada-Vargas et al., 2014]).

The method is adapted to our technological hypotheses, and produces a satisfying result to reverse-engineer a closed-loop DES. There are nevertheless two major directions for improvements: scalability for massively concurrent systems, and robustness of the unobservable behaviour inference.

2.4.1 Scalability and concurrency

As pointed out in Section 2.2, a first difficulty resides in the computation of the OEFFs. Given an output OE_j and n input events, suppose the following DCM column: $DCM_{1j} = 0.95$, $DCM_{2j} = 0.05$, and $\forall i \in [\![3, n]\!], DCM_{3j} = 0.01$. Such an observation means that IE_1 is the main cause for the output event OE_j, but is sometimes replaced by a scarce cause IE_2 (typically, a security in a system which is not triggered in every production cycle). All other inputs are unrelated, but sometimes observed together with OE_j due to concurrency and synchronization by the controller.

In this case, there are $1 + C_{n-2}^{5}$ combinations of input events such that the sum of their probabilities equals 1. 2^n combinations must be studied, even if most of these combinations are irrelevant. This is an issue to be dealt with in order to improve the scalability of the method on one hand, and the quality of the identified model on the other hand.

Another issue is the spurious synchronization of unrelated output events, that provokes an increase in the number of transitions and destroys the understandability of the net. To improve the scalability of the method and the quality of the model, the meaningless transitions should be detected and removed.

Chapter 3 proposes improvements to take into account the synchronization effect of the controller, and improve the scalability of the method. Understandable observable models can be built efficiently for concurrent systems of reasonable size.

Ultimately, when the system becomes massive enough such that a monolithic model, even simplified, becomes hard to understand, a partitioning of the system into subsystems of smaller size has to be considered. A solution to the task of automatic partitioning is presented in Chapter 5, and leads to distributed models, less costly and more understandable than the monolithic one.

2.4.2 Limits of the unobservable behaviour discovery

As was already seen in section 2.3, the use of numerous rules does not guarantee that every case is decidable. Namely, no rule was satisfying to decide whether (t_3, t_5) were concurrent or causal. No proof for the computation of the initial marking was also presented, and the verification procedure only verifies occurrences of transitions, but not that S was indeed firable.

In some cases places can not even be found, as in the following example.

Example 2.1. *Consider a system with two inputs (u_1, u_2) and two outputs (Y_1, Y_2). The observable part of the net is in Figure 2.12(a); it consists in two fragments, and two isolated transitions, since u_2 could not be related to any output, for a total of 6 transitions.*

The firing sequence is:

$$t_1t_2t_1t_2t_3t_4t_5t_1t_2t_6t_1t_2t_1t_2t_3t_4t_5t_1t_2t_6t_1t_2t_1t_2t_3t_4t_5t_6t_1t_2t_1t_2t_1t_2t_3t_4t_5t_1t_2$$
$$t_6t_1t_2t_1t_2t_3t_4t_5t_6t_1t_2t_1t_2t_1t_2t_3t_4t_5t_1t_2t_6t_1t_2t_1t_2t_3t_4t_5t_1t_2t_6t_1t_2t_1t_2t_3t_4t_5t_6\ldots$$

from which the Seq set is computed:

$$Seq = \{(t_1, t_2)(t_2, t_1)(t_2, t_3)(t_3, t_4)(t_4, t_5)(t_5, t_1)(t_2, t_6)(t_6, t_1)(t_5, t_6)\}$$

In this case, no couple can be decided concurrent by using the proposed rules. Consequently, all couples are supposed causal. Since $(t_2, t_3), (t_2, t_6) \in CausalR^2$, and $t_3 \nparallel t_6$, a place $[t_2, t_3t_6]$ is created, but deleted by the token-flow verification ($Occ(t_3)$=$Occ(t_6)$=20; $Occ(t_2)$=60). The resulting net is in Figure 2.12(b). S is reproducible, but there are still sink and source transitions.

An actual solution that reproduces S is the net of Figure 2.12(c). A place such as $[t_2t_5, t_1t_3]$ can never be computed by the proposed rules since (t_5, t_3) was never observed, and does not belong to Seq.

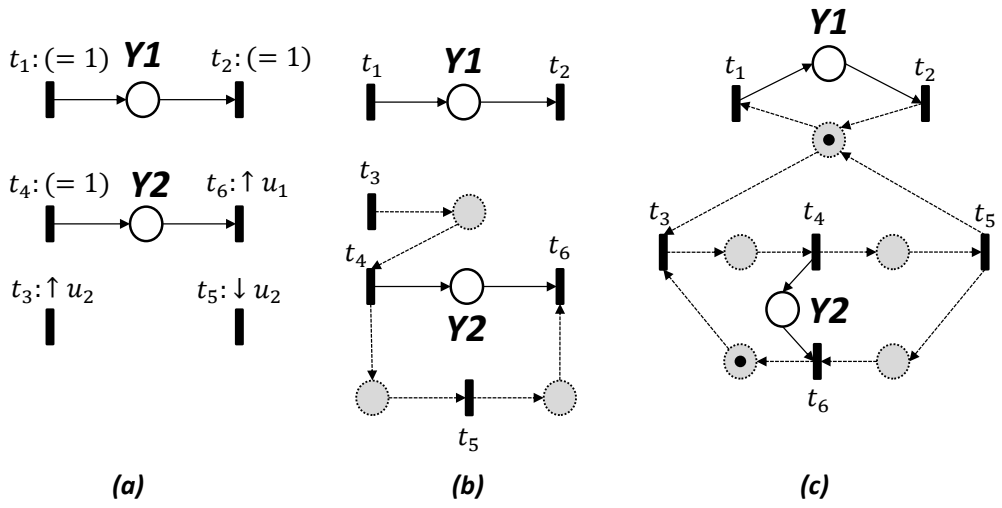


Figure 2.12: (a) The observable part of the net; (b) The net identified ; (c) A net that satisfies the problem

Inferring places using only the *Seq* set naturally excludes places working as memories by definition. Have a look at the net of Figure 2.13. Transitions t_1 and t_4 can never occur sequentially, since t_3 will always be fired inbetween. The place connecting these two transitions can not be inferred with the current method.

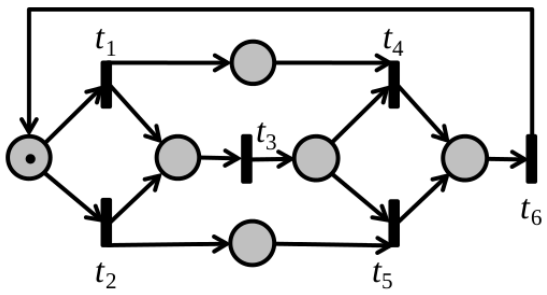


Figure 2.13: A non free choice net with two memory places

The proposed approach can be improved towards more genericity. Chapter 4 proposes a new approach in that direction, based on the use of projections, to infer the unobservable places. Only one dependency relation is defined on transitions, and the genericity of the approach is proven in a single theorem.

Chapter 3
Scalability of the observable behaviour construction

Introduction

This chapter proposes two approaches to improve the scalability of the construction of the observable part of the IPN. Firstly, the system (MSS) used as illustrative example through the whole thesis is introduced in section 3.1. Notably, it exhibits enough concurrency to provoke a degradation of the observable part, both in calculus cost and quality of the model. These issues are notably due to the synchronization of the PLC, as exposed in Section 3.2. The first proposition is a filter of the direct causality matrix based on the boolean absorption identity, exposed in Section 3.3; its result is the ability to compute all firing functions, and improve their quality as well. Then, the second proposition is the removal of spurious transitions, replaced by equivalent firings, exposed in Section 3.4. Both methods are tested and validated on the MSS.

3.1 Illustrative system: the MSS

3.1.1 Presentation

The Mechatronics Standard System (MSS)(Figure 3.1) is a real-world laboratory manufacturing system developed by Bosch, available on the experimental platform of the LURPA (ENS Cachan, France). The purpose of this system is to sort workpieces according to material and presence of a bearing. Workpieces of plastic, brass and steel are treated. The system is decomposed into 4 stations, displayed in Figure 3.2.

The workpieces are introduced in the Lift of the Feeder station (1). When the process is started, the pieces are brought to the height of the Pusher, which makes them to fall in the Chute. They are stored in the Chute until the chariot of the Testing station (2) comes to get the workpieces one at a time. The workpiece is carried under the sensors of the Sensor array, where material and presence of a bearing are detected and recorded. Then the workpiece is carried to the chariot of the Processing station (3) by a gripper. The In-press is used to put a bearing in an empty gear, and the Out-press to extract a

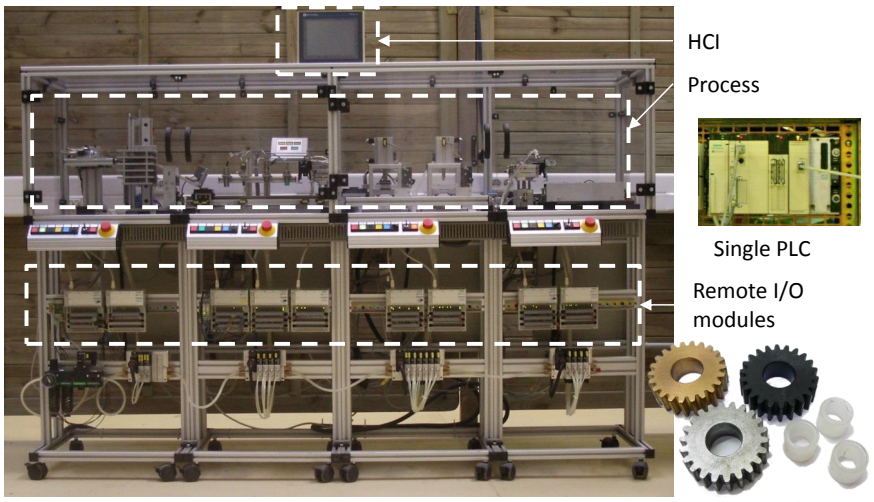


Figure 3.1: A picture of the MSS in the LURPA, and of the workpieces (gears and bearings) processed.

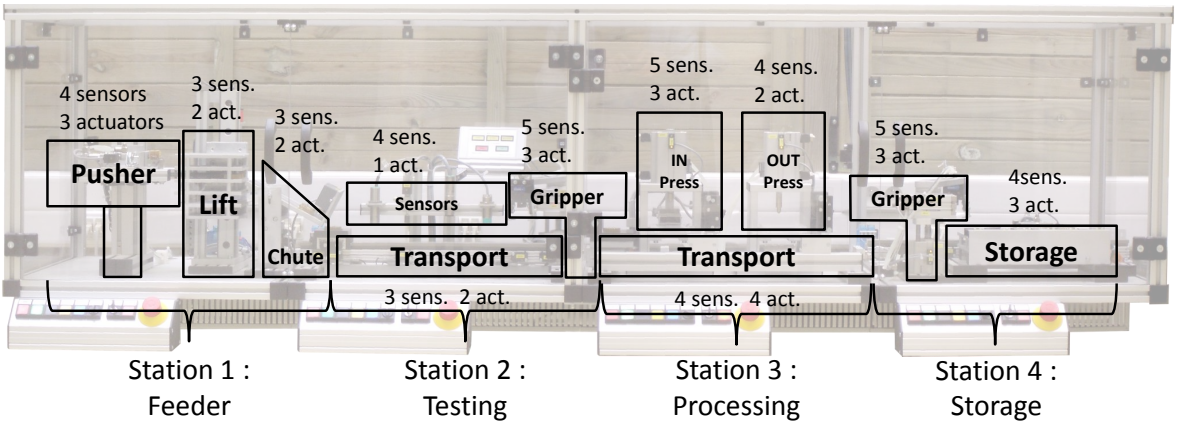


Figure 3.2: Scheme of the MSS, decomposed in 4 stations and 11 subsystems

bearing. Finally, the workpiece is caught by the Rotary gripper of the Storage station (4) and given to the feeder who sorts it according to its material.

The interest of this chain is the exhaustivity of the technologies used for the various sensors and actuators. Among the sensors, there are optical, inductive, capacitive or mechanical detectors as well as reed or cylinder switches. The actuators are electronic relays to aliment electrical-powered devices such as motors or magnets, and valves to aliment pneumatic-powered devices such as cylinders. All these components have binary values.

There are 43 sensors and 30 actuators relevant to the processing of the workpieces, all other inputs or outputs being lights or buttons to be used by an operator. The repartition of these 73 components is explicited in Figure 3.2. The first number in the labelling of an I/O determines to which station it belongs (1xxx belongs to station 1).

The control of the chain is realized by a single PLC, who communicates with the remote I/O modules through Modbus TCP via Ethernet. The deported modules can

be seen under the boards in the picture of Figure 3.1. There is one module dedicated to inputs, and one module dedicated to outputs for each station (an additional output module is available in the second section). The duration of a PLC cycle lasts around 5ms.

Even though each workpiece is sequentially treated by each station, many workpieces are treated simultaneously in the whole chain, namely during a continuous production phase. This chain exhibits therefore massive concurrency, and the behaviours of the different stations are often interleaved. For instance, outputs from different stations can be updated in the same PLC cycle, even though they are unrelated. Also, the chain exhibits choices with shared ressources. Namely, the chariot of the third station is sollicited by the two grippers and the two presses. This chain is therefore representative of reactive industrial DES, controlled by a PLC and exhibiting multiple simultaneous processes, hence high concurrency.

3.1.2 Data collection

The system was observed during 20 production cycles. Each production cycle lasts around 8 minutes. It consists in processing three tables of eight gears. The gear materials, and the presence of bearings are random in each table. The recipe used in the third station depends on the material: bearings are removed from brass gears, put in steel gears, and removed then put in plastic gears.

Twenty I/O vector sequences are recorded and are concatenated, leading to an observed sequence of length $|w| = 63797$ vectors. The event vector sequence $E(k) = w(k+1) - w(k)$ is computed as well, and the observed languages are computed according to definitions presented in Section 1.2.2 from the event vector sequence. The resulting growths of L^n_{Obs} are plotted in Figure 3.3 for $n = 1, 2$.

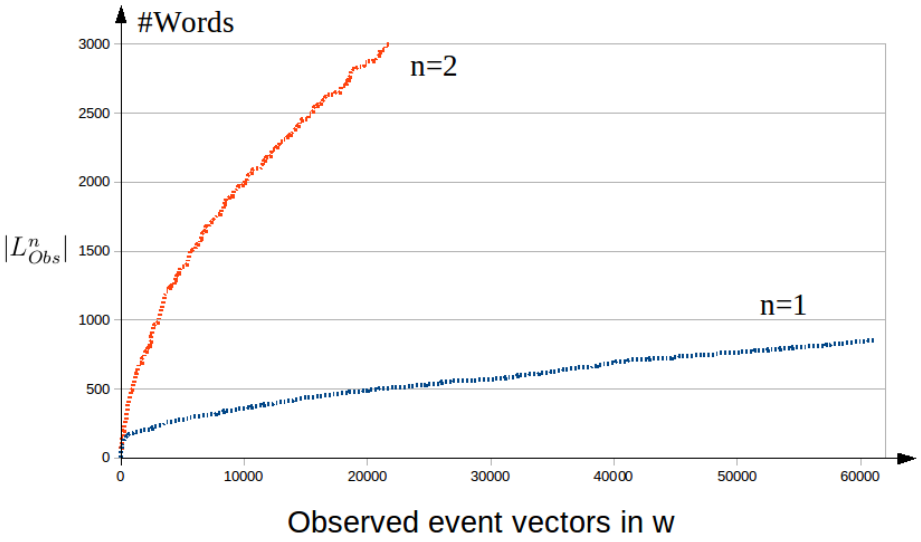


Figure 3.3: Observed language of the MSS over 20 production cycles, for $n = 1, 2$

Convergence is not achieved even for $n = 1$. It means that new event vectors $E(k)$ keep being observed, even after a long observation time. An explanation to the continuous observation of new event vectors is the combination of the massive concurrency of the system and the synchronous reading/writing cycle of the controller.

3.2 Resynchronization of asynchronous events in concurrent systems and consequences

A very simple example is used here for illustration. Two concurrent subsystems are considered: u_1 is an input that causes y_1, and u_2 is an input that causes y_2. The two processes are fully independent; the rising and falling edges of u_1 and u_2 can occur at any time. A chronogram corresponding to the most common case is given in Figure 3.4. The first two rows are the true values of the inputs; the second two rows are the values known by the controller, only updated during an input reading phase. Due to the observation protocol, the read values are recorded in the event vectors; the true values of the inputs remain unknown. In this case, the duration between the real occurences of input events was longer than the duration of a cycle; the two elementary input events were read in different PLC cycles. u_1 and u_2 are observed asynchronoulsy and the resulting event vector is:

$$E = [\uparrow u_1 \uparrow y_1]; [\uparrow u_2 \uparrow y_2]; [\varepsilon]; [\downarrow u_1 \downarrow y_1]$$

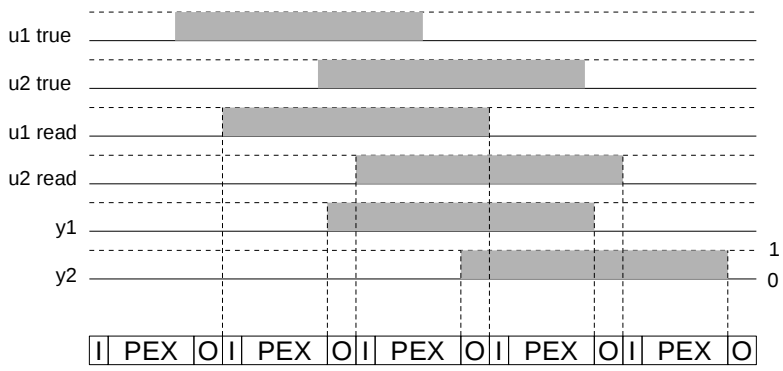


Figure 3.4: Two concurrent processes observed asynchronously

However, the duration between u_1 and u_2 might be shorter, and the two real elementary events might occur between two input reading phases. This case is illustrated in Figure 3.5. Even though the real events are asynchronous, they are read in the same cycle, and spuriously perceived as synchronous. The resulting event vector is:

$$E = [\uparrow u_1 \uparrow u_2 \uparrow y_1 \uparrow y_2]; [\varepsilon]; [\downarrow u_1 \downarrow u_2 \downarrow y_1 \downarrow y_2]; [\varepsilon]$$

The second case occurs less often than the first one. The controller cycle lasts between 5 and 15ms, which is a short time window within which the elementary events must occur. However, when size and concurrency of the system increase, this case

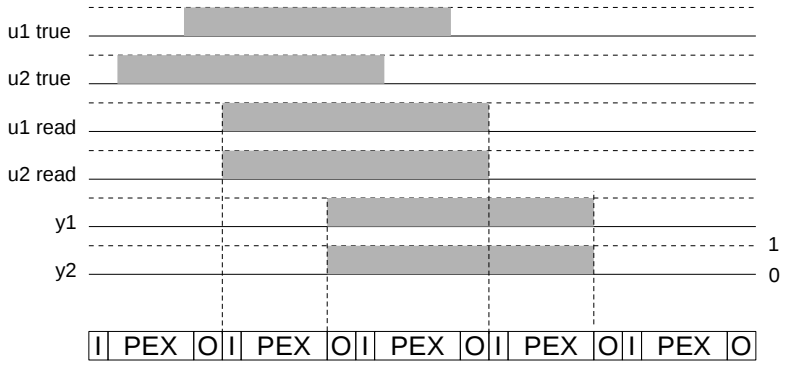


Figure 3.5: Two concurrent processes spuriously observed synchronously

becomes more and more common. New observations of spuriously synchronized events keep happening, as shown by the language growth for $n = 1$ in Figure 3.3. In this very simple case, four additional event vectors can be observed: $[\uparrow u_1 \uparrow u_2 \uparrow y_1 \uparrow y_2]$, $[\uparrow u_1 \downarrow u_2 \uparrow y_1 \downarrow y_2]$, $[\downarrow u_1 \uparrow u_2 \downarrow y_1 \uparrow y_2]$ and $[\downarrow u_1 \downarrow u_2 \downarrow y_1 \downarrow y_2]$. Suppose now that n similar concurrent processes are considered, and 2^n event vectors can be observed.

When a new event vector $E(k)$ is observed, two effects might happen, depending on the content of $E(k)$:

$IE(k) \neq 0 \wedge OE(k) \neq 0$ At least one input event is observed with at least one output event. Corresponding cases in the DCM see their values increased (Section 2.2.1)

$|OE(k)| > 1$ At least two outputs event were observed. A transition is created to represent the corresponding output events and connect the corresponding observable places (Section 2.2.2)

Suppose that in the previous case, the processes are observed synchronously only 4 in 100 times, uniformly between the four possible cases. The resulting net is presented in Figure 3.6(a) and the DCM in Table 3.1. In the DCM, the direct causalities can still be discovered, but other cases are no longer zero. In the net, transitions t_5, t_6, t_7, t_8 are created, but their occurrences are scarcer. Firing t_5 is equivalent as firing t_1 and t_3. Net (b) is obtained after getting rid of these scarce transitions.

	$\uparrow Y_1$	$\downarrow Y_1$	$\uparrow Y_2$	$\downarrow Y_2$
$\uparrow u_1$	1	0	0.02	0.02
$\downarrow u_1$	0	1	0.02	0.02
$\uparrow u_2$	0.02	0.02	1	0
$\downarrow u_2$	0.02	0.02	0	1

Table 3.1: DCM of two concurrent processes

We now consider the case of the MSS, with observation of Figure 3.3. There are 43 inputs and 30 outputs, leading to a DCM of size 86×60. The mean number of non-zero cells is 9.15 per column, maxing out at 29. 85% of these cells are lower than

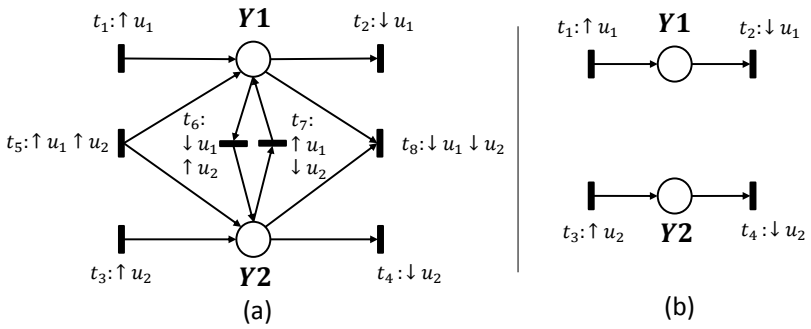


Figure 3.6: (a) Observable part identified for two concurrent processes with spurious transitions (b) Simplified desirable model

1%, meaning there are a lot of spurious, noisy observations, to be distinguished from true causes. Recall that the complexity of finding an OEFF is $\mathcal{O}(2^c)$, where c is the number of non-empty cells of the column. Only 50 of the 60 OEFFs could be computed (25 outputs) on a laptop.

Regarding the transitions, 402 transitions are built based on the observation (of the 25 computable outputs). The frequencies of the transitions, numbered in the chronological order of their first firing, are plotted in Figure 3.7. The last transitions are very unfrequent, and due to spurious synchronizations. Namely, 126 (31%) transitions have been fired only once; *i.e.* correspond to a combination of output events observed only once, and 278 (69%) have been fired less than 20 times (the number of executions of the process). These transitions spoil the understandability of the model.

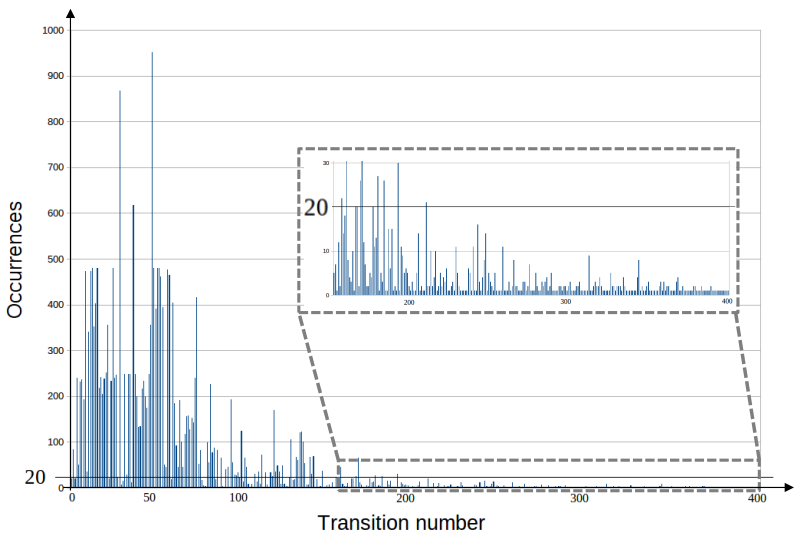


Figure 3.7: MSS transition occurences

Two improvements are therefore proposed. The causality matrix is first filtered to get rid of noisy I/O correlations and ease the computation of OEFF in Section 3.3. Then, unfrequent spurious transitions are reduced in Section 3.4, to ease the construction and

the understanding of the observable fragments.

3.3 Filtering of the causality matrix

The objective of the filter is summarized in Figure 3.8: given an output event OE_j, for each input event IE_i simultaneously observed, *i.e.* $DCM_{i,j} \neq 0$, decide whether IE_i is a potential cause, or is instead a noisy correlation due to a synchronization of the controller. In the latter case, the value of $DCM_{i,j}$ is set to zero.

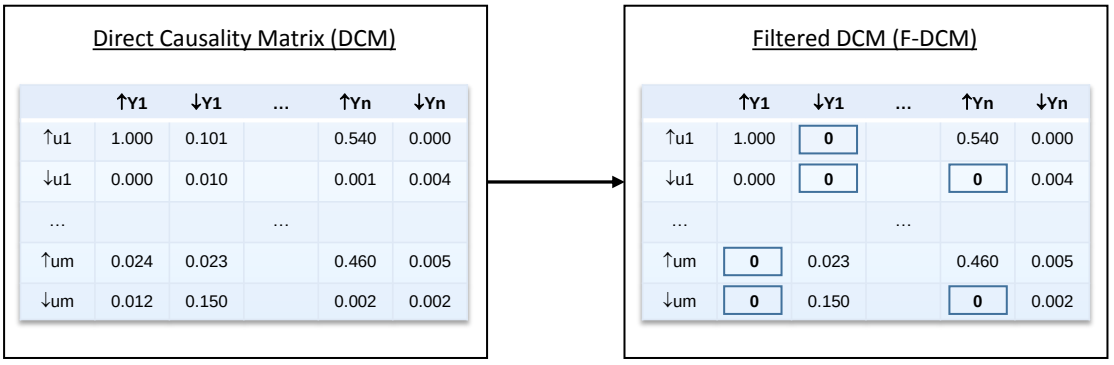


Figure 3.8: Principle of the filter: put zeros in cells corresponding to noisy observations

3.3.1 Design of the filter

A first idea is to set a threshold, for instance each cell whose value is lower than 0.05 is nullified. However, the risk is high to get rid of a scarce, but true cause by using a frequency criterion. For instance, consider the output event 1Y04_0 of the MSS, which is the end of the extension of a cylinder. Its corresponding DCM column is presented in Table 3.2. Only the 28 non-empty cells are represented.

1B06_1	1S10_1	2B11_0	2B11_1	2B12_0	2B12_1	2B15_1
0,937	0,063	0,001	0,002	0,001	0,009	0,001
2B17_1	2S06_1	2S07_0	2S07_1	2S10_0	2S10_1	3B06_1
0,004	0,001	0,002	0,001	0,001	0,001	0,001
3B07_0	3B07_1	3B10_0	3B11_1	3B13_1	3B16_0	4B07_0
0,001	0,001	0,001	0,001	0,001	0,001	0,002
4B10_0	4B11_0	4B12_1	4B13_0	4B14_1	4B15_0	4B16_0
0,002	0,002	0,002	0,001	0,001	0,001	0,003

Table 3.2: Non empty cells of the column $\mathrm{DCM}_{i,1Y04_0}$

The main cause seems to be 1B06_1, and a more unfrequent one is 1S10_1. Physically, the main cause is the switch at the end position of the cylinder, whereas the secondary is the end position of another cylinder working in the same area, *i.e.* it is

due to a security constraint. The expected event part of the OEFF is $G(1Y04_0) = 1B06_1 \oplus 1S10_1$. If the threshhold is set higher than 0.063, the secondary cause is lost, and the discovered OEFF is $G(1Y04_0) = \varepsilon$. It is impossible to choose a threshold *a priori*.

The proposed filter is therefore not based on frequencies, but is purely logical, based on the absorption identity in Boolean algebras:

$$a + a.b = a$$

The idea is to look for prime implicants of the output events, expressed as a Boolean formula. Input events who are prime implicants are conserved, whereas other are nullified.

Definition 3.1 ([Goldsmith et al., 2008]). *An implicant of a Boolean formula φ is a monomial C such that $C \rightarrow \varphi$ is valid. A monomial C is a prime implicant of φ if and only if C is an implicant of φ and for every proper subset $S \subset C$ it holds that S is not an implicant of φ.*

To build the Boolean formula, let OE_i be the output event of interest, and consider the set of input event vectors observed simultaneously $Sim(OE_i) = \{IE(k) | OE_i \in OE(k) \wedge IE(k) \neq 0\}$. The events are assimilated to boolean variables (1 if the event is present, 0 if it is not). A conjunction of input events is built for each observation where OE_i is present. The formula is then built by taking the disjunction of all observations. If OE_i is observed alone at least once, then $OE_i = 1$.

$$OE_i = \sum_{\substack{k \\ OE_i \in OE(k)}} \left(\prod_{\substack{j \\ IE_j(k) \neq 0}} IE_j \right)$$

The output event is expressed as a Boolean formula in the disjunctive normal form (DNF), where the monomials are input events conjunctions. The obtained formula is *monotone*, as it does not contain negations. For monotone formula, the minimum equivalent DNF is unique and is the disjunction of all prime implicants. It is shown in [Goldsmith et al., 2008] that the function which on input φ, a monotone formula, outputs the smallest DNF for φ, is in output-polynomial time if and only if P = NP. The problem of finding all prime implicants is therefore not simple.

The implicants can be simplified into prime implicants by using the absorption identity. It is illustrated on the following example.

Example 3.1. *Suppose the following observation for output event OE_i (#Occ representing the number of occurences of each event vector, for a total number of 100 obser-*

vations):

$$\begin{cases} IE_1 \wedge IE_2 \wedge OE_i \ (\#Occ = 41) \\ IE_1 \wedge IE_3 \wedge IE_5 \wedge OE_i \ (\#Occ = 8) \\ IE_4 \wedge OE_i \ (\#Occ = 45) \\ IE_4 \wedge IE_5 \wedge OE_i \ (\#Occ = 6) \end{cases}$$

$Sim(OE_i)$ has four members, whose disjunction builds the Boolean formula:

$$OE_i = IE_1.IE_2 + IE_1.IE_3.IE_5 + IE_4 + IE_4.IE_5$$

The fourth term is absorbed by the third one, resulting in the minimal disjunctive form:

$$OE_i = IE_1.IE_2 + IE_1.IE_3.IE_5 + IE_4$$

In this example, IE_5 is absorbed by IE_4, IE_4 being sufficient to cause OE_i. IE_4 seems to be the true cause in the fourth conjunction, because it appeared alone in the third one. IE_5 is therefore interpreted as noisy: it is most likely an unrelated input event that was synchronized with IE_4.

IE_5 can not be a sufficient condition of OE_i. We therefore make the additional assumption that if an input event is ruled once as noisy regarding a given output event, it is also noisy in all the cases it was observed simultaneously with said output event. This is a realistic assumption for industrial systems: a noisy input IE_5 might have been the cause of another output event occuring simultaneously with OE_i. Consequently, regarding the DCM, a zero is put in the cell DCM_{IE_5,OE_i}, and IE_5 is removed totally from the updated formula:

$$OE_i = IE_1.IE_2 + IE_1.IE_3 + IE_4$$

To apply the filter, the sets $Sim(OE_i)$ must additionally be computed from the event vector sequence E. This set is partitioned into subsets $Sim(OE_i, n)$ to introduce n as the number of simultaneously observed input events. Finally, $N = max(n)$ is the maximal number of input events observed simultaneously with OE_i. In the worst case, $N = 2|\mathbb{U}|$. The filter is described by Algorithm 3.1. The idea is to decide for each input event whether it might be a valid cause (IE_{Val}) or if it is noisy (IE_{Noisy}). Input events observed alone are decided valid, and input events who are absorbed by them are ruled noisy. If input events remain unsorted, the procedure is repeated for 2-uples of input events, then 3-uples ... until all events are classified.

Proposition 3.1. *Algorithm 3.1 terminates and runs in $\mathcal{O}(N.|Sim(OE_i)|)$*

Proof. The highest value that n can take is N. In this ultimate loop, lines 6-10 rule every remaining input event as valid. Lines 11-17 are not applied, and $IE_{Undecided}$ becomes empty at line 18, ensuring an exit of the while loop and termination of the algorithm.

Loops 6-10 and 11-17 are executed at worst $|Sim(OE_i, n)|(|Sim(OE_i, n+1)|)$ times.

Algorithm 3.1 Filter of a column of the DCM

Require: $Sim(OE_i) = \{Sim(OE_i, 1), Sim(OE_i, 2), \ldots Sim(OE_i, N)\}$
Ensure: IE_{Noisy}

```
1: IE_Undecided ← Sim(OE_i)
2: IE_Val ← ∅
3: IE_Noisy ← ∅
4: n =1
5: while IE_Undecided ≠ ∅ do
6:    for Observation ∈ Sim(OE_i, n) | Observation ∩ IE_Undecided ≠ ∅ do
7:       {Input events in this observation are not decided yet}
8:       IE_Val ← IE_Val∪ (Observation ∩ IE_Undecided)
9:       {Unsorted input events whose observation matches k are decided valid.}
10:   end for
11:   for Observation' ∈ Sim(OE_i, n + 1) do
12:      {Checking for noise in observations of bigger size using absorption}
13:      if Observation' ∩ IE_Val ≠ ∅ then
14:         IE_Noisy ← IE_Noisy∪ (Observation' - IE_Val)
15:         {Absorbed input events are noisy}
16:      end if
17:   end for
18:   IE_Undecided ← IE_Undecided − IE_Val − IE_Noisy
19:   n ← n + 1
20: end while
```

At worst, $|Sim(OE_i, n)| = |Sim(OE_i)|$, therefore the loops run in $\mathcal{O}(|Sim(OE_i)|)$. Finally, the while loop is executed at most N times. Algorithm 3.1 runs at worst in $\mathcal{O}(N.|Sim(OE_i)|)$. □

Recall that in a real system, the dynamics of the process is quite slow (compared to the reactivity of the controller), and simultaneous inputs do not occur so often. Typically, N=3 for the MSS. $|Sim(OE_i)|$ remains of reasonable size as well, and is bounded by the length of the observed event sequence $|E|$. Algorithm 3.1 can therefore be applied efficiently.

Example 3.2 (Example 3.1 cont.)**.** *For $n = 1$, IE_4 is decided valid, since it is observed alone in the third observation. Consequently, IE_5, never observed alone, is absorbed by IE_4 in the fourth observation, and decided noisy. IE_1, IE_2 and IE_3 are not decided yet. For $n = 2$, they are validated due to the first and second observation, and all input events are sorted.*

Consequently the DCM becomes :

IE_1	IE_2	IE_3	IE_4	IE_5		IE_1	IE_2	IE_3	IE_4	IE_5
0.49	0.41	0.08	0.51	0.14	→	0.49	0.41	0.08	0.51	**0**

The event part of the firing function is:

$$G(OE_i) = (IE_1 \oplus IE_4).(IE_2 \oplus IE_3 \oplus IE_4)$$

We made the hypothesis (Chapter 2) that two input events in a disjunction do not occur together, hence $IE_4.(IE_1 + IE_2 + IE_3) = 0$. By developping the event part under this hypothesis, the Boolean formula is rediscovered using exclusive ORs:

$$G(OE_i) = IE_1.IE_2 \oplus IE_1.IE_3 \oplus IE_4$$

3.3.2 Application

The filter is applied to the DCM of the MSS; the filtered DCM (size 86×60) is shown in Figure 3.9. 49.5% of the non-zero cells are ruled as noisy; the mean number of non-zero celles is now 5.25 per column, maxing out at 21. For instance, 26 of the 28 cells of the column corresponding to the output event 1Y04_0 are noisy; only 1B06_1 and 1S10_1 remain, as expected. Besides, the denoising is validated by the fact that the noisy input events actually belong to different stations, and were not related to 1Y04_0.

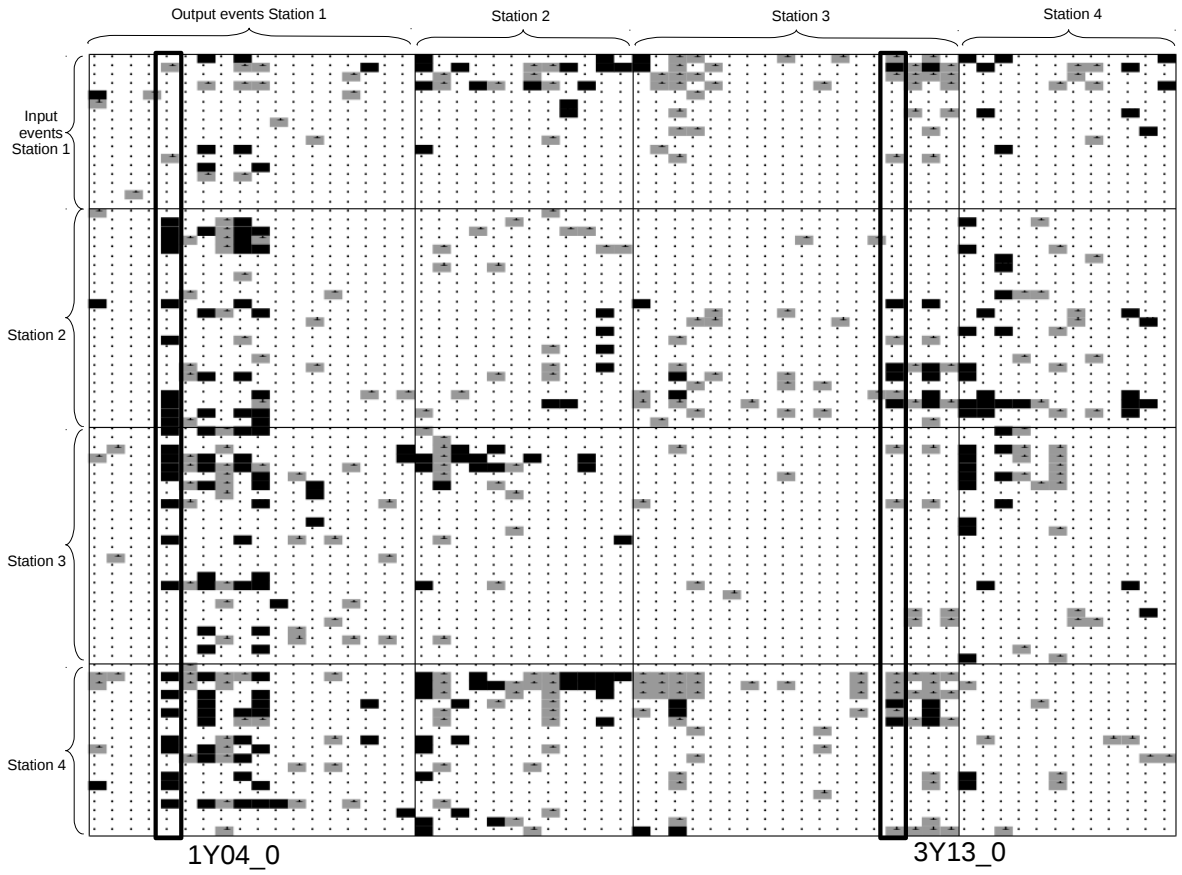


Figure 3.9: Result of the Filter on the DCM of the MSS. Grey cells correspond to validated non-empty cells, and black cells to noise.

All OEFFs of the MSS can be computed. It can be seen that some columns are well denoised, whereas some columns remain unchanged. Figure 3.10 exhibits a correlation between the sum of the columns and its denoising. Each dot represents an output event of the MSS (60 in total).

If the sum of the cells is greater than 1, at least one causal disjunction is likely to be discovered. The filter is efficient, getting rid of all the noise.

If the sum of the cells is lower than 1, necessarily the output event was observed alone at least once; it is impossible to find a direct cause anyway. However, the closer to one the sum is, the better the denoising is: a sum close to, but lower than 1 means that some input events form a sufficient but not necessary cause, *i.e.* the disjunction is not complete. These input events still absorb some noise.

Finally, for very low values of the sum, the denoising is ineffective. All values in the cells are low, so it is highly probable that all observations are noisy, but it can not be decided.

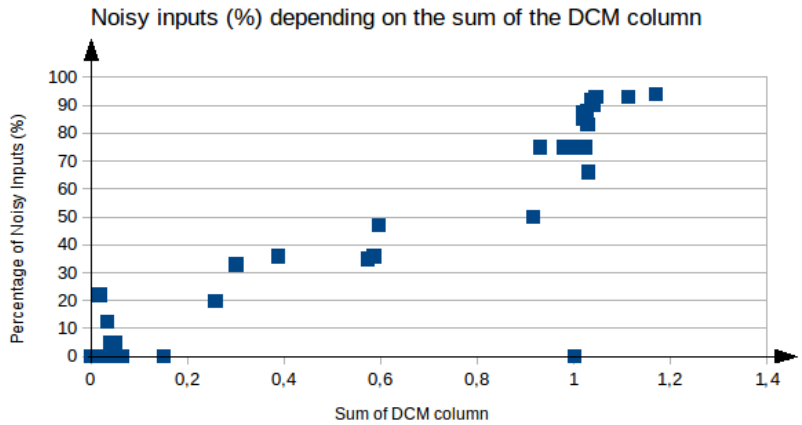


Figure 3.10: Efficacity of the filter: better denoising for column sums close to one.

In the intermediate case (sum lower than 1, but not close to 0), the denoising remains interesting, even though the event part of the OEFF is ε, because it is helpful in building the level part. Recall that to build the disjunctions in the level part F of the OEFF, only input levels, whose corresponding cells in the DCM are non-empty, are considered. Discarding noisy input events in the DCM therefore implies discarding irrelevant input levels to build the level part.

Example 3.3. *The output 3Y13_0 of the MSS is considered. The sum of its DCM column is 0.563. Its firing function without denoising is:*

$$\begin{aligned} F(3Y13_0) =& (\overline{1B06} \oplus 1B06).(\overline{2B21} \oplus 2B21).(\overline{4B11} \oplus 4B11). \\ & (4B10 \oplus \overline{4B11} \oplus 3B06 \oplus 3B11 \oplus 2B17 \oplus \overline{1B06} \oplus 1S10). \\ & (\overline{4B10} \oplus 4B10).(\overline{4S07} \oplus 4S07).(4S06 \oplus 4B10 \oplus \overline{1B06} \oplus 1S10) \end{aligned}$$

With denoising, 35% of the 19 non empty celled are reduced to 0. The level part of the firing function is reduced to:

$$F(3Y13_0) = (\overline{4S07} \oplus 4S07)$$

To conclude, the filtering brings two improvements to the computation of the OEFF. First, the computability of the event part of an OEFF, exponential in number of non-empty cells of the column, is increased by reducing the number of non-empty cells. The filter is only inefficient when the event part is mandatory ε, in which case the calculation is not required. In this case, the quality of the level part is still increased and more relevant, as noisy input levels are discarded as well.

3.4 Transitions reduction

The reduction method proposed in this section takes place after the building of the observable part. The objective of the reduction is to remove unfrequent transitions from the net, corresponding to spurious synchronizations of output events. Their occurrences in the firing sequence S must be replaced by equivalent firings, as presented in Figure 3.11.

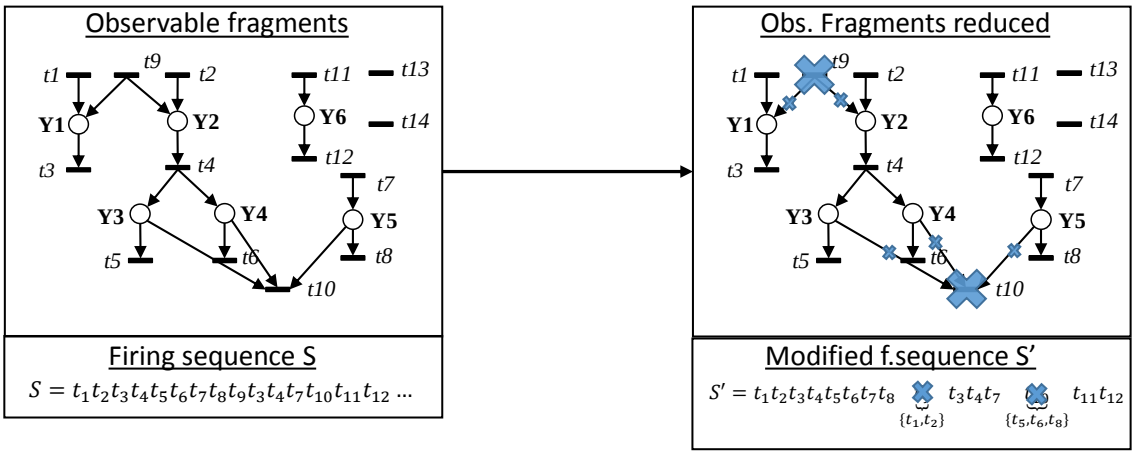


Figure 3.11: Principle of the reduction: remove transitions and replace in S

Beforehand, a slight change is brought to the building of transitions. Given a combination of output events, instead of creating a transition for each observed combination of consistent input events, a single transition is created, labelled by the combination of the OEFFs of the output events. This is illustrated by Figure 3.12. A direct consequence is the increase of permissivity of the model (since not all combinations allowed by the OEFF are observed, but allowed in the model). Here, the combination $IE_2 \bullet u_4$ was never observed, but is allowed by the more compact model.

This change decreases the structural complexity of the net by reducing its sheer size, but increases the complexity of the interpretation layer of the net.

3.4.1 Replacement of spurious transitions

Net reductions methods are used to simplify structure and markings of nets while conserving dynamical properties such as liveness or boundedness. Such reductions are developped to ease model analysis, by discarding details unrequired for analysis.

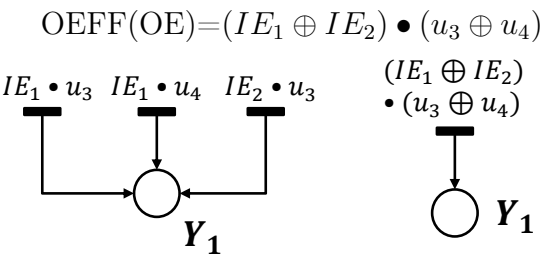


Figure 3.12: Increase of the permissivity of transitions by OEFF labelling

The reduction rule ϕ_T is proposed in [Desel and Esparza, 1995], based on a linear characterization of reducible transitions:

Definition 3.2. *Let $G = (P, T, C)$ be a net structure. A transition $t \in T$ is a nonnegative linearly dependant transition if there exists a vector $\Lambda \in \mathbb{Q}^{|T|}$ such that:*

$$\begin{cases} \Lambda(t) = 0 \\ C.\Lambda = C.t \\ \Lambda \geq 0 \end{cases}$$

Proposition 3.2 ([Desel and Esparza, 1995] Reduction rule ϕ_T)**.** *Let $G = (P, T, C)$ be a net structure. If $|T| \geq 2$, if there are no isolated transitions, and if t is a nonnegative linearly dependant transition, then t can be removed, forming a reduced net G'. The following equivalency stands:*

$$G \text{is well-formed} \Leftrightarrow G' \text{is well-formed}$$

This rule requires first no isolated transitions; isolated transitions correspond to zero-columns in the incidence matrix, and can therefore be deleted. In our case, only the observable behaviour is built yet, isolated transitions are created and labelled with unrelated input events D. These transitions are connected in the second step by unobservable places, and can therefore not be removed.

The characterization of [Desel and Esparza, 1995] provides a set of transitions, such that their firing is structurally equivalent to the one of the reduced transition. However, in the case of IPNs, the equivalency, in addition to being structural, must also be verified in terms of interpretation, as shown in the next example:

Example 3.4. *Consider the net of Figure 3.13 (observable part only). Using the linear characterization, the vectors $[t_2, t_3]$ and $[t_4, t_5]$ are structurally equivalent to t_1. However, firing t_4t_5 leads to the generation of output events $\{\downarrow Y_1, \uparrow Y_3\}$, whereas firing t_2t_3 generates $\{\downarrow Y_1, \uparrow Y_2, \downarrow Y_2 \uparrow Y_3\}$. Only $[t_4, t_5]$ generates the same output events as t_1. Besides, it is impossible to observe in the same event vector both the rising and the falling edge of an output, meaning that $[t_2, t_3]$ is definitely not an acceptable solution.*

The reduction rule is therefore extended by taking interpretation into account. First, the relevant interpretation of a transition is defined. Notice that in the previous example,

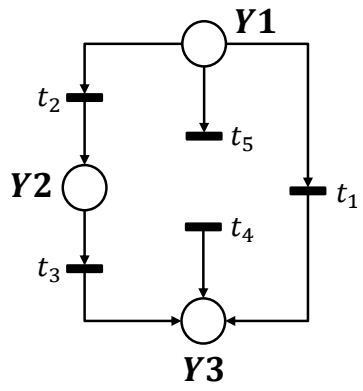


Figure 3.13: Example of an observable fragment, with t_1 reducible by $[t_4, t_5]$.

only output events were considered. A transition t is fired only if the input events in $\lambda(t)$ occur, and output events are generated according to marking changes in the connected observable places. When a transition is fired, its interpretation consists in three parts: (1) the output events generated; (2) the causal input events, belonging to the OEFFs of the output events; (3) unrelated input events, not issued from the OEFFs but from D. D is the set of input events that have not been causally associated to any output event, *i.e.* that do not appear in any firing function (See page 48). The causal input events (2) are a necessary and sufficient condition of the occurrence of output events (1). Whenever an output event (1) occurs, its causal input events are always present as well. Output events (1) and unrelated input events (3) are sufficient for the search of reductions.

Definition 3.3. *Let $t \in T$ be a transition. The reduced interpretation of t, denoted $RI(t)$ is defined as :*

$$RI(t) = \underbrace{\{\downarrow \varphi({}^\bullet t)\}}_{(1)} \cup \underbrace{\{\uparrow \varphi(t^\bullet)\}}_{(2)} \cup \underbrace{\{\lambda(t) \cap D\}}_{(3)}$$

(1) (resp (2)) is the set of falling (resp rising) edges of outputs associated to the observable pre-places (resp post-places) of t. (3) is the set of input events associated to unrelated inputs in the firing function of t.

The elementary events associated to a transition t are events that were observed during the same controller cycle (in the same event vector). It is impossible for the controller to change twice the status of an output in the same cycle. The sets $\varphi({}^\bullet t)$ and $\varphi(t^\bullet)$ are necessarily disjoint. Notice also that two transitions can not have the same RI, resulting from the permissivity increase: transitions with the same RI have been merged under a more permissive labelling.

Example 3.5. *Consider the transition $t1$ in the observable fragment of Figure 3.14. $\{\downarrow \varphi({}^\bullet t1)\} = \{\downarrow Y1\}$, $\{\uparrow \varphi(t1^\bullet)\} = \{\uparrow Y2\}$ and $\{\lambda(t1) \cap D\} = \{\uparrow u1\}$. The RI sets are*

the following :

$$RI(t1) = \{\downarrow Y1, \uparrow Y2, \uparrow u1\}$$
$$RI(t2) = \{\uparrow Y2\}$$
$$RI(t3) = \{\downarrow Y1\}$$
$$RI(t4) = \{\uparrow u1\}$$
$$RI(t5) = \{\downarrow Y1, \uparrow Y2, \uparrow Y3 \uparrow u1\}$$
$$RI(t6) = \{\uparrow Y3\}$$

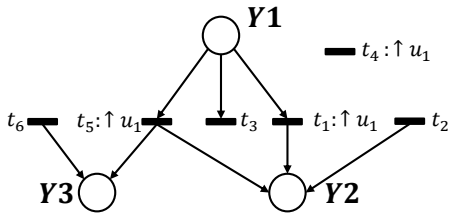


Figure 3.14: An observable fragment; only the unrelated input events are shown in the firing functions

If the behaviour of a transition, in terms of interpretation, is equivalent to the successive firings of multiple other transitions, such a transition could be replaced by said successive firing. For instance, should there be a token in $Y1$, firing $t1$ in Figure 3.14 has the same outcome as firing successively $t2$,$t3$ and $t4$, in any order. Indeed, the resulting marking would be the same ($Y2$ marked), and the generated associated events are the same ($RI(t1) = RI(t2) \cup RI(t3) \cup RI(t4)$).

This consideration leads to the notion of reducibility, as the behaviour expressed by $t1$ already exists in the net, and $t1$ could be replaced.

Definition 3.4. *Let $t \in T$ be a transition. t is said to be reducible if*

$$\exists \{t_1, \ldots, t_n\} \in 2^{T-\{t\}}, n \geq 2, \begin{cases} RI(t) = \bigcup_{i=1}^{n} RI(t_i) \\ \forall (i,j), i \neq j, RI(t_i) \cap RI(t_j) = \emptyset \end{cases}$$

The set $\{t_i\}_{1 \leq i \leq n}$ is called a reduction of t. $Red(t)$ designs the set of all reductions of t

Remark 3.1. *A few comments on this definition:*

- *A transition for which no reduction exists is called irreducible*
- *Transitions verifying /RI/=1 are irreducible*
- *All transitions in a reduction $\{t_i\}$ are necessarily different, due to the disjunction of the $RI(t_i)$ sets.*

- *For each event $e \in RI(t)$, given a reduction $\{t_i\}$ of t, there exists exactly one transition $t_i \in \{t_i\}$ such that $e \in RI(t_i)$. The sets $RI(t_i)$ are a partitioning of $RI(t)$*

In Figure 3.14, $\{t2, t3, t4\}$ is the only reduction for $t1$. $t5$ is reducible as well, by $\{t1, t6\}$. All other transitions are irreducible.

Transitions such as $t1$ and $t5$ are typically created when a spurious synchronized reading occurs. The firing conditions of $t2, t3, t4$ were simultaneously enabled during two input readings, leading to the simultaneous occurrence of their consequential output events (or unrelated input event for $t4$). Notice that $\{t1, t6\}$ is a reduction of $t5$, but $t1$ is itself reducible, therefore $\{t2, t3, t4, t6\}$ is a reduction of $t5$ as well:

Proposition 3.3. *Let $\{t_1, t_2, \dots t_n\}$ be a reduction of t and $\{t_{11}, t_{12}, \dots, t_{1m}\}$ a reduction of t_1. Then, $\{t_{11}, t_{12}, \dots, t_{1m}, t_2, \dots t_n\}$ is also a reduction of t.*

Proof. $\{t_{11}, t_{12}, \dots, t_{1m}\}$ is a reduction of t_1, therefore $RI(t_1) = \bigcup_{j=1}^{m} RI(t_{1j})$. Hence:

$$\begin{aligned} RI(\{t_{11}, t_{12}, \dots, t_{1m}, t_2, \dots t_n\}) = & \ (\bigcup_{j=1}^{m} RI(t_{1j})) \cup (\bigcup_{i=2}^{n} RI(t_i)) \\ = & \ RI(t_1) \cup (\bigcup_{i=2}^{n} RI(t_i)) \\ = & \ RI(t) \end{aligned}$$

Besides, $\forall\ 1 \leq j \leq m, RI(t_{1j}) \subset RI(t_1)$ and $\forall\ 2 \leq i \leq n, RI(t_1) \cap RI(t_i) = \emptyset$.
Hence, $\forall\ 1 \leq j \leq m, \forall\ 2 \leq i \leq n, RI(t_{1j}) \cap RI(t_i) = \emptyset$
$\{t_{11}, t_{12}, \dots, t_{1m}, t_2, \dots t_n\}$ is indeed a reduction of t. □

It is possible to extend reductions, until irreducible reductions are reached, *i.e.* reductions who consist only in irreducible transitions. Each extension adds at least one transition to the reduction. However, recall that spurious transitions are associated to synchronizations of concurrent processes by the controller, and are unfrequent. Synchronizing two concurrent processes is a rare event, synchronizing three is even less probable, etc... Irreducible reductions are maximal in number of transitions, as they can not be further extended. Instead, minimal reductions are more relevant, as they represent a minimal number of processes synchronized.

Definition 3.5. *A minimal reduction of a transition t is a reduction $\{t_i\}_{1 \leq i \leq n} \in Red(t)$, of size n, such that:*

$$\forall \{t_j\} \in Red(t), |\{t_j\}| \geq n$$

Remark 3.2. *Minimal reductions are not unique. Consider the net of Figure 3.15. Two reductions exist for t_5: $\{t_1, t_3\}$ and $\{t_2, t_4\}$. Both are minimal, equivalent, and can be chosen as reductions for t_5.*

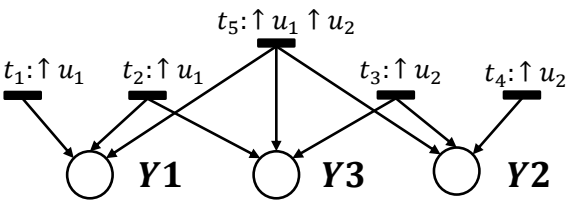


Figure 3.15: Non-unicity of minimal reductions illustrated by t_5

For now, reducible transitions have been defined by equivalency of interpretation. This definition is more restrictive than the linear caracterization of [Desel and Esparza, 1995], which is a consequency of Definition 3.4:

Proposition 3.4. *Let $N = (P_{Obs}, T, C)$ be the net structure, $t \in T$ a reducible transition, and $\{t_i\}_{1 \leq i \leq n} \in Red(t)$ a reduction of t. Define*

$$\Lambda : t_j \in T \rightarrow \begin{cases} 1 \text{ if } t_j \in \{t_i\}_{1 \leq i \leq n} \\ 0 \text{ otherwise} \end{cases}$$

Then (1) $\Lambda(t) = 0$, (2) $\Lambda \geq 0$, (3) $C.\Lambda = C.t$

Proof. (1) and (2) are straightforward.

Let $m = |P_{Obs}| = |\mathbb{Y}|$ be the number of outputs. For each $j \in [\![1, m]\!]$, $(C.\Lambda)_j$ can take three values: 1 if output event $\uparrow Y_j$ occurs, -1 if $\downarrow Y_j$ occurs, and 0 if no event occurs. Λ is built on a reduction of t, such that $RI(\{t_i\}) = RI(t)$, hence the output events are the same, and $C.\Lambda = C.t$. □

When removing a reducible transition, its firings in S must be replaced; minimal reductions offer an adapted set of transitions for replacement. It remains to prove that the replacement is possible.

Proposition 3.5. *Let $\{t_i\}_{1 \leq i \leq n}$ be a reduction of t. Then, there exists a firing sequence σ, containing exactly one firing of each transition of $\{t_i\}$, that always results in the same marking evolution as the firing of t, i.e.:*

$$\exists \sigma = t_1 t_2 \ldots t_n \mid \forall (M, M'), M \xrightarrow{t} M' \iff M \xrightarrow{\sigma} M'$$

σ is called a replacement for t

The proof is in Appendix B. If a transition t is reducible, then its occurences in the observed firing sequence S can be replaced by the firing sequence σ, which is behaviourally equivalent. σ is however not unique. In fact, t can be subsituted by every permutation of σ.

Proposition 3.6. *Let $\{t_i\}_{1 \leq i \leq n}$ be a reduction of t, and σ one replacement. Then, any permutation σ' of σ is also a replacement for t.*

Proof. Any permutation being a composition of elementary transpositions, it is sufficient to prove that σ' is a replacement for t when it is obtained by an elementary transposition from σ. An elementary transposition is the exchange of two successive transitions in σ. Let $\sigma = \dots t_i t_j \dots$ be a replacement and suppose that $\sigma' = \dots t_j t_i \dots$ can not be fired. It means that in σ, the firing of t_i is required to enable t_j. Hence there exists a place p such that $t_i \in {}^\bullet p$ and $t_j \in p^\bullet$.This place is observable, hence corresponds to an output event Y_p. The firing of t_i and t_j hence correspond to the events $\uparrow Y_p$ and $\downarrow Y_p$. σ being a replacement for t $\{\uparrow Y_p, \downarrow Y_p\} \subset RI(t)$. However, $\varphi({}^\bullet t) \cap \varphi(t^\bullet) = \emptyset$, so $\uparrow Y_p$ and $\downarrow Y_p$ can not both belong to $RI(t)$, hence a contradiction. Necessarily, σ' is firable. □

If a spurious transition is removed, it expressed a synchronization of concurrent processes. The ordering of the processes is therefore not relevant, and any permutation can be chosen as replacement in S.

In the observable net, transitions are classified into reducible and irreducible transitions. However, not every reducible transition should be actually reduced, as synchronizations are not all spurious. We suppose here that the number of similar executions of the process performed during the observation is known, called #Exec. For instance, data was recorded for the MSS during 20 production cycles of 24 gears each, hence #Exec=20. If no knowledge on the observation is known, #Exec=1.

Consider now a reducible transition; the existence of such a transition might have been caused by a spurious synchronized reading. Occurences of such spurious readings are scarce. If the associated events did not occur together at least once per execution, it is reasonable to assume that their simultaneous occurence was caused by a synchronized reading. The rule for removal of a transition is therefore the following:

Reduction rule *Let t be a transition, and S be the observed firing sequence. If t is reducible (Def 3.4), and the number of occurences of t in S is lower than #Exec, then:*

- *t is removed from the net*
- *Every occurrence of t in S is replaced by any permutation (Prop 3.6) of a replacement σ (Prop 3.5) built on a minimal reduction of t (Def 3.5).*

The modified firing sequence S is firable in the resulting simplified net (Prop 3.4).

3.4.2 Application

The hardest part of the reduction procedure is to actually find a reduction of a given transition, if it exists. Formally, given t, the aim is to find a set of transitions $\{t_i\}$ such that the $RI(t_i)$ are a partition of $RI(t)$. Recall that given an event set RI, there exists at most one transition t_j such that $RI(t_j) = RI$. The maximal number of possible sets $\{t_i\}$ is therefore given by the Bell number, expressing the number of partitions of a set of size $|RI(t)|$:

$$B_{|RI(t)|} = \sum_{k=1}^{|RI(t)|} S(|RI(t)|, k)$$

where $S(|RI(t)|, k)$ is the Stirling number of the second kind:

$$S(|RI(t)|, k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} C_k^j j^{|RI(t)|}$$

Even though the number of partitions is theoretically huge, a reduction can in practice be computed. First, the maximal value of $|RI(t)|$ is never so high. Recall that $|RI(t)|$ mirrors the number of output events (and unrelated input events) that occurred in the same event vector, *i.e.* whose causes occurred within the duration of a controller cycle (max 15ms). Typically, for the MSS, $max(|RI(t)|) = 5$. Since $B_5 = 52$, reductions are perfectly computable. Second, even if $|RI(t)|$ rises, the aim is set on minimal reductions, *i.e.* partitions with a minimal number of transitions. In most cases, testing all partitions is not required. It is sufficient to test size2 partitions, then size3 partitions, ..., until a solution is reached.

The full reduction procedure is described by Algorithm 3.2. The transitions are treated by decreasing sizes of $RI(t)$. The notation T_N designs the set of transitions t verifying $|RI(t)| = N$.

Algorithm 3.2 Reduction

Require: Observable net $(P_{Obs}, T, C), S, \#Exec$
Ensure: Reduced net

```
1:  N = max_t(|RI(t)|)
2:  while N ≥ 2 do
3:     for t ∈ T_N do
4:        {Take a transition t}
5:        if Count(T, S) ≤ #Exec then
6:           {If it is unfrequent enough}
7:           Compute {t_i} minimal reduction of t
8:           if {t_i} exists then
9:              Remove t from T
10:             Replace t by σ = t_1 t_2 ... t_n in S
11:          end if
12:       end if
13:       N ← N - 1
14:    end for
15: end while
```

The condition in the While loop is always reached, and transitions belonging to T_1 are irreducible by nature, hence do not required to be studied by Algorithm 3.2. The hardest step is at line 7, theoretically running at worst in $\mathcal{O}(B(N))$.

Example 3.6 (Example 3.5 cont.)**.** *The following occurences are introduced:* $\#t_1 = 8; \#t_2 = 15; \#t_3 = 15; \#t_4 = 15; \#t_5 = 4; \#t_6 = 16$, *and* $\#Exec = 10$. *The reduction starts at* $N = 4 = |RI(t_5)|$. *A minimal replacement of* t_5 *is* $\{t_1, t_6\}$. *Therefore,* t_5 *is deleted and replaced in* S *by* $\sigma = t_1 t_6$. *The new occurrences are* $\#t_1 = 12$ *and* $\#t_6 = 16$.

When the reduction moves on to $N = 3 = |RI(t_1)|$, t_1 is no longer unfrequent, and is kept. All other transitions are irreducible.

The reduction procedure is now applied to the MSS. All calculations have been run on a laptop (Intel® Core™ i5-3380M CPU @ 2.90GHz x4, 8Go RAM) from the data presented in Section 3.1. The models are generated with Graphviz[1], an open-source graph visualization software. Observable places are white, and labelled with the name of the associated output, while unobservable places are grey. If a place is initially marked, it is double-circled. The transitions are the rectangles, labelled on two rows: the first row is the event part F of the firing function, while the second is the level part G. These level parts are sometimes extensive, and are reduced to their first disjunction in the presented graphs for the sake of readability.

Before any improvement, only 25 outputs could be considered, and 402 transitions were built. The 5 remaining outputs (such as 1Y04_0) could not be computed because too many noisy input events were present. After the application of the filter and the permissivity increase, all 30 outputs of the MSS can be studied, and 295 transitions are built. The construction of this first observable model took 12s.

Figure 3.16 presents the observable model before the reduction. 29 of the 295 transitions are isolated, and the remainder of the transitions connect all the observable places, forming a huge spaghetti-like, unreadable fragment. Each observable place is connected in average to 23 transitions, maxing out at 48 for output 1Y06. This output is the relay of a magnet of the pusher station, actively sollicited during the arrival of new pieces in the chain. Due to its intense activity, it is often observed with other outputs.

After the application of the procedure for $\#Exec = 20$, 101 transitions remain. The observable fragments remaining are shown in Figure 3.17. 24 isolated transitions remain (the 5 transitions who were labelled with two events have been reduced), and the huge observable fragment is split into 13 fragments. Places are connected in average to 2.56 transitions, maxing out at 8 (always 1Y06). 9 of the 13 fragments possess only one place, and the biggest fragment consists in 8 places. This observable behaviour is much more understandable. Consider the top fragment of Figure 3.17: it consists mostly in outputs of the third station (labelled 3xxx), connected with some outputs of the second and the fourth, who are at the border of the third station. Likewise, the second biggest fragment consists mostly in outputs of the first station. The reduction took 3s, hence the full computation of the observable behaviour took 15s.

[1] http://graphviz.org

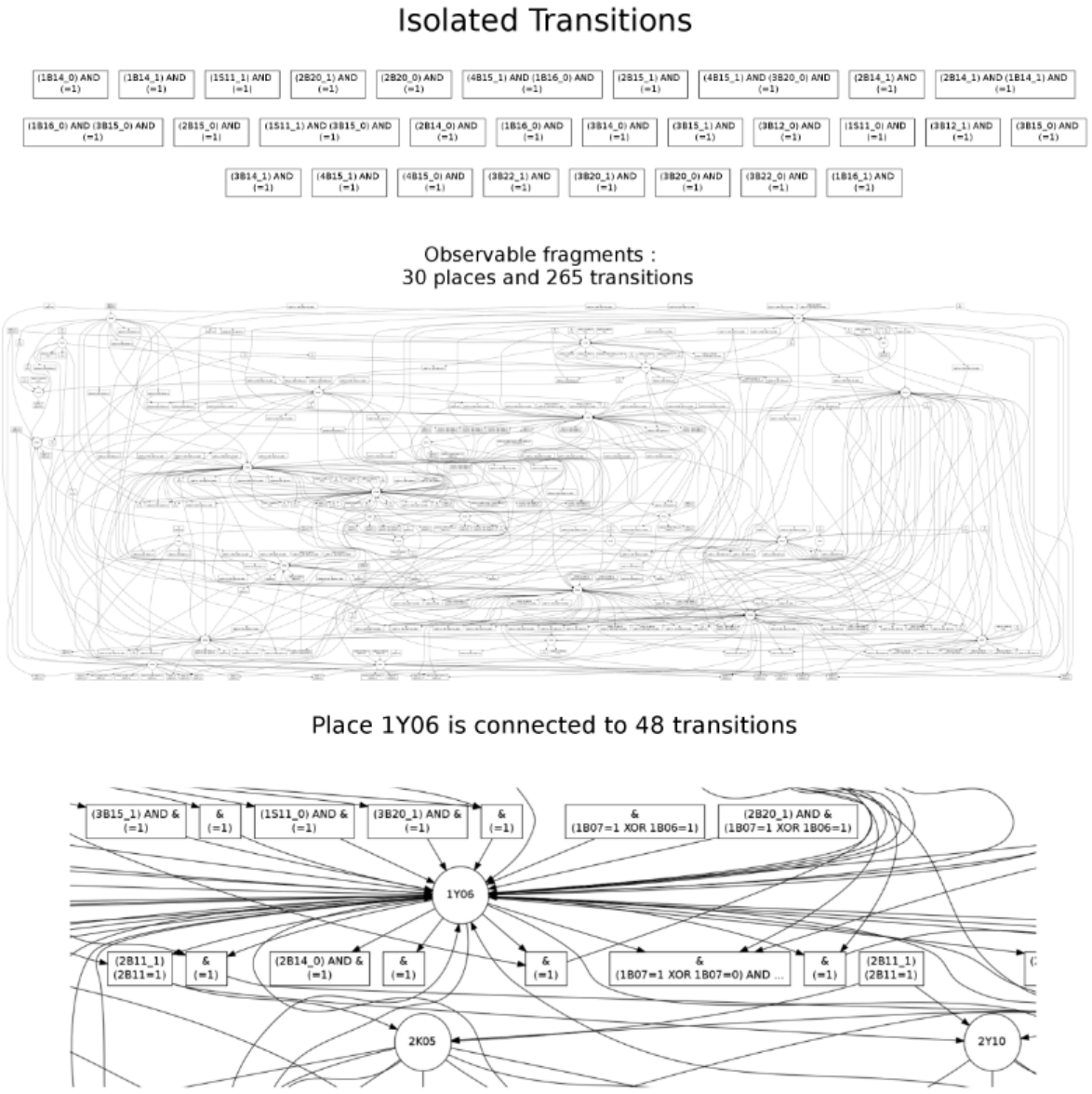


Figure 3.16: Observable Behaviour of the MSS pre-reduction: one single spaghetti fragment

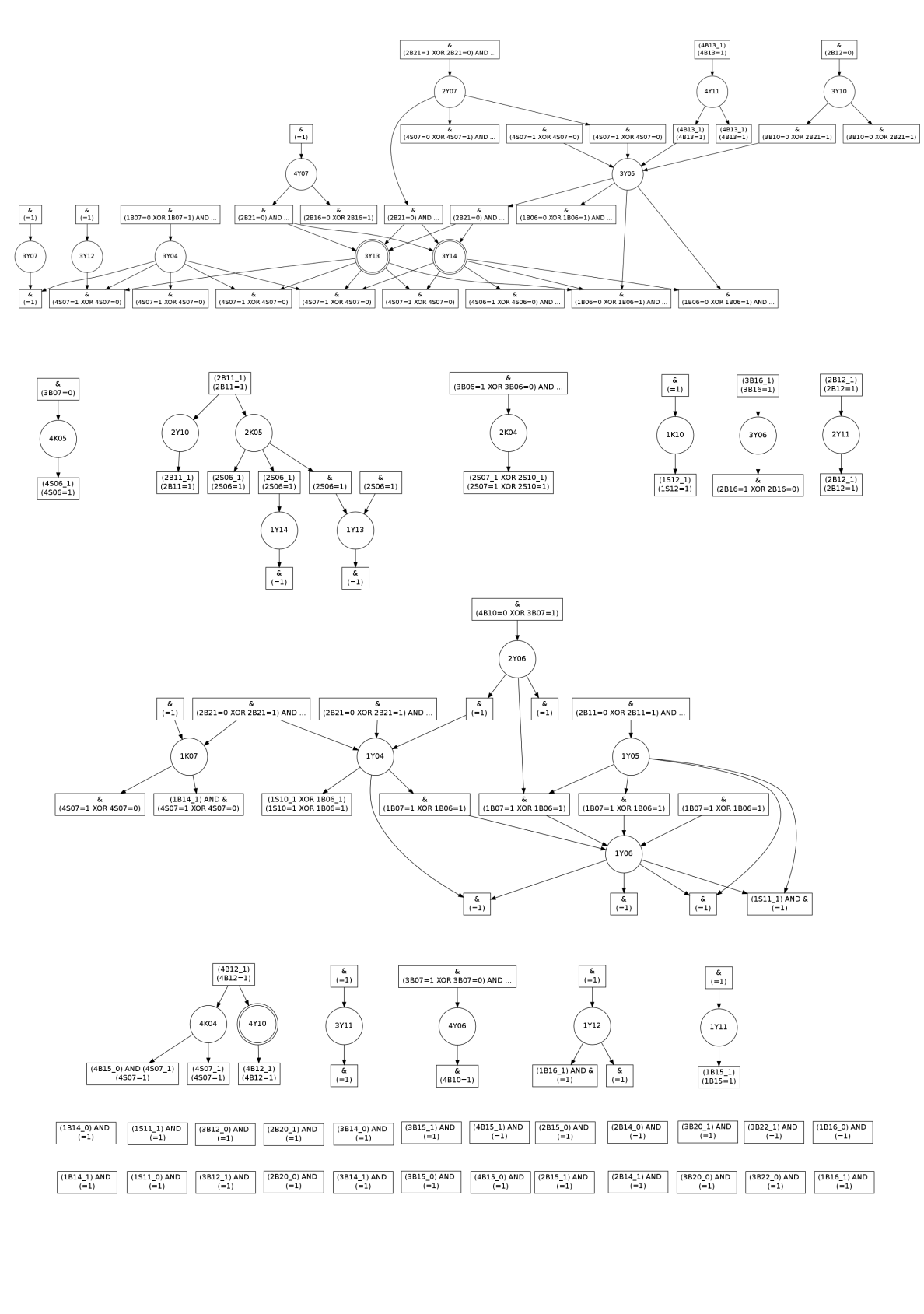


Figure 3.17: The result of the reduction for the MSS: 30 observable places and 101 transitions. 13 observable fragments and 24 isolated transitions

Recall Figure 3.3, where no convergence in the language observed of the MSS was found, even for $n = 1$. This issue was attributed to spurious synchronized readings. They are now taken into account in the discovery of the observable behaviour, and the observable model is corrected accordingly. The language growth curves are plotted again for the corrected model in Figure 3.18, for $n = 1, 2, 3$, this time based on words observed in the corrected firing sequence. Convergence is observed for $n = 1$, hence the correction is efficient.

The divergence still observed for $n \geq 2$ are due to the massive concurrency of the system. All length-2 interleavings have not been observed; Proposition 2.1 can not apply, and a rule-based approach will be incomplete. A new approach is therefore proposed in next chapter to deal with the unobservable discovery.

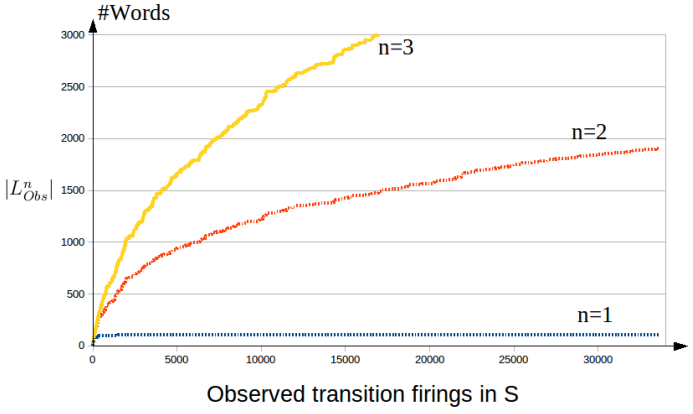


Figure 3.18: Observed language of the MSS expressed on the transitions, for $n = 1, 2, 3$

3.5 Conclusions

It has been shown in this chapter that the method proposed in Chapter 2 did not scale well. Massive concurrent systems are observed with spurious synchronizations from the controller, which hindered the application of the previous method. This chapter proposed two improvements to limit the noise induced by these synchronizations. A filter is developped to improve the computation of the OEFFs, both in quality of the result and calculus cost. Spurious transitions are also detected and reduced, leading to a simpler observable part.

It remains to add the unobservable behaviour to this simplified observable part, which is the purpose of next chapter.

Chapter 4
Discovery of the unobservable behaviour

Introduction

This chapter proposes a new approach to discover the unobservable behaviour of the system, and express it as unobservable places. This approach is robust to concurrency, despite the incompleteness of the observation. Firstly, the discovery problem is settled and the hypotheses precised in Section 4.1. The theoretical results characterizing unobservable places are given in Section 4.2. Then, since the solution is not unique, quality criteria are introduced in Section 4.3 to choose a model adequate for reverse-engineering. Based on these criteria, Section 4.4 presents a heuristic to conduct the discovery and achieve good models. Section 4.5 proposes some extensions to the main contribution, while Section 4.6 illustrates its efficiency with examples. Finally, Section 4.7 proposes a more accurate comparison to other similar approaches.

4.1 Problem statement

The observable part of the model $G^{Obs} = \{P^{Obs}, T, I^{Obs}, O^{Obs}\}$ and the firing sequence S are the input data. To obtain a full Petri Net structure, unobservable places remain to be added with their edges to connect the transitions and an initial marking must be inferred, to ensure that the structure can reproduce the observed sequence. The inference problem is defined as follows:

Unobservable Behaviour Inference Problem

Given a set of transitions $T = \{t_1, \ldots, t_n\}$ and $S \in T^*$ a finite observed sequence, compute a Petri Net structure $G^{Unobs} = \{P^{Unobs}, T, I^{Unobs}, O^{Unobs}\}$ and an initial marking M_0 such that S is firable in (G^{Unobs}, M_0), and verifiying the following assumptions:

1. G^{Unobs} is ordinary

2. (G^{Unobs}, M_0) is 1-bounded

3. There are no silent transitions, *i.e.* transitions labelled with an event not appearing in the sequence S
4. G^{Unobs} is a free-labelled Petri Net, *i.e.* two transitions of T can not have the same label
5. G^{Unobs} is selfloop-free

The first four assumptions are consequences of the set of transitions being built by the observable behaviour discovery step. The transitions are associated with events, therefore can not be silent; observable places are associated with binary outputs, hence having multiple tokens residing in one place is forbidden. The fifth assumption is a working hypothesis. It is also worth noting that the number of places $|P^{Unobs}|$ is unknown *a priori*.

Notice that the observable places do not intervene in the formulation of the inference problem, as only the transitions and the firing sequence are required. The upcoming approach proposed to solve it aims at being generic, therefore the theoretical concepts behind the resolution do not require the observable places. They will instead be required in the practical application, to build the complete IPN G by merging G^{Obs} and G^{Unobs}.

4.2 Theoretical background

When the inference problem is stated for a given set of transitions T and a sequence S, an "empty" net with neither places ($P = \emptyset$) nor edges, but only the transitions T, is generated, the observable places being set aside for now. This net is a solution and is the starting point of the inference procedure. To solve the problem, one should aim at adding places to such a net, with their edges, which are *admissible*:

Definition 4.1. *Let $N = ((\{P, T, I, O\}, M_0)$ be a PN solution of the inference problem.*

Let $N' = (\{P', T, I', O'\}, M'_0)$ be a PN constructed from N by adding a place p with its edges $i[T]$ and $o[T]$ and its initial marking m_p, i.e. :

$$P' = P \cup \{p\}$$
$$I(O)' = \begin{bmatrix} I(O) \\ i(o)[T] \end{bmatrix}$$
$$M'_0 = \begin{bmatrix} M_0 \\ m_p \end{bmatrix}$$

Then p is said to be an admissible place if, and only if N' is also a solution.

4.2.1 From the firing sequence to admissible places

The main operator used in the procedure is the projecting operator, or projector, whose definition is recalled below from [Mazurkiewicz, 1995]:

Definition 4.2. *Let Σ and Σ_p be two alphabets such that $\Sigma_p \subseteq \Sigma$; and $S \in \Sigma^*$ a firing sequence. The projector Π_{Σ_p} is defined by:*

$$\Pi_{\Sigma_p}(S) = \begin{cases} \varepsilon, \ if \ S = \varepsilon \\ \Pi_{\Sigma_p}(a)t, \ if \ S = at, t \in \Sigma_p \\ \Pi_{\Sigma_p}(a), \ if \ S = at, t \notin \Sigma_p \end{cases}$$

$\Pi_{\Sigma_p}(S)$ is called the projection of S on Σ_p

Let $S \in T^*$ be a firing sequence, $(\Sigma_i, \Sigma_j) \in (2^T - \{\emptyset\})^2$ be two non-empty subalphabets of T such that $\Sigma_i \cap \Sigma_j = \emptyset$, and $\Pi_{\Sigma_i \cup \Sigma_j}$ the projector on $\Sigma_i \cup \Sigma_j$, written Π for simplification. Then, the generic form of the result of the projection is:

$$\Pi(S) = \sigma_i^1 \sigma_j^1 \sigma_i^2 \sigma_j^2 \ldots \ if \ \Pi(S)_1 \in \Sigma_i$$
$$\Pi(S) = \sigma_j^1 \sigma_i^1 \sigma_j^2 \sigma_i^2 \ldots \ if \ \Pi(S)_1 \in \Sigma_j$$

where $\sigma_i^k \in \Sigma_i^+$ is a finite nonempty sequence of transitions from Σ_i (same for $\sigma_j^l \in \Sigma_j^+$).

Example 4.1. *Consider the following sequence S and the alphabets $\Sigma_1 = \{t_1\}$ and $\Sigma_2 = \{t_2, t_3\}$.*

$$S \ = t_1 \ t_2 \ t_3 \ t_1 \ t_4 \ t_2 \ t_1 \ t_2 \ t_3 \ t_1 \ t_2 \ t_4 \ t_1 \ t_3 \ t_2 \ t_1$$

It comes:

$$\Pi(S) = \underbrace{t_1}_{\sigma_1^1} \underbrace{t_2 \ t_3}_{\sigma_2^1} \underbrace{t_1}_{\sigma_1^2} \quad \underbrace{t_2}_{\cdots} t_1 \ t_2 \ t_3 \ t_1 \ t_2 \quad t_1 \ t_3 \ t_2 \ t_1$$

For different alphabets $\Sigma_1 = \{t_1\}$ and $\Sigma_3 = \{t_3, t_4\}$:

$$\Pi(S) = \underbrace{t_1}_{\sigma_1^1} \quad \underbrace{t_3}_{\sigma_3^1} \underbrace{t_1}_{\sigma_1^2} \underbrace{t_4}_{\cdots} \quad t_1 \quad t_3 \ t_1 \quad t_4 \ t_1 \ t_3 \quad t_1$$

For the second case, notice that each subsequence σ_1^k or σ_3^k consists in only one firing. It can be inferred from such a firing sequence that the firing of t_1 always requires the firing of either t_3 or t_4 to be enabled again; the reverse can be inferred as well. To discover such relations between transition sets, the interest lies in the length of the subsequences computed from the projection.

Definition 4.3. *Let $\Pi(S) = \sigma_i^1 \sigma_j^1 \sigma_i^2 \sigma_j^2 \ldots$ be the result of the projection of S on $\Sigma_i \cup \Sigma_j$. If, $\forall k > 0, \forall l > 0, |\sigma_i^k| = |\sigma_j^l| = 1$, the alphabets Σ_i and Σ_j are said to be mutually dependant, written $\Sigma_i \leftrightarrows \Sigma_j$.*

The set of mutual dependencies is denoted $\{\leftrightarrows\} = \{(\Sigma_i, \Sigma_j) | \Sigma_i \leftrightarrows \Sigma_j\}$

Proposition 4.1. *Mutual dependency is a reflexive and symetric relation.*

The proof is straightforward.

Since each of the σ_i^k is a subsequence of length 1, it contains only 1 transition $t_i \in \Sigma_i$ and can therefore be assimilated as $\sigma_i^k = t_i^k \in \Sigma_i^1$. The mutual dependancy of two

alphabets Σ_i and Σ_j ensure that the projection of the sequence exhibits a strict firing alternance between transitions of both sets. Mutual dependency is a generalization of the notion of Systematic Precedence, from Definition 2.4, to sets of transitions. Namely, if $|\Sigma_i| = |\Sigma_j| = 1$,

$$(t_i \leftrightarrows t_j) \Leftrightarrow (t_i \in SP(t_j) \wedge t_j \in SP(t_i))$$

When a mutual dependency exists, it can be associated to a PN structure fragment involving two places, as illustrated by the following example:

Example 4.2 (Example 4.1 cont.)**.** *The projection on $\{t_1\} \cup \{t_3, t_4\}$ reveals a mutual dependency. t_1 is required to fire either t_3 or t_4, a place p_1 is created such that ${}^\bullet p_1 = \{t_1\}$ and $p_1^\bullet = \{t_3, t_4\}$ to express this dependency. Similarly, t_3 or t_4 is required to fire t_1; a place p_2 is created such that ${}^\bullet p_2 = \{t_3, t_4\}$ and $p_2^\bullet = \{t_1\}$. The first transition fired in S being t_1, p_2 is given a token as initial marking, to enable t_1.*

Similarly, it can be seen that $\{t_1\} \leftrightarrows \{t_2\}$. Two places p_3 and p_4 are added to express this dependency. The resulting net can be seen in Figure 4.1. Notice that this net reproduces S, and is a solution. The concurrency between two processes synchronized by t_1 is easily captured.

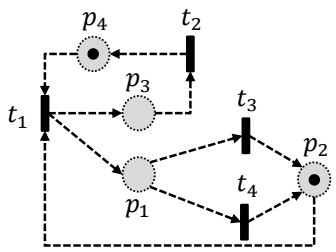


Figure 4.1: An unobservable PN built from S using mutual dependencies.

The previous example exhibits two cases of unobservable places which could be created from mutual dependencies. This can be generalized:

Definition 4.4. *Let Σ_i and Σ_j be two mutually dependent alphabets. The mutual dependency is associated to two places p_{ij} and p_{ji}, defined as following:*

$${}^\bullet p_{ij} = \Sigma_i; p_{ij}^\bullet = \Sigma_j; \begin{cases} M_0(p_{ij}) = 0 \ if \ \Pi(S)_1 \in \Sigma_i \\ M_0(p_{ij}) = 1 \ if \ \Pi(S)_1 \in \Sigma_j \end{cases}$$

$${}^\bullet p_{ji} = \Sigma_j; p_{ji}^\bullet = \Sigma_i; \begin{cases} M_0(p_{ji}) = 0 \ if \ \Pi(S)_1 \in \Sigma_j \\ M_0(p_{ji}) = 1 \ if \ \Pi(S)_1 \in \Sigma_i \end{cases}$$

The two places associated to $\Sigma_i \leftrightarrows \Sigma_j$ are shown in Figure 4.2.

Example 4.1 suggests that a net solution can be built from iteratively adding unobservable places associated to mutual dependencies. This result is shown in the following theorem, which is the main theoretical result [Saives et al., 2015a]:

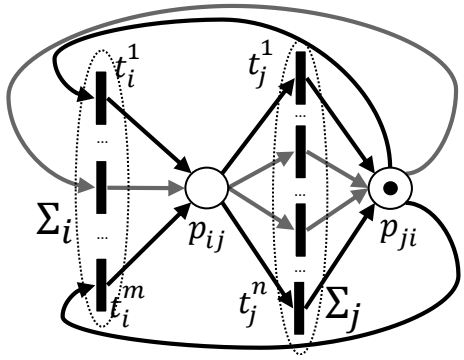


Figure 4.2: A PN structure composed of two places p_{ij} and p_{ji} for $\Sigma_i = \{t_i^1, \ldots, t_i^m\}$, $\Sigma_j = \{t_j^1, \ldots, t_j^n\}$. t_i^1 is the first transition fired.

Theorem 1. *Let $S \in T^*$ be a firing sequence, and (G, M_0) a 1-bounded Petri Net such that $S \in L(G, M_0)$. Let Σ_i and Σ_j be two alphabets in $(2^T - \{\emptyset\})$ ensuring $\Sigma_i \cap \Sigma_j = \emptyset$, and Π the projector on $\Sigma_i \cup \Sigma_j$. If $\Sigma_i \leftrightarrows \Sigma_j$, i.e.*

$$\Pi(S) = t_i^1 t_j^1 t_i^2 t_j^2 \ldots$$

or

$$\Pi(S) = t_j^1 t_i^1 t_j^2 t_i^2 \ldots$$

Then the net G' defined by the addition of places p_{ij} and p_{ji} to G is 1-bounded and $S \in L(G', M_0)$. In other words, places p_{ij} and p_{ji} associated to the mutual dependency are admissible.

Proof. Suppose that $\Pi(S)_1 \in \Sigma_j$ (*i.e.* the first case, the reasoning being the same in the other case).

1-boundedness:

G is 1-bounded. It remains to prove that the places places p_{ij} and p_{ji} are 1-bounded. They verify ${}^\bullet p_{ji} = p_{ij}^\bullet$, ${}^\bullet p_{ij} = p_{ji}^\bullet$, and $M_0(p_{ij}) + M_0(p_{ji}) = 1$. It ensures that $\forall M \in R(G', M_0), M(p_{ij}) + M(p_{ji}) = 1$, therefore G' is 1-bounded.

$\mathbf{S} \in \mathbf{L}(G', M_0)$:

Suppose that $S \notin L(G', M_0)$, and let t be the first transition that can not be fired. Since the transitions in $T - (\Sigma_i \cup \Sigma_j)$ have the same pre- and post-places in G' and in G, and t is firable in G, t must belong to $\Sigma_i \cup \Sigma_j$. Suppose that $t \in \Sigma_j$ (same reasoning for $t \in \Sigma_i$). Then p_{ji} is the only new place in ${}^\bullet t$ that can prevent t from firing, thus must be empty when t should be fired. Let k be an integer such that $\Pi(S)_{2k-1} = t$. Then, when $\Pi(S)_{2k-2} \in \Sigma_i$ was fired, p_{ji} was filled with a token. Since no other transition in Σ_j was fired between $\Pi(S)_{2k-2}$ and $\Pi(S)_{2k-1}$, p_{ji} still contains a token when t should be fired, leading to a contradiction. Therefore, $S \in L(G', M_0)$.

□

Whenever a mutual dependency is discovered between two sets of transitions, two places (Figure 4.2) with associated unweighted edges are admissible for the inference problem: they ensure the 1-boundedness of the resulting net, and that the firing sequence is still firable. Transitions are not duplicated nor created, and the condition $\Sigma_i \cap \Sigma_j = \emptyset$ ensures that no selfloops are created. Therefore, Theorem 1 provides a single rule that characterizes couples of admissible places.

Given a set of mutual dependencies $\{\leftrightarrows\}$, it is converted into a set of pairs of admissible places by Theorem 1. Any combination of these pairs is a solution. Theoretically, if the knowledge of mutual dependencies is complete, say $|\{\leftrightarrows\}| = n$, then all solutions are known: they are the combinations of the pairs of admissible places associated to $\{\leftrightarrows\}$, building 2^n net solutions. Some solutions have isolated transitions, the number of connected solutions is naturally lower than that number. The multiplicity of solutions is illustrated by the following example:

Example 4.3. *Let* $T = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ *be the alphabet, and* $S = t_6t_1t_7t_4\ t_6t_1t_7t_4\ t_6t_1t_2t_4\ t_6t_0t_7t_3\ t_6t_0t_2t_3\ t_6t_0t_2t_3\ t_6t_0t_7t_3\ t_6t_0t_7t_5t_4\ t_6t_0t_7t_3\ t_6t_0t_5t_2t_4\ t_6t_0t_5t_7t_4\ t_6t_1t_7$ *be the observed sequence. All mutual dependencies discoverable in this sequence are the following:*
$\{t_6\} \underset{1}{\leftrightarrows} \{t_2, t_7\}$, $\{t_6\} \underset{2}{\leftrightarrows} \{t_0, t_1\}$, $\{t_6\} \underset{3}{\leftrightarrows} \{t_3, t_4\}$, $\{t_5\} \underset{4}{\leftrightarrows} \{t_1, t_4\}$, $\{t_0\} \underset{5}{\leftrightarrows} \{t_3, t_5\}$, $\{t_2, t_7\} \underset{6}{\leftrightarrows} \{t_0, t_1\}$, $\{t_2, t_7\} \underset{7}{\leftrightarrows} \{t_3, t_4\}$, $\{t_0, t_4\} \underset{8}{\leftrightarrows} \{t_5, t_6\}$, $\{t_3, t_4\} \underset{9}{\leftrightarrows} \{t_0, t_1\}$,$\{t_6\} \underset{10}{\leftrightarrows} \{t_1, t_3, t_5\}$. *There are 10 mutual dependencies. Each of the* 2^{10} *combinations leads to a different net, which is a solution. For instance, two nets are presented in Figure 4.3; net (a) is build on dependencies {1,2,3,4,5} and net (b) on dependencies {2,3,4,5,6,7}. Implicit places*[1] *have been removed in both nets.*

Finally, in this case, any proper superset of {2,3,4,5,6,7} leads, after deletion of implicit places, to net(b). Namely, it means that net (a) can be transformed into net (b) by adding dependencies {6,7} to {1,2,3,4,5}.

However, to formulate the problem as choosing a set of places from all possible combinations, the knowledge of the mutual dependencies $\{\leftrightarrows\}$ is required. A look at the complexity of finding mutual dependencies, and converting them into places, is required beforehand.

4.2.2 Complexity of finding admissible places

To discover a mutual dependency, the required inputs are two disjoint non-empty subsets of T, generically designed as (Σ_i, Σ_j). Notice the symmetry in the roles of Σ_i and Σ_j. Therefore, unordered pairs $\{\Sigma_i, \Sigma_j\}$ are considered. For the remainder of this

[1] Implicit places are places whose removal does not alter the reachability graph, see Section 4.5.1 for more details

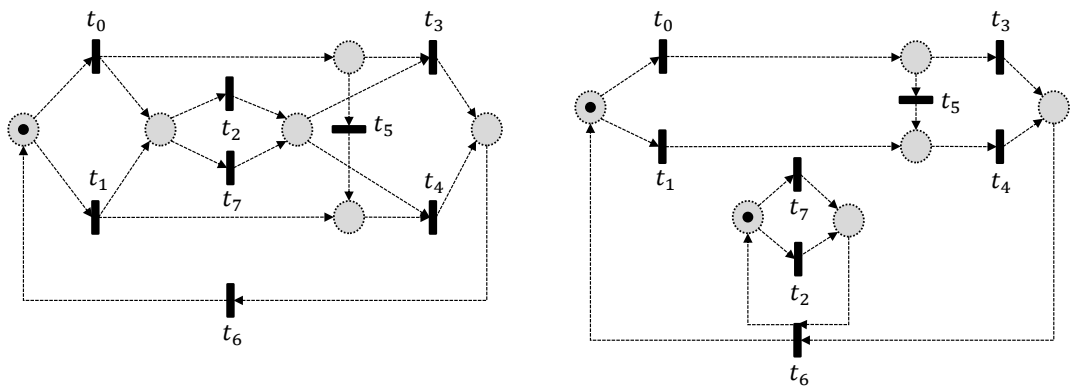


Figure 4.3: Two net solutions, built with different sets of admissible places

work, the notation $(2^T - \{\emptyset\})^2_{\not\emptyset}$ will design the set of all these unordered pairs:

$$(2^T - \{\emptyset\})^2_{\not\emptyset} = \{\{\Sigma_i, \Sigma_j\} | (\Sigma_i, \Sigma_j) \in (2^T - \{\emptyset\})^2 \wedge \Sigma_i \cap \Sigma_j = \emptyset\}$$

The problem is to build the set of mutual dependencies $\{\leftrightarrows\} \subseteq (2^T - \{\emptyset\})^2_{\not\emptyset}$. This is achieved by applying Algorithm 4.1. Given two alphabets $\{\Sigma_i, \Sigma_j\} \in (2^T - \{\emptyset\})^2_{\not\emptyset}$, the projected sequence is a sequence of subsequences $\sigma_i^k \in \Sigma_i^+$,$\sigma_j^l \in \Sigma_j^+$. The idea of Algorithm 4.1 is to evaluate the maximal length of these subsequences; if it is one, then a mutual dependancy between the alphabets is discovered. If a mutual dependancy is discovered, the first alphabet observed in the projected sequence is returned as well.

Proposition 4.2. *Algorithm 4.1 determines if two alphabets Σ_i and Σ_j are mutually dependant according to a sequence S. Let δ_i (resp δ_j) be the size of Σ_i (resp Σ_j), and $\delta = \delta_i + \delta_j$. Algorithm 1 runs in $\mathcal{O}(|S|.\delta)$*

Proof. For each element s_k of the sequence S (line 3), at least one test (line 4), and at most two tests (lines 4 and 15) are run. Verifying the belonging of s_k to Σ_i is at worst of complexity $\mathcal{O}(\delta_i)$; hence, if two tests are run, the loop (3-21) runs at worst in $\mathcal{O}(\delta)$ (since all other operations run in $\mathcal{O}(1)$).

Each element of the sequence is studied, hence the loop (3-21) is run |S| times. Finally, the complexity of Algorithm 4.1 is $\mathcal{O}(|S|.\delta)$. □

Notice that the disjunction of the alphabets implies that $\delta \leq |T|$. The complexity in the worst case is $\mathcal{O}(|T||S|)$. Hence applying Algorithm 4.1 is efficient because of its linear complexity regarding $|S|$ and $|T|$.

If the result of Algorithm 4.1 is True, two places are added to the net, according to Algorithm 4.2. The notation $\delta_{\Sigma_i}[T]$ designs a row vector such that $\delta_{\Sigma_i}[t] = 1$ if $t \in \Sigma_i$, 0 otherwise.

Proposition 4.3. *If the result of Algorithm 4.1 is True, Algorithm 4.2 adds two places p_{ij} and p_{ji} with their edges and initial marking, according to Theorem 1. Algorithm 4.2 runs in $\mathcal{O}(|S||T|)$.*

Algorithm 4.1 Verifying the conditions of Theorem 1

Require: $S = s_1 \dots s_{|S|}$ a sequence, $\{\Sigma_i, \Sigma_j\}$ two disjoint non-empty subsets of T.
Ensure: Decision for $\Sigma_i \leftrightarrows \Sigma_j$

1: {This part aims at calculating the maximal length of the subsequences σ_i^k, σ_j^l in the projected sequence. These lengths are memorized in the variables max_i and max_j. Furthermore, the alphabet to which belongs the first transition is memorized.}
2: $max_i = max_j = 0$; $counter_i = counter_j = 0$; $first = \emptyset$
3: **for** $s_k \in [\![1, |S|]\!]$ **do**
4: **if** $s_k \in \Sigma_i$ **then**
5: **if** $first == \emptyset$ **then**
6: $first \leftarrow \Sigma_i$
7: **end if**
8: **if** $counter_j \neq 0$ **then**
9: $max_j \leftarrow max(max_j, counter_j)$; $counter_i \leftarrow 0$
10: **end if**
11: $counter_i \leftarrow counter_i + 1$
12: **else if** $s_k \in \Sigma_j$ **then**
13: **if** $first == \emptyset$ **then**
14: $first \leftarrow \Sigma_j$
15: **end if**
16: **if** $counter_i \neq 0$ **then**
17: $max_i \leftarrow max(max_i, counter_i)$; $counter_i \leftarrow 0$
18: **end if**
19: $counter_j \leftarrow counter_j + 1$
20: **end if**
21: **end for**
22: {If the maximal lengths of max_i and max_j are 1, Σ_i and Σ_j are mutually dependant}

23: **if** $max_i == 1 \wedge max_j == 1$ **then**
24: **return** True, first
25: **else**
26: **return** False
27: **end if**

Proof. The tests and operations on lines (4,8,10) run in $\mathcal{O}(1)$, while operations on lines(5,6) run in $\mathcal{O}(|T|)$. There is a call to Algorithm 4.1 on line 2, hence running in $\mathcal{O}(|S||T|)$. The resulting complexity is the sum, hence $\mathcal{O}(|S||T|)$. □

Applying both algorithms for a given pair of subalphabets is not costly, which means that the construction of the net is easy if well-chosen pairs are given. In the generic case, it remains to evaluate the number of possible pairs. The following proposition evaluates the size of $(2^T - \{\emptyset\})^2_{/\emptyset}$, given an alphabet T:

Proposition 4.4. *Let Σ_n be a size n alphabet ($n \geq 2$). The number of pairs of disjoint, non-empty subsets of Σ_n, noted P_n, is given by the formula:*

$$P_n = 3^{n-2} + \sum_{k=0}^{n-3} 3^k (2^{n-k-1} - 1)$$

Algorithm 4.2 Adding places if Theorem 1 is satisfied

Require: $N = (\{P, T_{Obs}, I, O\}, M_0)$ a net satisfying the inference problem for $S = s_1 \dots s_{|S|}$ a sequence, $\{\Sigma_i, \Sigma_j\}$ two disjoint subsets of T_{Obs}.

Ensure: N' a net satisfying the inference problem for S

1: {If a mutual dependancy between Σ_i and Σ_j is discovered by Algorithm 4.1, two places p_{ij} and p_{ji} are added}
2: MutDep, FirstAlph $\leftarrow$ Algorithm 4.1(S,$\{\Sigma_i, \Sigma_j\}$)
3: **if** MutDep == True **then**
4: $\quad P \leftarrow P \cup \{p_{ij}\} \cup \{p_{ji}\}$
5: $\quad I \leftarrow \begin{bmatrix} I \\ -\delta_{\Sigma_i}(T) \\ -\delta_{\Sigma_j}(T) \end{bmatrix}$
6: $\quad O \leftarrow \begin{bmatrix} O \\ \delta_{\Sigma_j}(T) \\ \delta_{\Sigma_i}(T) \end{bmatrix}$
7: $\quad$ **if** FirstAlph == Σ_i **then**
8: $\quad\quad M_0 \leftarrow \begin{bmatrix} M_0 \\ 1 \\ 0 \end{bmatrix}$
9: $\quad$ **else**
10: $\quad\quad M_0 \leftarrow \begin{bmatrix} M_0 \\ 0 \\ 1 \end{bmatrix}$
11: $\quad$ **end if**
12: **end if**

The proof by recurrence is in Appendix B.

Given the size of the transition set $|T|$, there are roughly $|(2^T - \{\emptyset\})^2_{\not\emptyset}| \simeq 3^{|T|}$ pairs to be studied. Therefore, a complete study requires to apply Algorithm 4.2 $3^{|T|}$ times, leading to complexity $\mathcal{O}(|T||S|3^{|T|})$. This number quickly becomes untractable for reasonable values of |T| (for instance, 30 transitions represent roughly 10^{14} pairs to study).

4.2.3 Intermediate conclusions

This section gives an important complexity result, illustrated by Figure 4.4:

- Given two disjoint alphabets, finding their relationship and translating it into PN structure is easy. (Algorithm 4.2, $\mathcal{O}(|T||S|)$)
- However, the number of alphabets to be studied raises exponentially regarding the number of transitions ($\simeq 3^{|T|}$)

The main difficulty results therefore in building the set of mutual dependencies $\{\leftrightarrows\}$, whose size n is *apriori* unknown. Achieving the completeness of this set requires dealing with $3^{|T|}$ cases. The full knowledge should not be achieved; instead, a smaller subset $\{\leftrightarrows\}_{n'}$ of size $n' \leq n$ can be built. However, only $2^{n'}$ solutions remain (out of the 2^n possible solutions). Therefore $\{\leftrightarrows\}_{n'}$ should be judiciously elaborated, in order that a

$$(S,T) \underbrace{\longrightarrow}_{\sim 3^{|T|} choices} \overset{(2^T - \{\emptyset\})^2_{\not\emptyset}}{(\Sigma_i, \Sigma_j)} \underbrace{\longrightarrow}_{\substack{Algorithm\ 4.2 \\ \mathcal{O}(\delta.|S|)}} (N, M_0)$$

Figure 4.4: Visualisation of the complexity of the problem

'good' solution to the problem remains available. Suppose that this solution requires $k \leq n'$ dependencies to be built. To limit the computational effort, it is better to limit the size of $\{\leftrightarrows\}_{n'}$, so that $n' \simeq k$.

Therefore, given n' mutual dependencies, instead of having to choose any of the $2^{n'}$ possible solutions, only the solution using *all* n' dependencies is studied. If a new dependency is added, a new net is built using $n' + 1$ dependencies. The set of mutual dependencies $\{\leftrightarrows\}$ can be built iteratively by adding new dependencies, and checking the quality of the net at every step.

Example 4.4 (Example 4.3 cont.)**.** *Net (a) is built from the set* $\{\leftrightarrows\}_5 = \{1, 2, 3, 4, 5\}$. *Now, if dependencies {6,7} are added, net(b) is built from* $\{\leftrightarrows\}_7 = \{1, 2, 3, 4, 5, 6, 7\}$. *Any superset of* $\{\leftrightarrows\}_7$ *leads to net (b) as well after the deletion of implicit places.*

To sum things up, two issues remain to be dealt with:

- What is a 'good' net?
- How to ensure that the partial set of mutual dependencies $\{\leftrightarrows\}_{n'}$ leads to a 'good' net?

Recall that the purpose of the identified net is to be used for reverse-engineering. The first question leads to the definition of criteria to assess the quality of a net for reverse-engineering, and is dealt with in Section 4.3. The second question introduces a *limited discovery* problem: designing a relevant heuristic to partially explore $(2^T - \{\emptyset\})^2_{\not\emptyset}$ and build the partial subset of mutual dependencies $\{\leftrightarrows\}_{n'}$. The heuristic must naturally be result-oriented, according to the quality criteria; it is presented in Section 4.4.

4.3 Assessing the quality of a net

Numerous nets are solutions to the inference problem, so "what makes a good identified model ?". According to [Van der Aalst, 2011b], a good model in process discovery should balance four quality dimensions: *fitness, simplicity, precision* and *generalization.* Fitness expresses the ability of the identified model to replroduce the observed behaviour. Notice that in our approach, any solution must be guaranteed to reproduce S, hence fitness is guaranteed. Precision and generalization are opposite quality metrics that qualify the exceeding behaviour, *i.e.* the behaviour accepted by the net that has not

been observed. Finally, simplicity refers to *Occam's Razor*, stating that "the simplest explanation is usually the correct one".

Precision is an unavoidable quality of nets used for fault diagnosis, as minimizing the exceeding behaviour lowers the number of false alerts ([Roth et al., 2009a]. However, for reverse engineering models, the model is destined to be read, therefore simplicity takes the upper hand; a good model is one that is easily understandable.

Two metrics are introduced here for quantifying both the precision and the understandability of the net, then an example illustrating the importance of understandability is presented. Additional reflexions on the choice of metrics are presented in Appendix A.

4.3.1 Quality metrics

4.3.1.1 Precision

Precision qualifies the closeness of the behaviour of the identified net to the behaviour observed in the sequence. These behaviours are described by languages, respectively observed in the sequence, and generated by the identified net. Since the observation is finite, the observed language is necessary finite, whereas the identified language might be infinite (if the identified net is cyclic). A length parameter n is introduced to allow the comparison of the two languages.

Definition 4.5. *The language of length n generated by a Petri net N , i.e. the set of words of length n generated by N, or identified language is:*

$$L^n_{Id}(N) = \{w \in T^* | \ |w| = n \ \wedge \exists M' \in R(N), \ M \xrightarrow{w} M'\}$$

The language of length n, observed in a firing sequence S, is:

$$L^n_{Obs}(S) = \bigcup_{1 \leq t \leq |S|-n+1} s_t.s_{t+1} \dots s_{t+n-1}$$

The precision of the identified net is quantified by the size of its exceeding language:

$$L^n_{Exc}(N, S) = L^n_{Id}(N) \setminus L^n_{Obs}(S)$$

Since fitness is guaranteed by the theoretical results, $L^n_{Obs2}(S) \subseteq L^n_{Id2}(N)$, therefore $L^n_{Exc}(N, S) \geq 0$. The lower the size of the exceeding language, the more precise the net. However, recall the observation curves of Section 1.2.2; for values of n for which the convergence of the observed language was not observed, it is unreasonable to aim for perfect precision ($L^n_{Exc}(N, S) = 0$, which is the goal of synthesis approaches), since the observed behaviour is clearly not complete, and no counter-examples are available.

Nevertheless, precision can be maximized in the inference procedure. An interesting property of the identified nets is that the addition of any place to N reduces the size of the identified language, as shown in the following propositions. Languages considered in these propositions are not parametrized by n, potentially infinite: $L = \bigcup_{n=0}^{\infty} L^n$.

Proposition 4.5. *Let $N = \{(P, T, C), M_0\}$ be a PN, and $L_{M_0}(N) = \{S | M_0 \xrightarrow{S}\}$ the (infinite) set of sequences firable from the initial marking.*
Let $N' = \{(P', T, C'), M'_0\}$ be a PN constructed from N by adding a place p with its edges $i[T]$ and $o[T]$, such that $c_p[T] = o[T] - i[T]$, and its initial marking m_p, i.e. :

$$P' = P \cup \{p\}$$
$$C' = \begin{bmatrix} C \\ c_p[T] \end{bmatrix}$$
$$M'_0 = \begin{bmatrix} M_0 \\ m_p \end{bmatrix}$$

Then $L_{M'_0}(N') \subseteq L_{M_0}(N)$

The proof lies in Appendix B. The result remains identical when taking all marking into accounts, hence our definition of identified language.

Proposition 4.6. *In the same conditions as Proposition 4.5, $L_{Id}(N') \subseteq L_{Id}(N)$, where $L_{Id}(N) = \bigcup_{M \in R(N)} \{S | M \xrightarrow{S}\}$*

Proof. Let $w \in L(N')$ be a word firable in N'. Let M' be an accessible marking of N' such that M' enables w. M' being accessible, let $\sigma \in T^*$ be a firing sequence such that $M'_0 \xrightarrow{\sigma} M' \xrightarrow{w}$.

Then σw is a word of $L_{M'_0}(N')$, thus of $L_{M_0}(N)$ according to Proposition 4.5. σw being firable in N, there is a marking $M \in R(N)$ such that $M_0 \xrightarrow{\sigma} M \xrightarrow{w}$, and M is an accessible marking of N that enables w. Therefore $w \in L(N)$. □

Proposition 4.6 is extremely generic: whatever the place-centered structure added (weight and number of edges, initial marking), the language generated by the net is reduced, or remains identical. It can be applied to the resolution of the inference problem: the more admissible places are discovered, the smaller the identified language will be. Consequently:

Proposition 4.7. *If all disjoint non-empty subalphabets $\{\Sigma_i, \Sigma_j\} \in (2^T - \{\emptyset\})^2_{\not\cap}$ have been tested with Algorithm 4.2, then $\{\leftrightarrows\}$ is maximal and the resulting net is the most precise.*

Proof. Each couple of added places reduced the size of the identified language. If all alphabet couples have been tested, all mutual dependencies have been discovered, and all possible admissible places have been added to the net. No more places can be added, therefore its identified language can not be further reduced by the addition of places. The resulting net is the most precise solution. □

Since $(2^T - \{\emptyset\})^2_{\not\cap}$ must be fully explored, finding the most precise net requires exponential time regarding transitions. However, optimal precision is not the main quality criterion, so this is not an issue.

4.3.1.2 Understandability and structural complexity

Understandability of the net is related to the difficulty of reading the net, namely picturing the possibilities of token displacements. This difficulty increases with concurrent transitions and conflict places (who are connected to multiple edges). The Coefficient of Network Complexity (CNC), originally introduced in ([Pascoe, 1966])is chosen: it is the ratio edges/nodes of the graph.

Definition 4.6 (adapted from [Pascoe, 1966]). *For a Petri net structure $G = (P, T, W)$, the Coefficient of Network Complexity is defined as:*

$$CNC_{PN} = \frac{|W|}{|P| + |T|}$$

Another valuable property of an identified net is strong connexity. Suppose that the identified nets consists in multiple components; it is hard to picture all interleavings being possible due to the full parallelism. Besides, if multiple unconnected components are identified, they probably belong to different functional subsystems, which could be identified separately. If the net consists in one strongly connected component, the concurrency is lessened and the model is easier to understand. Strong connexity is a good property to aim for in the inference procedure.

4.3.2 Importance of understandability

The balance between precision and understandability is illustrated here by an example.

Example 4.5. *Consider the system of Figure 4.5.*

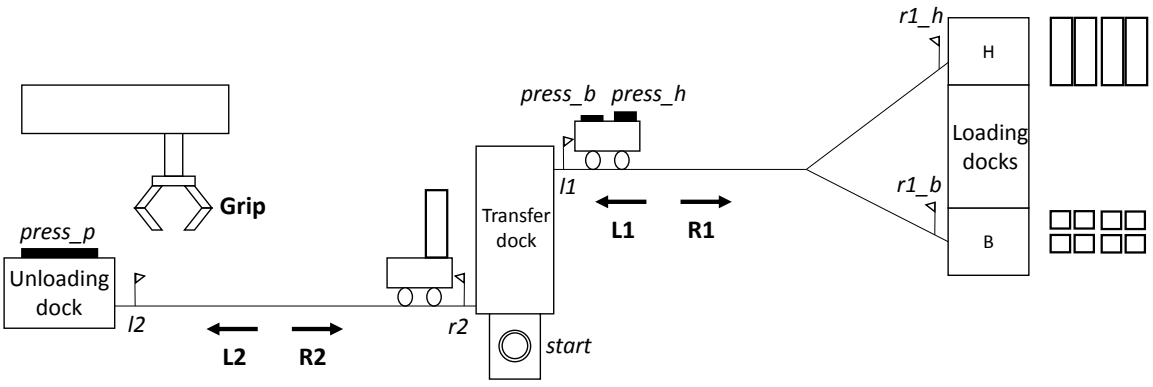


Figure 4.5: A system consisting of two chariots and a gripper

The middle is the transfer dock, with the *start* button and the initial position of two transport chariots (sensors *l1* and *l2*).

The upper chariot can roll right (actuator **R1**) to reach one of two loading docks (H and B), sensors *r1_h* and *r1_b* indicating its presence. Once charged with the corresponding load (the chariot is equipped with two pressure sensors *press_b* and *press_h* to assess the nature of the load), it goes left (**L1**) back to the transfer dock.

Meanwhile, the lower chariot rolls left (**L2**) with a load. When it reaches the unloading dock (*l2*), a gripper takes the load (**Gripper**) and deposits it on a pressure plate (*press_p*) while the chariot rolls right (**R2**) back to the transfer dock.

The observable behaviour is the one presented in Figure 4.6. It is associated to a sequence S of events on the alphabet $T_{Obs} = \{t_A, t_B, t_C, t_D, t_E, t_F, t_G, t_H, t_I\}$.

Only four types of subsequences have been observed: $\{\sigma_1 = t_A t_B t_D t_E t_H t_F t_I$, $\sigma_2 = t_A t_D t_B t_E t_H t_F t_I, \sigma_3 = t_A t_C t_D t_F t_G t_E t_I, \sigma_4 = t_A t_D t_C t_F t_G t_E t_I\}$ and $S = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_1$ $\sigma_2\sigma_3\sigma_4\sigma_2\sigma_2\sigma_4\sigma_4\sigma_1\sigma_3\sigma_3\sigma_2\sigma_2\sigma_4\sigma_3\sigma_2\sigma_1$ ($|S| = 147$)

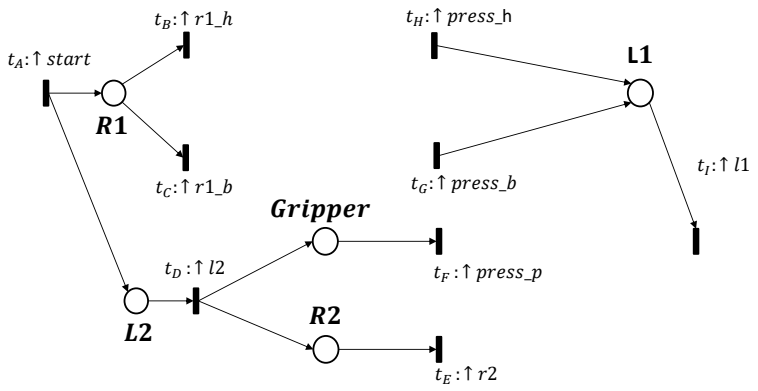


Figure 4.6: Observable part of the system. 5 outputs and 9 observable transitions

Two unobservable behaviour solutions are proposed in Figure 4.7, with the evolution of their exceeding language.

In solution (a), only 4 unobservable places have been added. Parallelism between the chariot movements is explicit, the succession and causality of actions is easy to read, namely because the number of additional places and edges remains low. This solution contains a lot of additional behaviour that was not observed in the sequence S (for instance, t_E could fire before t_B or t_C), characterized by the exceeding behaviour : $|L^2_{Exc}| = 38$.

Solution (b) is much closer to the observation, since $|L^n_{Exc}| = 0$ for $n \leq 6$. However, 9 unobservable places with 30 edges are added, the resulting net is very complex and almost unreadable. Furthermore, whereas each place in model (a) can be easily given a physical interpretation (for instance P5 = upper chariot waiting for a load at dock H), most of the places of solution (b) are very complex (for instance P13 = lower chariot returned to the transfer dock OR gripper unloaded of the load, enabling upper chariot to leave the loading dock OR to reach the transfer dock).

One of the objective of behavioural identification is to aim for an understandable model, therefore solution (a) is preferable to solution (b); notably, $CNC_{(a)} = 1.16$,

while $CNC_{(b)} = 1.82$.

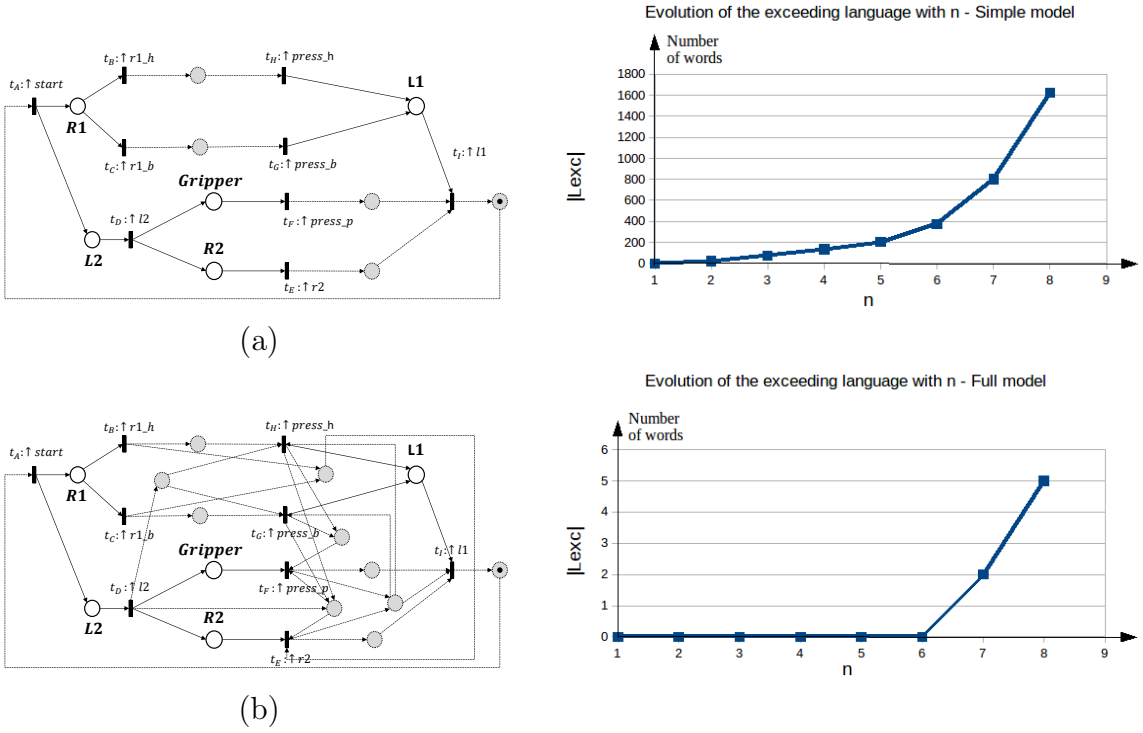


Figure 4.7: (a) A simple solution, with a lot of exceeding language ; (b) A complex solution, with no exceeding language up to $n = 6$

To conclude this section, we consider that a 'good' net for reverse engineering has three quality characteristics, who are ordered by importance as following:

1. It is strongly connected
2. It is structurally simple (minimimal CNC_{PN})
3. It is precise (minimal $|L^n_{Exc}|$)

Such a net should be computed at low calculus cost. The heuristic proposed in the next section therefore is designed to find such nets.

Notice that in the example, the observable behaviour of the system was included as well. The inference problem was stated without using the observable places, and solutions are found without them as well. However, they are required to understand the reactive behaviour, and the model of interest for reverse engineering is the full model, not just the underlying unobservable part. Therefore, all quality criteria must be evaluated on the full models.

4.4 Discovery in practice

The problem of building the net is reformulated into a partial exploration issue: testing a limited number of pairs in $(2^T - \{\emptyset\})^2_{\not\emptyset}$, to build the subset of mutual dependencies $\{\leftrightarrows\}_{n'}$, such that the resulting net is of high quality.

Firstly, the search space $(2^T - \{\emptyset\})^2_{\not\emptyset}$ is naturally partitioned into subspaces, using the sizes of the subalphabets as parameter δ. It is shown how δ is correlated to the structural complexity of the net. Then, the proposed strategy is to explore the subspaces by rising values of δ, to limit the complexification of the net. The exploration is stopped when the net becomes strongly connected.

4.4.1 Partitioning of the search space

An element of the search space $(2^T - \{\emptyset\})^2_{\not\emptyset}$ is an unordered pair $\{\Sigma_i, \Sigma_j\}$. The respective sizes of the alphabets are denoted $\delta_i = |\Sigma_i|$ and $\delta_j = |\Sigma_j|$. Recalling Figure 4.2, if $\Sigma_i \leftrightarrows \Sigma_j$, two places p_{ij} and p_{ji} are added to the net, verifying:

$$^\bullet p_{ij} = \Sigma_i; p_{ij}^\bullet = \Sigma_j$$
$$^\bullet p_{ji} = \Sigma_j; p_{ji}^\bullet = \Sigma_i$$

Therefore the in-degree of p_{ij} is exactly δ_i, while its out-degree is δ_j, and reciprocally for p_{ji}. The degree of both places is $\delta = \delta_i + \delta_j$. The number of edges associated to a place, which is a parameter of the structural complexity of the net, is directly linked to the sizes of the alphabets. A partition of the search space by the sum of the sizes of the alphabets (or equivalently, the degree of the places) comes naturally, and equivalence classes are defined accordingly.

Definition 4.7. *Let $S_1 = \{\Sigma_i^1, \Sigma_j^1\}$ and $S_2 = \{\Sigma_i^2, \Sigma_j^2\}$ be two unordered pairs of subalphabets of T. The degree-equivalence relation $\Leftrightarrow_\delta$ is defined on $(2^T - \{\emptyset\})^2_{\not\emptyset}$ as following:*

$$(S_1 \Leftrightarrow_\delta S_2) \Leftrightarrow \begin{cases} min(|\Sigma_i^1|, |\Sigma_j^1|) = min(|\Sigma_i^2|, |\Sigma_j^2|) \\ max(|\Sigma_i^1|, |\Sigma_j^1|) = max(|\Sigma_i^2|, |\Sigma_j^2|) \end{cases}$$

Proposition 4.8. *$\Leftrightarrow_\delta$ is reflexive, symetric and transitive, hence is an equivalence relation.*

The proof is straightforward.

To facilitate the understanding, from now on, the subscript i will always label the smallest subalphabet (*i.e.* $min(|\Sigma_i^1|, |\Sigma_j^1|) = |\Sigma_i^1|$), whereas j labels the biggest subalphabet. This labelling is not an issue since the pair of alphabets considered is unordered. Since $\Leftrightarrow_\delta$ is an equivalence relation, equivalence classes can be defined, and form a partition of $(2^T - \{\emptyset\})^2_{\not\emptyset}$.

Definition 4.8. *Let $(\delta_i, \delta_j) \in (\mathbb{N}^*)^2$ be two integers, such that $1 \leq \delta_i \leq \delta_j \leq |T|$. The subset of $(2^T - \{\emptyset\})^2_{\not\emptyset}$ of all unordered pairs of alphabets $S = \{\Sigma_i, \Sigma_j\}$ verifying $|\Sigma_i| = \delta_i$*

and $|\Sigma_j| = \delta_j$ *is denoted* $\mathbb{D}_{\delta_i,\delta_j}$.

$$\mathbb{D}_{\delta_i,\delta_j} = \{S = \{\Sigma_i, \Sigma_j\} | |\Sigma_i| = \delta_i \wedge |\Sigma_j| = \delta_j\}$$

Proposition 4.9. *For* $1 \leq \delta_i \leq \delta_j \leq |T|$, $\delta_i + \delta_j \leq |T|$, $\mathbb{D}_{\delta_i,\delta_j}$ *defines an equivalence class on* $(2^T - \{\emptyset\})^2_{\not\emptyset}$ *for* $\Leftrightarrow_\delta$. *The set* $\{\mathbb{D}_{\delta_i,\delta_j} | 1 \leq \delta_i \leq \delta_j \leq |T|, \delta_i + \delta_j \leq |T|\}$ *is the quotient set of* $(2^T - \{\emptyset\})^2_{\not\emptyset}$ *by* $\Leftrightarrow_\delta$ *and realizes a partition, i.e.*

$$(2^T - \{\emptyset\})^2_{\not\emptyset} = \bigcup_{\substack{1 \leq \delta_i \leq \delta_j \leq |T| \\ \delta_i + \delta_j \leq |T|}} \mathbb{D}_{\delta_i,\delta_j}$$

The proof is straightforward from the properties of equivalence classes.

Example 4.6. *Consider* $|T| = \{t_1, t_2, t_3, t_4\}$. *The different cells are:*

$$\begin{aligned}
\mathbb{D}_{1,1} = \; & [\{\{t_1\},\{t_2\}\};\{\{t_1\},\{t_3\}\};\{\{t_1\},\{t_4\}\};\{\{t_2\},\{t_3\}\};\{\{t_2\},\{t_4\}\};\{\{t_3\},\{t_4\}\}] \\
\mathbb{D}_{1,2} = \; & [\{\{t_1\},\{t_2,t_3\}\};\{\{t_1\},\{t_2,t_4\}\};\{\{t_1\},\{t_3,t_4\}\};\{\{t_2\},\{t_1,t_3\}\}; \\
& \{\{t_2\},\{t_1,t_4\}\};\{\{t_2\},\{t_3,t_4\}\};\{\{t_3\},\{t_1,t_2\}\};\{\{t_3\},\{t_1,t_4\}\}; \\
& \{\{t_3\},\{t_2,t_4\}\};\{\{t_4\},\{t_1,t_2\}\};\{\{t_4\},\{t_1,t_3\}\};\{\{t_4\},\{t_2,t_3\}\}] \\
\mathbb{D}_{1,3} = \; & [\{\{t_1\},\{t_2,t_3,t_4\}\};\{\{t_2\},\{t_1,t_3,t_4\}\};\{\{t_3\},\{t_1,t_2,t_4\}\};\{\{t_4\},\{t_1,t_2,t_3\}\}] \\
\mathbb{D}_{2,2} = \; & [\{\{t_1,t_2\},\{t_3,t_4\}\};\{\{t_1,t_3\},\{t_2,t_4\}\};\{\{t_1,t_4\},\{t_2,t_3\}\}]
\end{aligned}$$

The search space is now partitionned into cells $\mathbb{D}_{\delta_i,\delta_j}$, whose sizes are easy to estimate:

Proposition 4.10. *Let* $\mathbb{D}_{\delta_i,\delta_j}$ *be a cell of* $(2^T - \{\emptyset\})^2_{\not\emptyset}$. *Then*

$$\begin{aligned}
|\delta_i| < |\delta_j| \Rightarrow \quad & |\mathbb{D}_{\delta_i,\delta_j}| = C^{\delta_i}_{|T|} C^{\delta_j}_{|T|-\delta_i} \\
|\delta_i| = |\delta_j| \Rightarrow \quad & |\mathbb{D}_{\delta_i,\delta_i}| = \frac{1}{2} C^{\delta_i}_{|T|} C^{\delta_i}_{|T|-\delta_i}
\end{aligned}$$

where $C^k_n = \frac{n!}{k!(n-k)!}$ *expresses the number of combinations of* k *elements of a size* n *set.*

Proof. An element of $\mathbb{D}_{\delta_i,\delta_j}$ consists in two disjoints subalphabets of T. If $|\delta_i| < |\delta_j|$, there are $C^{\delta_i}_{|T|}$ combinations to build the first alphabet of size δ_i, and $C^{\delta_j}_{|T|-\delta_i}$ combinations of the remaining transitions to build the second alphabet. If $|\delta_i| = |\delta_j|$, each couple of alphabet will be counted twice with the first formula, hence the factor $\frac{1}{2}$. □

Suppose that $\delta_i, \delta_j \ll |T|$, an approximation of the size is $|\mathbb{D}_{\delta_i,\delta_j}| = \mathcal{O}(|T|^{\delta_i+\delta_j})$. The exploration of a cell for low values of δ_i, δ_j runs now in polynomial time regarding the number of transitions.

The value $\delta = \delta_i + \delta_j$ is important, as it is also the degree of places to be added, *i.e.* the number of edges that are added to the net with the place. For instance, mutual dependencies from $\mathbb{D}_{1,3}$ and $\mathbb{D}_{2,2}$ lead to the addition of places with 4 edges in both cases. Regarding the structural metric CNC_{PN}, this has the same effect on the net complexity. The study of the two cells has the same outcome on the net complexity,

hence they should not be separated. The search space can be coarsely re-partitionned by using the degree δ.

Definition 4.9. *Let $\delta \in [\![2, |T|]\!]$ be a fixed degree. Then the search space $\mathbb{D}_\delta$ is defined by:*

$$\mathbb{D}_\delta = \bigcup_{\substack{1 \leq \delta_i \leq \delta_j \leq |T| \\ \delta_i + \delta_j = \delta}} \mathbb{D}_{\delta_i, \delta_j}$$

Equivalently,

$$\mathbb{D}_\delta = \{(\Sigma_i, \Sigma_j) \in (2^T - \{\emptyset\})^2_{\not\emptyset} | |\Sigma_i| + |\Sigma_j| = \delta\}$$

Proposition 4.11. *The set $\{\mathbb{D}_\delta, 2 \leq \delta \leq |T|\}$ is the quotient set of $(2^T - \{\emptyset\})^2_{\not\emptyset}$ by $\Leftrightarrow_\delta$ and realizes a partition, i.e.*

$$(2^T - \{\emptyset\})^2_{\not\emptyset} = \bigcup_{\delta \in [\![2, |T|]\!]} \mathbb{D}_\delta$$

The proof is straightforward.

For instance $\mathbb{D}_5 = \mathbb{D}_{1,4} \cup \mathbb{D}_{2,3}$, or $\mathbb{D}_8 = \mathbb{D}_{1,7} \cup \mathbb{D}_{2,6} \cup \mathbb{D}_{3,5} \cup \mathbb{D}_{4,4}$. A visual representation is given in Figure 4.8; for instance $\mathbb{D}_4$ contains all alphabet pairs enabling the discovery of both 1vs3 ($\mathbb{D}_{1,3}$) and 2vs2 ($\mathbb{D}_{2,2}$) places, whereas $\mathbb{D}_2 = \mathbb{D}_{1,1}$ contains only 1vs1 places.

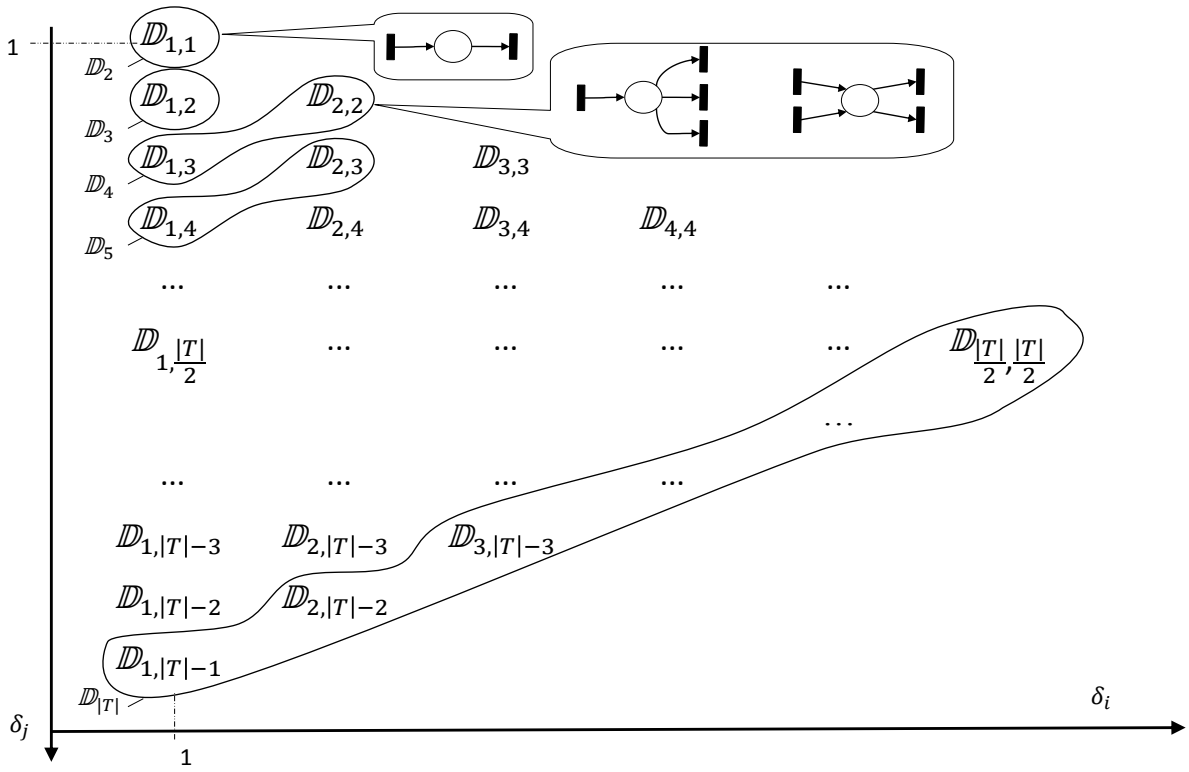


Figure 4.8: Visual representation of the partition of $(2^T - \{\emptyset\})^2_{\not\emptyset}$.

δ, sum of the sizes of the alphabets or equivalently degree of the places to be discovered, can be used as a parameter to partition the search space into equivalence classes.

The searching strategy has to decide in which order the cells will be studied to discover places.

4.4.2 Exploration strategy

The very nature of the studied system helps to choose an adequate strategy. A manufacturing system consists mostly of concurrent sequential processes: the operations are conducted one after another on a component, and the actuators are sollicited one after the other. A sequential process consists mostly of single-input single-output places, and is highly deterministic. Even if multiple subprocesses evolve simultaneously, exhibiting concurrency, their individual behaviour remains sequential; they are modelled by parallel streams of sequential single-input single-output places.

However, some choices can occur within the system. For instance, in Example 4.5, the first chariot can choose to fetch either a small or a big box. Choices are sparser than sequentialities in an actual system, and most often involve few inputs/outputs, hence places related to choices should also involve few transitions in our case. A reasonable assumption is therefore that the unobservable places verify $|{}^\bullet p_{Unobs}| + |p^\bullet_{Unobs}| = \delta \ll |T|$. Consequently, exploring $\mathbb{D}_\delta$ cells for low values of δ should be sufficient to obtain a good model of the real system.

Furthermore, recall that under this hypothesis, the size of the cells is estimated at $|\mathbb{D}_\delta| \sim |T|^\delta$ for $\delta \ll |T|$. Hence, studying only the cells for low values of *delta* is computationally efficient, compared to the full exploration.

Therefore, the proposed strategy is to study $\mathbb{D}_\delta$ cells by rising values of δ, starting at $\delta = 2$. $\mathbb{D}_2$ provides all purely sequential places; the sequential subprocesses are completely identified with just the exploration of $\mathbb{D}_2$, and concurrency is already explicit. Then, $\mathbb{D}_\delta$ provide choice places for $\delta \geq 3$. If choices are present in the system (due to production recipes or undeterminism), they are expected to occur for low values of δ.

To corroborate this strategy, have a look at the sizes of the different cells $\mathbb{D}_\delta$, plotted in Figure 4.9 for different values of $|T|$. The minimal value is always reached for $\delta = 2$. Given that it represents subalphabets consisting of only one transition, $|\mathbb{D}_2| = \frac{1}{2}|T|(|T|-1)$. Another simple cell to evaluate is $\mathbb{D}_{|T|}$: there are $2^{|T|} - 1$ non-empty subsets of T, therefore $2^{|T|} - 2$ pairs of non-empty subsets of T whose union is T, and finally $|\mathbb{D}_{|T|}| = 2^{|T|-1} - 1$ ordered pairs.

The size of $\mathbb{D}_\delta$ always rises until $\delta \approx 0.7|T|$, and starts to shrink for higher values of δ. However, the relevance of such high degree places is questionable: they are structurally hard to understand. Limiting the structural complexity should be more important than limiting the size of the search space. Therefore only the left side of the curves is of importance.

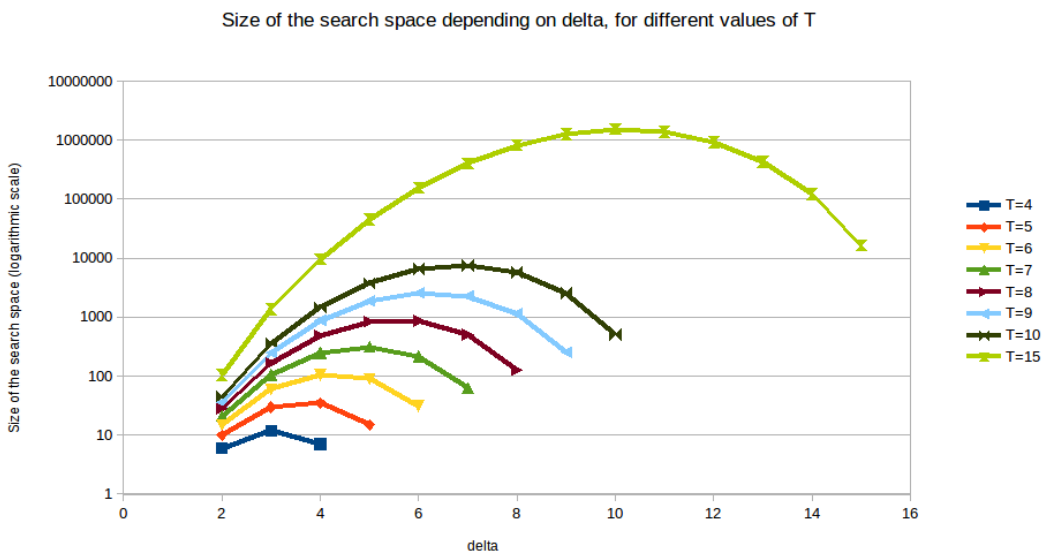


Figure 4.9: Size of the cell $\mathbb{D}_\delta$ of the search space depending on δ, plotted for different values of $|T|$. Y scale is logarithmic.

4.4.3 Stopping criterion

Suppose that the full exploration can be achieved, and that the result of the identification is two or more nets unconnected to each other. They can be considered as distinct subsystems sharing no relationship towards one another, having fully independent behaviours. If the isolation of the system to be identified is done correctly, it can be assumed that all inputs and outputs are sollicited during an operation, and are therefore connected in one way or another.

Therefore, a satisfying stopping criterion is to reach a strongly connected model, which is also a good quality for understandability. If choices are present within the system, it is likely that the model will not be strongly connected after the exploration of $\mathbb{D}_2$. Some conflicts could be discovered for the low values of δ greater than two, resulting in strong connexity. Checking strong connexity is effected in polynomial time using Tarjan's algorithm [Tarjan, 1972]. As a reminder, the definition of strongly connected components is recalled here:

Definition 4.10. *Let $G = (V, E)$ be a directed graph. Suppose that for each pair of vertices v, w in G, there exist paths $p1 : v \Rightarrow w$ and $p2 : w \Rightarrow v$ (a path $p : v \Rightarrow w$ in G is a sequence of vertices and edges leading from v to w). Then G is said to be strongly connected.*

Definition 4.11. *Let $G = (V, E)$ be a directed graph. Two vertices v and w are equivalent if there is a closed path $p : v \Rightarrow v$ which contains w. Let the distinct equivalence classes under this relation be V_i, let $G_i = (V_i, E_i)$, where $E_i = \{(v, w) \in E | (v, w) \in V_i^2\}$. Then :*
(i) Each G_i is strongly connected.
(ii) No G_i is a proper subgraph of a strongly connected subgraph of G.

Naturally, this criterion might never be met, either because the real system is truly composed of independent processes, or because the computation becomes untractable due to the sheer size of $\mathbb{D}_\delta$. A backup threshold is set to ensure the halting.

For instance, if a cell whose size is greater than 1Mio candidates is reached, the computation is stopped. This can happen for low values of δ when T increases, for instance, if $|T| = 30$, $|\mathbb{D}_{2,3}| \sim 1Mio$. Another possibility is to count time; if no strongly connected solution is found within the hour, the computation stops.

4.4.4 Algorithmic application

Example 4.5 is used to illustrate the procedure; the observable part N^{Obs} is recalled in Figure 4.10(a). However, recall that Theorem 1 is only applicable to nets who are already solution of the inference problem; the observable fragments issued of the first phase are most often not a solution. Namely, tA, tG or tH are source transitions, breaking the 1-boundedness property.

The first step before starting the exploration consists in completing the observable part with unobservable places, such that the resulting net N is a solution of the inference problem. The procedure is the following:

- An observable place is the image of a binary output, that is alternatively set to zero or one. The firing of pre-transitions (sets) are therefore constrained to alternate with the firing of post-transitions (resets) in the firing sequence. This means that any observable place $p_{i,Obs}$ is necessarily an admissible place, and that $\{{}^\bullet p_{i_Obs}\} \leftrightarrows \{p^\bullet_{i,Obs}\}$. $p_{i,Obs}$ is already one of the places associated to this dependency; the reverse place can be added as well. Given $p_{i,Obs}$, add $p_{i,Unobs}$ such that ${}^\bullet p_{i,Unobs} = {}^\bullet p_{i,Obs}$, $p^\bullet_{i,Unobs} = p^\bullet_{i,Unobs}$, and $M_0(p_{i,Obs}) + M_0(p_{i,Unobs}) = 1$

- A single transition has been observed at least once in the firing sequence, and represents an input that could not be linked to an output with a direct causality. At this point, the best to be said about this transition is that it can always fire; because no mutual dependency with any other transitions has been discovered yet. Therefore, it must always be enabled and never disabled; this is translated by a single selfloop place. As soon as the transition will be involved in a mutual dependency, the selfloop becomes implicit and is removed. Given t_j, add p_{jUnobs} such that ${}^\bullet p_{j,Unobs} = \{t_j\}$, $p^\bullet_{j,Unobs} = \{t_j\}$ and $M_0(p_{j,Unobs}) = 1$

The initial net built for the chariots is in Figure 4.10(b). There are no isolated transitions, and 5 unobservable places are added.

The proposed heuristic is resumed by Algorithm 4.3, where a cell-size threshold Thr has been chosen:

After the deletion of implicit places for $\mathbb{D}_2$, the net of Figure 4.10(c) is obtained. It is strongly connected, the discovery is stopped and the procedure achieved. Only $|\mathbb{D}_2| = 36$ pairs were tested, when the full problem had $|(2^T - \{\emptyset\})^2_{\not\emptyset}| = 9330$ pairs, hence

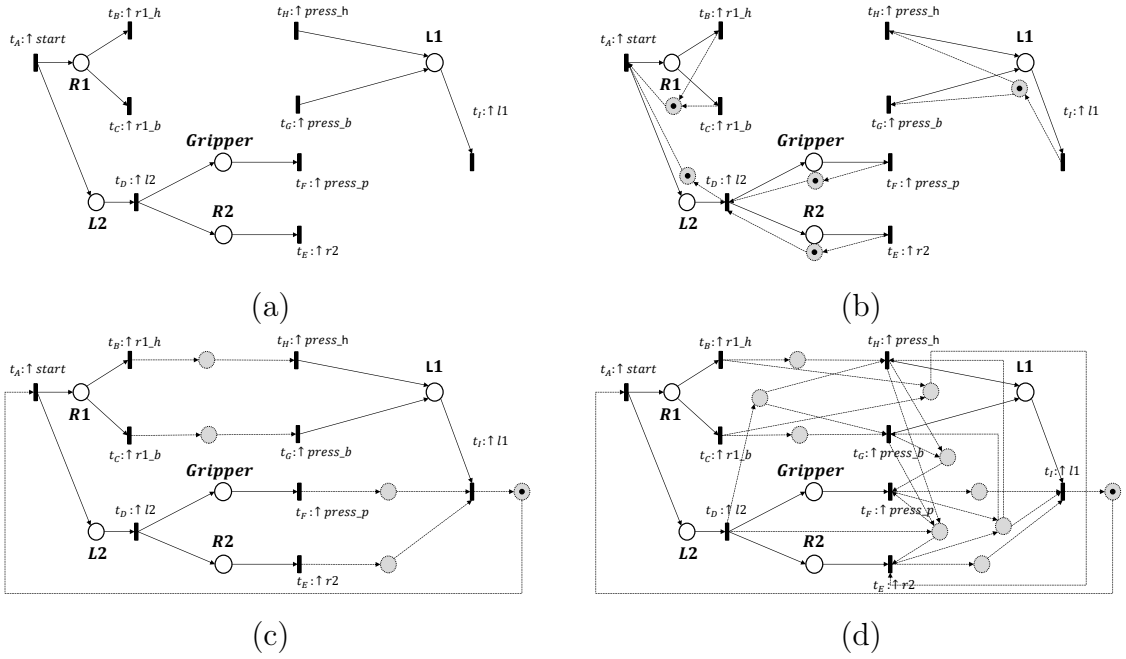


Figure 4.10: The chariots example: (a)Observable part; (b)Initial net for the exploration; (c)Net identified at the end of the limited exploration; (d)Net identified by a full exploration

Algorithm 4.3 Unobservable Behaviour Inference

Require: $S = s_1 \dots s_{|S|}$ a sequence, N the first net solution.
Ensure: Final net

```
δ = 2
while N is not strongly connected OR |D_δ| ≤ Thr do
    for (Σ_i, Σ_j) ∈ D_δ do
        if Σ_i ⇆ Σ_j then
            Add p_ij and p_ji to N
        end if
    end for
    δ ← δ + 1
end while
```

only 0.4% of the search space was explored. The net of Figure 4.10(d) is obtained if the full exploration is conducted; it is notably harder to understand, even though more precise.

The proposed heuristic has been designed adequately to find, within reasonable calculus cost, connected simple nets fitting for reverse engineering. Some additional results are proposed in the upcoming Section 4.5, and additional examples are given in Section 4.6 to illustrate the strengths of the method.

4.5 Extensions

This section proposes two reflexions to complement the main contribution: around implicit places and the further reduction of the search space.

4.5.1 Implicit places and consequences

Firstly, theoretical background on implicit places is recalled. They have been extensively studied in [Garcia-Valles and Colom, 1999], from which the following definition is extracted:

Definition 4.12 ([Garcia-Valles and Colom, 1999]). *Let $N = (G, M_0)$ be a net system and $N' = (G', M'_0)$ the net system N without a place p. p is an implicit place if and only if $L(N) = L(N')$, i.e. the removal of p preserves all firing sequences of N.*

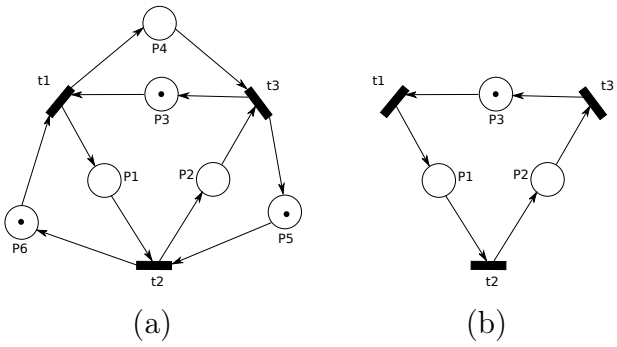


Figure 4.11: An example of a net (a), after deletion of implicit places (b)

An implicit place can be interpreted as a false synchronisation in the net. Look at place p_4 in Figure 4.11. It is filled after the firing of t_1, but its post-transition t_3 will not be fired until p_2 is filled as well. p_2 is the actual constraint of synchronisation for firing t_3, unlike p_4, which makes the latter implicit from the viewpoint of t_3. It is actually a shortcut for the path p_1p_2. [Garcia-Valles and Colom, 1999] proposed a linear programming problem to detect implicit places.

Now, suppose that a sequence S is given, that a net solution $N = (G, M_0)$ has been identified after the exploration of $\mathbb{D}_\delta$, and that the study is pursued with $\mathbb{D}_{\delta+1}$. Suppose that a new couple of places (p_{ij},p_{ji}) is discovered in $\mathbb{D}_{\delta+1}$, and added to N, building N'. The following can happen:

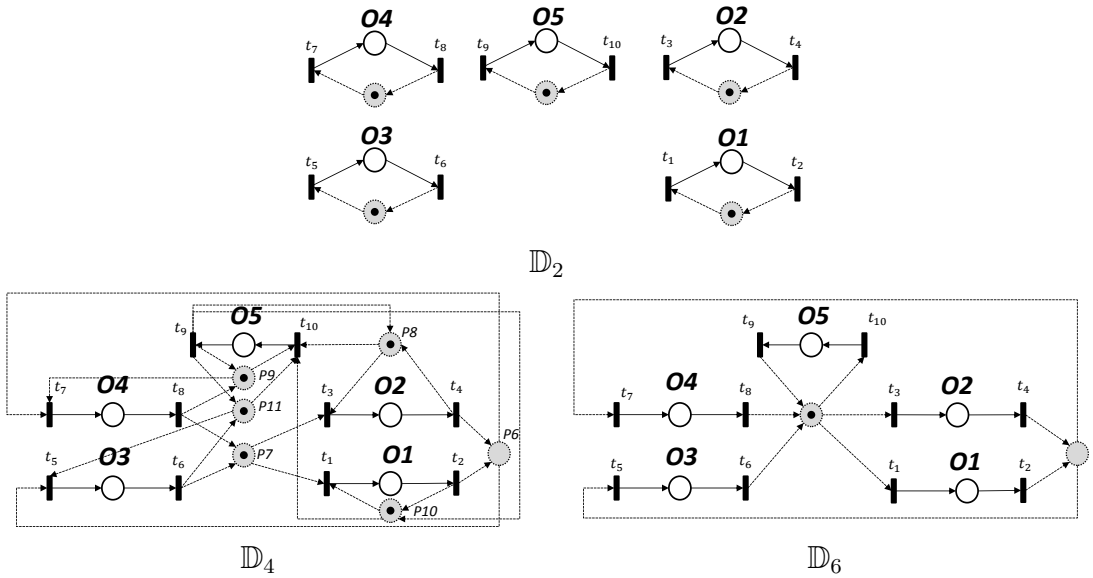
- $L(N') = L(N)$. The two places are implicit, and are the most complex places of N' (degree $\delta + 1$). They will be deleted.
- $L(N') \subset L(N)$. In this case, the places improve the precision to the net, and will be kept.

However, in the latter case, some places previously belonging to N might have become implicit. If the number of such places is low enough, the result of both their deletion and the addition of (p_{ij},p_{ji}) should result in N' being still more complex than N. But if enough low-degree places are removed, the complexity might drop, as presented in the following example.

Example 4.7. *Consider a system composed of five actuators labelled O1 to O5, moving in the same workspace. The rules implemented in the controller are the following:*

- *Only one actuator can move at the same time.*
- *A movement of O1 or O2 is always followed by a movement of O3 or O4.*
- *O3 and O4 can not move unless O1 or O2 moved first.*

The observable part leads to 5 observable fragments, each consisting in one observable place with one input and one output transition, hence 10 transitions t_1 *to* t_{10}*. The firing sequence is* $S = t_1t_2t_7t_8t_1t_2t_5t_6t_9t_{10}t_3t_4t_7t_8t_3t_4t_7t_8t_1t_2t_5t_6t_9t_{10}t_9t_{10}t_9t_{10}t_3t_4t_7t_8t_3t_4t_5t_6t_3t_4$ $t_7t_8t_1t_2t_5t_6t_1 \ldots$, $|S| = 200$*. The result of the exploration is presented as three nets in Figure 4.12, along with a table resuming their properties*



Net	$\|P\|$	$\|W\|$	CNC_{PN}	$\|L^2_{Exc}\|$
$\mathbb{D}_2$	10	20	1	72
$\mathbb{D}_4$	11	34	1.62	4
$\mathbb{D}_6$	7	20	1.18	0

Figure 4.12: Results of the identification of Example 4.7

The first net is obtained after the exploration of $\mathbb{D}_2$. There are five strongly connected components, hence the exploration is pushed further. The exploration of $\mathbb{D}_3$ did not reveal any place, but the exploration of $\mathbb{D}_4$ revealed new places, namely 6 remaining non observable places after the deletion of implicit places. Since the net is strongly connected, the inference should stop.

However, pushing the exploration up to $\mathbb{D}_6$ leads to the discovery of one place in $\mathbb{D}_{3,3}$, which makes implicit five places previously discovered in $\mathbb{D}_{2,2}$. This last net is both the simplest strongly connected assembled net, and the most precise of all. It is obviously simpler than the second net, although it required more computations to reach.

This example illustrates that the simplicity metrics are not monotonous with the chosen strategy, unlike the precision metric. When the identication is stopped due to reaching a strongly connected net, it is not guaranteed to be the simplest strongly connected one.

4.5.2 Reduction of the algorithmic cost

During the exploration of the search space, information can be extracted to avoid studying irrelevant couples of alphabets. To demonstrate how, a new relationship between alphabets, the *domination*, is introduced. This relationship is a weaker form of the mutual dependency:

Definition 4.13. *Let* $\Pi(S) = \sigma_i^1\sigma_j^1\sigma_i^2\sigma_j^2\ldots$ *be the result of the projection of* S *on* $\Sigma_i \cup \Sigma_j$*. If,* $\forall l > 0, |\sigma_j^l| = 1$ *and* $\exists k > 0, |\sigma_i^k| > 1$*, the alphabet* Σ_i *is said to dominate the alphabet* Σ_j*, written* $\Sigma_i \rightarrow \Sigma_j$*.*

Σ_i dominating Σ_j means that at least once in the projected sequence, two transitions of Σ_i have been observed consecutively. One or more transitions might be "missing" in Σ_j for the two alphabets to be mutually dependant. See for instance the following sequence and projections on $T = \{t_1, t_2, t_3\}$:

$$\begin{aligned} S &= t_1t_2t_1t_3t_1t_2t_1t_2t_1t_2t_1t_3t_1t_3t_1t_2t_1t_3t_1t_3 \\ \Pi_{t_1,t_2}(S) &= t_1t_2t_1 \quad t_1t_2t_1t_2t_1t_2t_1 \quad t_1 \quad t_1t_2t_1 \quad t_1 \\ \Pi_{t_1,t_3}(S) &= t_1 \quad t_1t_3t_1 \quad t_1 \quad t_1 \quad t_1t_3t_1t_3t_1 \quad t_1t_3t_1t_3 \end{aligned}$$

The projection on $\{t_1\} \cup \{t_2\}$ reveals that $\{t_1\}$ dominates $\{t_2\}$. There might be missing a transition that can fill the holes between two successives occurrences of t_1; if such a transition is added to the set $\{t_2\}$ the resulting set might be mutually dependant of $\{t_1\}$. A domination is also found of $\{t_1\}$ on $\{t_3\}$. Since $\{t_1\}$ dominates both $\{t_2\}$ and $\{t_3\}$, it is interesting to test $\{t_1\}$ with $\{t_2, t_3\}$; in this case, a mutual dependency is discovered.

If dominations are discovered, a dominated set can be constructed for the dominant alphabet:

Definition 4.14. *Let* Σ *be a non-empty alphabet. The set dominated by* Σ*, written* $Dom(\Sigma)$ *is the set of all alphabets dominated by* Σ*:*

$$Dom(\Sigma) = \{\Sigma_i \in (2^T - \emptyset) |\ \Sigma \rightarrow \Sigma_i\}$$

This set can be parametrized by the value of the degree δ *:*

$$Dom_\delta(\Sigma) = \{\Sigma_i \in (2^T - \emptyset) |\ \Sigma \rightarrow \Sigma_i \wedge |\Sigma_i| + |\Sigma| = \delta\}$$

Notice that defining $Dom_\delta(\Sigma)$ implies that $|\Sigma| \leq (\delta - 1)$. In the previous example, $Dom_2(\{t_1\}) = \{\{t_2\}, \{t_3\}\}$. A $Dom_\delta(\Sigma)$ is constructed during the exploration of $\mathbb{D}_\delta$, or

more precisely, during the exploration of $\mathbb{D}_{|\Sigma|,\delta-|\Sigma|}$, assuming that $|\Sigma| < \delta - |\Sigma|$.

The (non-)emptiness of $Dom_\delta(\Sigma)$ is interesting information, due to the following proposition:

Proposition 4.12. *Let Σ_i and Σ_j be two alphabets such that $\Sigma_i \leftrightarrows \Sigma_j$, and Σ' a non-empty strict subalphabet of Σ_j, i.e. $\Sigma' \subset \Sigma_j$. Then, Σ_i dominates Σ', i.e. $\Sigma_i \rightarrow \Sigma'$.*

Proof. When projecting the sequence on $\Sigma_i \cup \Sigma'$, all t_i remain in the projection, but at least one $t_j \in \Sigma_j - \Sigma'$ will be missing. Therefore, at least two consecutives occurences of t_i will be observed, and $\Sigma_i \rightarrow \Sigma'$.

□

Setting $|\Sigma_i| = \delta_i$, $|\Sigma_i| = \delta_j$ and $\delta = \delta_i + \delta_j$, Proposition 4.12 states the following:

$$\Sigma_i \leftrightarrows \Sigma_j \Rightarrow \begin{cases} \forall \delta_k \in [\![\delta_i + 1, \delta - 1]\!], Dom_{\delta_k}(\Sigma_i) \neq \emptyset \\ \forall \delta_k \in [\![\delta_j + 1, \delta - 1]\!], Dom_{\delta_k}(\Sigma_j) \neq \emptyset \end{cases}$$

Proposition 4.12 has two consequences:

- If a mutual dependency $\Sigma_i \leftrightarrows \Sigma_j$ is to be discovered for a given degree δ, then dominations must necessarily be discovered for lower degrees ($\delta_k < \delta$). Candidates for the mutual dependency with Σ_i are necessarily unions of elements of $Dom_{\delta_k}(\Sigma_i)$ (same for Σ_j).
- Given an alphabet Σ_i, if there exists a value $\delta \geq \delta_i + 1$ for which $Dom_\delta(\Sigma_i)$ is empty, then there is no need to look for mutual dependencies of degree higher than δ implying Σ_i.

Notice however that the reverse of Proposition 4.12 is not true. If Σ_i dominates all subsets of Σ_j, the mutual dependency is not guaranteed.

Consider namely $T = \{t_1, t_2, t_3, t_4\}$ and $S = t_1\ t_4\ t_1\ t_3\ t_4\ t_1\ t_4\ t_2\ t_1\ t_2\ t_1\ t_4\ t_3\ t_4\ t_1\ t_3\ t_1$. For $\delta = 2$, the only relations discovered are $\{t_1\} \rightarrow \{t_2\}$ and $\{t_1\} \rightarrow \{t_3\}$, *i.e.* $Dom_2(\{t_1\}) = \{\{t_2\}, \{t_3\}\}$. However, $\{t_1\} \rightarrow \{t_2, t_3\}$ and no mutual dependency can be found. Hence, the test for mutual dependency must still be run for any candidate.

Dominations are discovered during the exploration of a cell $\mathbb{D}_\delta$, and only a subset of $\mathbb{D}_{\delta+1}$ is consistent with them. The idea is to build this subset of candidates. In the worst case, the subset of candidates is the whole cell, but most of the time, the $Dom_\delta(\Sigma)$ sets are empty or rather small.

Definition 4.15. *Let $\mathbb{D}_\delta$ be an explored cell of the search space, and Σ an alphabet such that $Dom_\delta(\Sigma) \neq \emptyset$. Then, the set of candidates implying Σ for a research in $\mathbb{D}_{\delta+1}$ is:*

$$Cand_{\delta+1}(\Sigma) = \{\Sigma_1 \cup \Sigma_2 | (\Sigma_1, \Sigma_2) \in (Dom_\delta(\Sigma))^2 \wedge |\Sigma_1 \cap \Sigma_2| = \delta - |\Sigma| - 1\}$$

Dominations are propaged vertically, as illustrated in Figure 4.13. A direct consequence is that only all $\mathbb{D}_{\delta_i,\delta_i}$ cells must be fully explored.

$$\begin{array}{l} \mathbb{D}_{\delta_i,\delta_i} \\ \quad\downarrow \quad \}Cand_{\delta_i+\delta_i+1}(\Sigma_i) \\ \mathbb{D}_{\delta_i,\delta_i+1} \\ \quad\downarrow \\ \quad\vdots \\ \quad\downarrow \\ \mathbb{D}_{\delta_i,\delta_j} \end{array}$$

Figure 4.13: Vertical propagation of the domination relationship

Example 4.8. *Look at the following sequence S:*

$$\begin{array}{rl} S = & t_1\ t_3\ t_2\ t_3\ t_1\ t_4\ t_2\ t_4\ t_1\ t_5\ t_2\ t_5\ t_1\ t_3\ t_1\ t_4\ t_1\ t_5\ t_2\ t_5\ t_2\ t_3\ t_2\ t_4 \\ \Pi_{t_1,t_2,t_3,t_4}(S) = & t_1\ t_3\ t_2\ t_3\ t_1\ t_4\ t_2\ t_4\ t_1\ \ \ t_2\ \ \ t_1\ t_3\ t_1\ t_4\ t_1\ \ \ t_2\ \ \ t_2\ t_3\ t_2\ t_4 \\ \Pi_{t_1,t_2,t_3,t_5}(S) = & t_1\ t_3\ t_2\ t_3\ t_1\ \ \ t_2\ \ \ t_1\ t_5\ t_2\ t_5\ t_1\ t_3\ t_1\ \ \ t_1\ t_5\ t_2\ t_5\ t_2\ t_3\ t_2 \\ \Pi_{t_1,t_2,t_4,t_5}(S) = & t_1\ \ \ t_2\ \ \ t_1\ t_4\ t_2\ t_4\ t_1\ t_5\ t_2\ t_5\ t_1\ \ \ t_1\ t_4\ t_1\ t_5\ t_2\ t_5\ t_2\ \ \ t_2\ t_4 \end{array}$$

For $\delta \in \{2,3\}$, no mutual dependencies are discovered. When studying $\delta = 4$, only the following domination relationships can be discovered: $\{t_1,t_2\} \to \{t_3,t_4\}, \{t_1,t_2\} \to \{t_3,t_5\}, \{t_1,t_2\} \to \{t_4,t_5\}$, i.e. $Dom_4(\{t_1,t_2\}) = \{\{t_3,t_4\},\{t_3,t_5\},\{t_4,t_5\}\}$. This hints that the only candidate for a degree 5 mutual dependency with $\{t_1,t_2\}$ is $\{t_3,t_4,t_5\}$. Indeed, the mutual dependency is tested and validated.

Discovering domination relationships is not hard: the projection is executed when running Algorithm 4.1 for an unordered pair $\{\Sigma_i,\Sigma_j\}$, and the variables max_i and max_j express the lengths of the longest subsequences belonging to Σ_i^+ and Σ_j^+. If exactly one of these two variables has 1 for value, then a domination is discovered according to Definition 4.13. The end of Algorithm 4.1 can be slightly altered to discover domination relationships, leading to Algorithm 4.4:

Algorithm 4.4 Modification of Algorithm 4.1 : Discover dominations and mutual dependancies

Require: $S = s_1 \dots s_{|S|}$ a sequence, (Σ_i,Σ_j) two disjoint non-empty subsets of T.
Ensure: Decision for $\Sigma_i \leftrightarrows \Sigma_j$, $\Sigma_i \to \Sigma_j$

```
1: {Lines 1 to 22 are identical to Algorithm 4.1}
23: if max_j == 1 then
24:    if max_i == 1 then
25:       return True, False {Σ_i ⇆ Σ_j}
26:    else
27:       return False, True {Σ_i → Σ_j}
28:    end if
29: else
30:    return False, False
31: end if
```

Operations in lines (22-27) in Algorithm 4.1 ran in $\mathcal{O}(1)$, and operations in lines (23-33) in Algorithm 4.1 run as well in $\mathcal{O}(1)$. Discovering dominations does therefore

not add complexity to the resolution of the problem.

Based on all previous results, Algorithms 4.5 and 4.6 are finally presented: they describe the exploration of a cell $\mathbb{D}_{\delta_i,\delta_j}$. In the second, most common case ($\delta_i < \delta_j$), the set $Cand_\delta(\Sigma_i)$ is generated from $Dom_{\delta-1}(\Sigma_i)$ according to Definition 4.15.

Algorithm 4.5 Exploring $\mathbb{D}_{\delta_i,\delta_j}$, $\delta = \delta_i + \delta_j$. Case $\delta_i = \delta_j$,

Require: $S = s_1 \dots s_{|S|}$ a sequence, T.
Ensure: Full exploration of $\mathbb{D}_{\delta_i,\delta_i}$

```
 1: {All possible pairs are tested}
 2: for Σ_i ∈ 2^T, |Σ_i| = δ_i do
 3:    for Σ_j ∈ 2^T, |Σ_j| = δ_j, Σ_j ∪ Σ_i = ∅ do
 4:       Mut, Dom ← Algorithm 4.4(S,{Σ_i, Σ_j})
 5:       if Mut==True then
 6:          Apply Algorithm 4.2(S,{Σ_i, Σ_j})
 7:       else if Dom==True then
 8:          Dom_δ(Σ_i) ← Dom_δ(Σ_i) ∪ Σ_j
 9:       end if
10:    end for
11: end for
```

Algorithm 4.6 Exploring $\mathbb{D}_{\delta_i,\delta_j}$, $\delta = \delta_i + \delta_j$. Case $\delta_i < \delta_j$,

Require: $S = s_1 \dots s_{|S|}$ a sequence, T, exploration of $\mathbb{D}_{\delta_i,\delta_j-1}$.
Ensure: Reduced exploration of $\mathbb{D}_{\delta_i,\delta_j}$

```
 1: {Only the candidates are studied}
 2: for Σ_i ∈ 2^T, |Σ_i| = δ_i do
 3:    for Σ_j ∈ Cand_δ(Σ_i) do
 4:       {Lines 4 to 9 identical to Algorithm 4.5}
10:    end for
11: end for
```

Algorithm 4.5 runs in $\mathcal{O}(|\mathbb{D}_{\delta_i,\delta_i}|)$, and Algorithm 4.6 runs at worst in $\mathcal{O}(|\mathbb{D}_{\delta_i,\delta_j}|)$, but the number of candidates can however be expected to be very low.

4.6 Practical examples

4.6.1 Unobservable behaviour only

These first examples are only dealing with unobservable behaviour, to show the kind of Petri net structures that can be discovered with the proposed approach. The structures presented here are classical difficulties met in the litterature.

Example 4.9 (Memory places)**.** *Consider the following sequence:* $S = t_1t_3t_4t_6\ t_2t_3t_5t_6\ t_2t_3t_5t_6\ t_1t_3t_4t_6\ t_1t_3t_4t_6\ t_2t_3t_5t_6$ *. After the exploration of* $\mathbb{D}_2$*,* $\{\leftrightarrows\} = \{\{t_1\},\{t_4\}\}$*;* $\{\{t_2\},\{t_5\}\}$*;* $\{\{t_3\},\{t_6\}\}$*, which leads to net (a) of Figure 4.14. It suggests that three concurrent processes interleave eachother. The exploration of* $\mathbb{D}_3$ *reveals choices that are*

added to the identified net (b). There is in fact only one process who can choose between two paths, and dependencies $\{\{t_1\},\{t_4\}\};\{\{t_2\},\{t_5\}\}$ *are kept as memory places. Memory places are discovered before choices.*

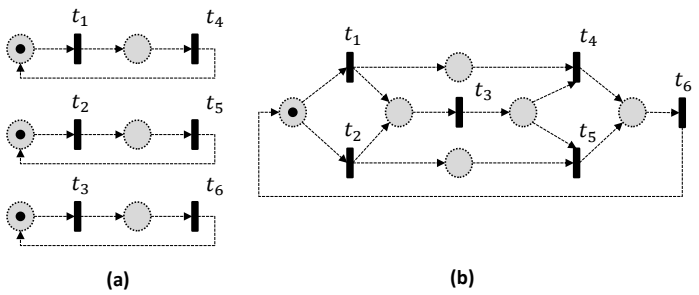


Figure 4.14: Example 4.9: (a) Net discovered for $\mathbb{D}_2$; (b) Net discovered for $\mathbb{D}_3$

Example 4.10 (Non resetable nets)**.** *There is no hypothesis on the reachability of the initial marking; in this example, the net of Figure 4.15 is identified for* $\delta = 3$ *from* $S = t_4t_5t_6\ \ t_1t_3t_4t_6\ \ t_0t_2t_5t_6\ \ t_1t_4t_3t_6\ \ t_0t_2t_5t_6\ \ t_1t_3t_4t_6\ \ t_0t_2t_5t_6\ \ t_0t_2t_5t_6\ \ t_0t_2t_5t_6\ \ t_0t_2t_5t_6\ \ t_0t_5t_2t_6$

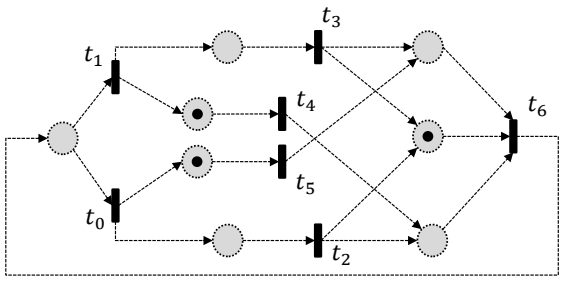


Figure 4.15: Example 4.10: the initial marking is unreachable after any firing.

Example 4.11 (Nested cycles)**.** *Nested cycles, i.e. backward loops in a process, are not an issue. They always imply choice places, hence require to explore at least up to* $\delta = 3$*. The net of Figure 4.16 is identified for* $\delta = 3$ *from* $S = t_0t_2t_3t_4t_5t_1t_0t_2t_3t_4t_5t_6t_{11}t_{12}t_8t_2t_3t_4t_5t_6t_7t_8t_2t_3t_4t_9t_{10}t_3t_4t_5t_6t_7t_8t_2t_3t_4t_5t_6t_7t_8t_2t_3t_4t_9t_{10}t_3t_4t_9t_{10}t_3t_4t_5t_1t_0t_2t_3t_4t_5t_6t_{11}t_8t_2t_3t_{12}t_4t_9t_{10}t_3t_4t_5t_6t_{11}t_8t_2t_{12}t_3t_4t_9t_{10}t_3t_4t_9t_{10}t_3t_4t_5t_1t_0t_2t_3t_4t_9t_{10}t_3t_4t_9t_{10}t_3t_4t_5$.

As pointed out in the litterature review, rule-based approaches often suffer to find concurrency in the sequence, whereas the approach proposed here discovers it easily:

Example 4.12 (High concurrency)**.** *5 chariots are considered in parallel. Each chariot goes right (di) then comes back left (gi). The chariots are synchronized by m. The observed sequence is*

$$\begin{aligned} S = {} & md_2d_4g_2d_1g_4d_3g_3g_1d_5g_5\ \ md_4d_3d_1g_1g_4d_5g_3d_2g_2g_5\ \ md_4g_4d_1d_2d_3g_3g_1g_2d_5g_5 \\ & md_1g_1d_2d_4d_5g_4g_2d_3g_3g_5\ \ md_1d_4d_5g_1g_4d_2d_3g_3g_2g_5\ \ md_2d_4d_3g_3d_5g_2g_4g_5d_1g_1 \\ & md_5d_2d_4d_1g_4g_1g_2d_3g_5g_3\ \ md_3d_2g_3d_5g_5g_2d_1g_1d_4g_4\ \ md_5g_5d_3d_1d_2d_4g_3g_4g_2g_1 \end{aligned}$$

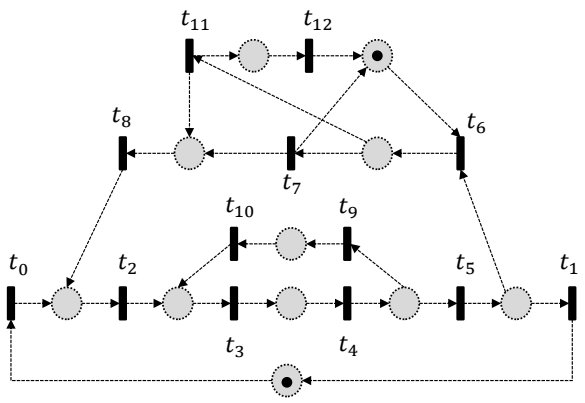


Figure 4.16: Example 4.11; there are two backward loops in the main process t0 $\longrightarrow$ t1

The sequence is far too short to observe all possible interleavings; nevertheless, the exploration of $\mathbb{D}_2$ *is sufficient to obtain the net of Figure 4.17, which perfectly exhibits the concurrency. Namely,* $|L^2_{Exc}| = 34$*; there are missing 34 interleavings of length 2 in the sequence (such as* d_1d_3*,* d_1d_4 *or* $d_2d_5 \ldots$*).*

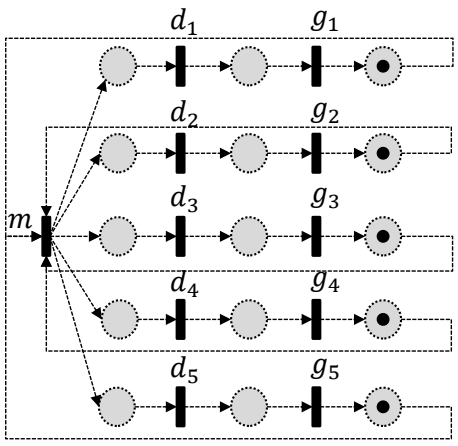


Figure 4.17: Example 4.12: 5 fully concurrent processes, synchronized by m

4.6.2 Complete approach

Examples including observable behaviour are proposed. First, the limiting example of the end of Chapter 2 is reviewed, to illustrate the improvement over the multiple rules approach.

4.6.2.1 Example 2.1 cont.

Example 4.13 (Example 2.1 cont.)**.** *Looking back at Example 2.1, the initial net for the discovery is shown in Figure 4.18(a), and the firing sequence is:*

$$S = t_1t_2t_1t_2t_3t_4t_5t_1t_2t_6t_1t_2t_1t_2t_3t_4t_5t_1t_2t_6t_1t_2t_1t_2t_3t_4t_5t_6t_1t_2t_1t_2t_1t_2t_3t_4t_5t_1t_2t_6$$

The study of $\mathbb{D}_2$ *leads to net (b), where 2 connected components are discovered.*

Notice the place between t_3 and t_6 even though the two transitions were never observed consecutively. It is required to push the discovery to $\mathbb{D}_4$ to discover net (c) with the choice between t_1 and t_3.

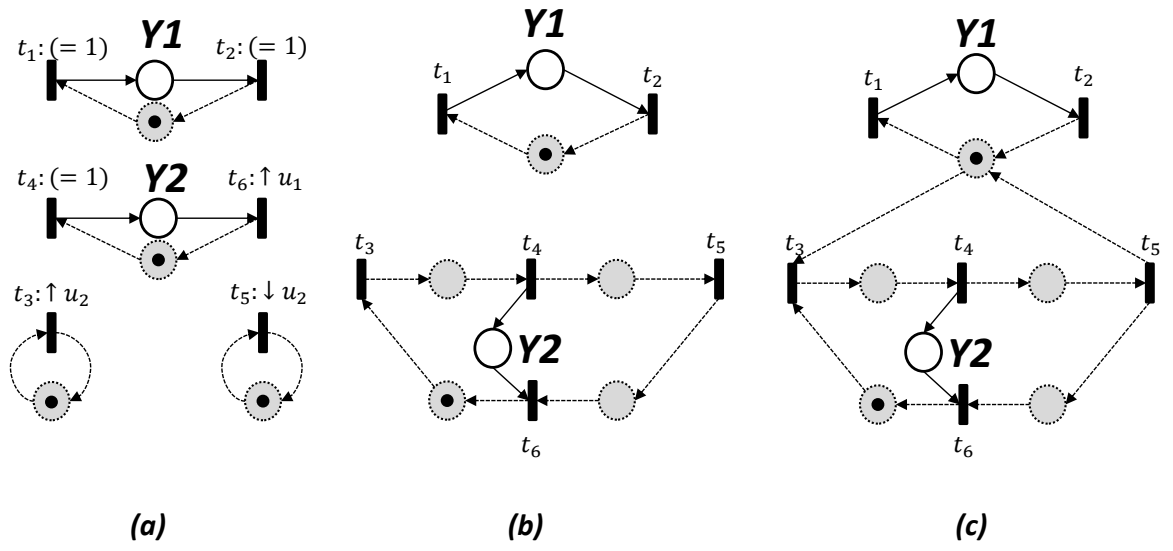


Figure 4.18: Example 4.13: (a)Observable part; (b)Solution $\mathbb{D}_2$; (c)Solution $\mathbb{D}_4$

The chariots example is now extended to include different typical behaviours of real closed-loop systems, and see how they are represented in the model.

4.6.2.2 Extended chariots example

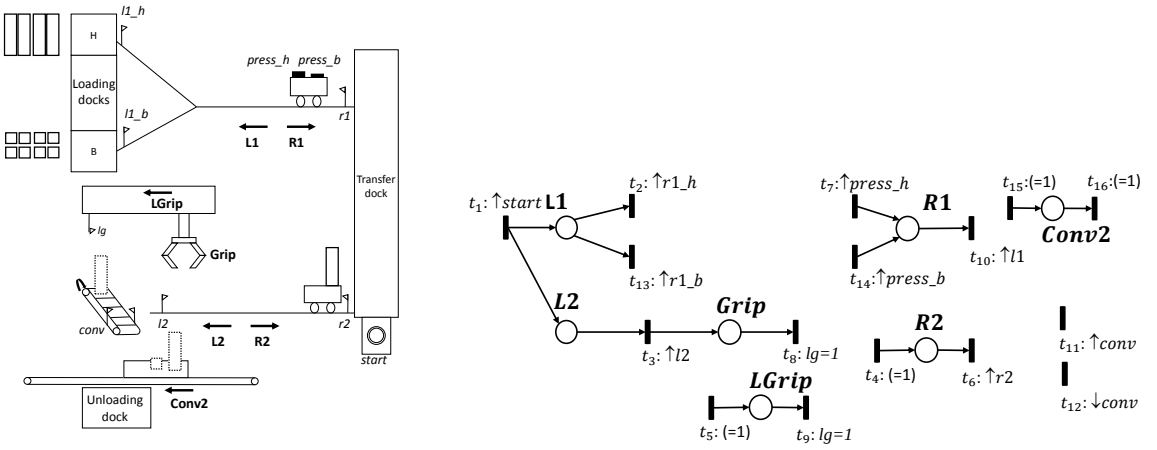


Figure 4.19: Extended chariots example, with its observable behaviour

Example 4.14. *In this example, the future of the box transported by the second chariot is extended. Once it reaches r2, the gripper handles the box (Grip), waits for 1s after the Grip, and moves it to the conveyor (LGrip). The chariot waits for 5s before going back*

home (R2). After the release of the box, LGrip is stopped, leading to the return of the gripper to its original position (monostable pre-actionner), and the box is transferred on an exit conveyor. This conveyor is always running. A sensor (conv) detects the arrival of the box at the end of the conveyor, hence in the unloading dock. Once three boxes have been brought to the dock, the second conveyor (Conv2) is activated to evacuate the boxes. The system is presented in Figure 4.19

There are seven outputs $\mathbb{Y} = \{$L1, R1, L2, R2, LGrip, Grip, Conv2 $\}$ and ten inputs $\mathbb{U} = \{$l1_h, l1_b, r1, l2, r2, lg, conv, start, press_h, press_b$\}$. Its operation has been observed during the process of 10 boxes of different sizes. The observable behaviour is shown as well in Figure 4.19.

One quite connected fragment implies 3 outputs, whereas the four other outputs are alone. Notice the condition (=1) on t_4 and t_5, mirroring the fact that the rising edges of LGrip and R1 are delayed, and not directly caused by an input event. The input conv has not been connected to any output; two transitions labelled with ↑conv and ↓conv are created. The observed firing sequence S of length 126 is

$$\begin{aligned} S = & t_1t_2t_3t_4t_5t_6t_7t_8t_9t_{10}\ t_1t_{11}t_2t_{12}t_3t_4t_6t_5t_7t_8t_9t_{10}\ t_1t_3t_{11}t_{12}t_2t_5t_4t_6t_7t_8t_9t_{11}t_{10} \\ & t_{12}t_{15}t_{16}t_1t_{13}t_3t_5t_8t_{11}t_{12}t_9t_4t_{14}t_6t_{10}\ t_1t_3t_{13}t_5t_8t_9t_{11}t_{12}t_{14}t_4t_6t_{10}\ t_1t_2t_3t_4t_5t_6t_7t_8t_9t_{10} \\ & t_1t_{11}t_2t_{12}t_{15}t_3t_{16}t_4t_6t_5t_7t_8t_9t_{10}\ t_1t_3t_{11}t_{12}t_2t_5t_4t_6t_7t_8t_9t_{11}t_{10} \\ & t_{12}t_1t_{13}t_3t_5t_8t_{11}t_{12}t_9t_{15}t_{16}t_4t_{14}t_6t_{10}\ t_1t_3t_{13}t_5t_8t_9t_{11}t_{12}t_{14}t_4t_6t_{10} \end{aligned}$$

.

Algorithm 4.3 is applied, with $|T| = 16$, which means $|(2^T - \{\emptyset\})^2_{\not\emptyset}|$=21Mio. Firstly, the 120 cases in $\mathbb{D}_2$ are tested, and 31 mutual dependencies are discovered. These dependencies lead to the addition of 62 places. After the deletion of implicit places, only 14 unobservable places are kept. The result is presented in Figure 4.20.

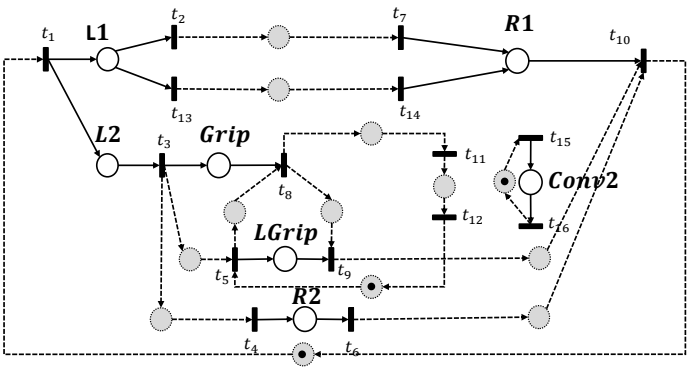


Figure 4.20: The resulting net after the exploration of $\mathbb{D}_2$.

The net consists in two strongly connected components. If the focus is set only on the biggest connected component, interesting behaviours can be noticed. Even though (t_2, t_7) and (t_{13}, t_{14}) were never consecutively observed in the firing sequence, unobservable memory places have been identified. Parallelism between the movements of the

chariots and the gripper is explicit (transitions t_1, t_3 and t_8 having multiple post places; t_{10} acting as a synchronization). The timed behaviours are agregated in the unobservable pre-places of t_4 and t_5. The duration of the stay of the token in the places is not specified, since it is out of the scope of this work.

The second conveyor is not connected to the main process yet. Its behaviour is conditionned by a counter. To connect the net, the exploration is pursued. A strongly connected component is finally reached for $\delta = 5$. For instance, the mutual dependency $\{t_9, t_{16}\} \leftrightarrows \{t_{13}, t_{15}, t_7\}$ is discovered, but it is hard to interprete.

Less than 4% of the search space has been explored after $\delta = 5$. While the precision of the identified nets naturally always increases, the structural complexity of the identified nets has risen quickly and the nets are too complicated to be displayed here, their properties are resumed in Table 4.1.

Net	Precision		Simplicity			
	$\frac{\lvert L^2_{Exc}\rvert}{\lvert L^2_{Obs}\rvert}$	$\frac{\lvert L^3_{Exc}\rvert}{\lvert L^3_{Obs}\rvert}$	Places	Edges	CNC_{PN}	s.c.
$\mathbb{D}_2$	4.13	31.1	20	42	1.17	
$\mathbb{D}_3$	2.88	12.7	27	63	1.47	
$\mathbb{D}_4$	1.95	4.48	30	79	1.72	
$\mathbb{D}_5$	1.48	2.55	36	113	2.17	✓

Table 4.1: Evaluation of the precision and simplicity metrics on the discovered nets for the chariots.

Finding counters is a hard task in a blackbox approach, without the knowledge of their presence. By locally injecting some knowledge (becoming a greybox approach), a possible way of representing them in our given class of IPNs is by adding self loops, as shown in Figure 4.21. Notice that the number of firings of t_{11}/t_{12} required before firing t_{15} is nevertheless not explicited. It can not be inferred by the current method.

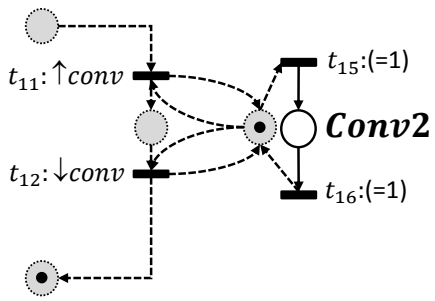


Figure 4.21: A representation of the counter of the second conveyor, using self-loops

4.6.2.3 MSS chain

Finally, the identification of the MSS chain is ended in this section by adding the unobservable behaviour to the observable model computed at the end of Chapter 3 (Figure 3.17). The system consists in 73 I/Os (43 inputs and 30 outputs), and the observed

data consisted in a sequence w of length 63797 I/O vectors. After the computation and simplification of the observable behaviour, the observable model consists in a net of 30 observable places and 101 transitions. Additionally, the length of the firing sequence is $|S| = 33621$. All computations are ran on a laptop (Intel® Core™ i5-3380M CPU @ 2.90GHz x4, 8Go RAM); the identification of the observable behaviour took 15s.

There are $|T| = 101$ transitions. Given the high number of transitions, the search space is already huge, even for low values of δ. Notably $|\mathbb{D}_2| = 5050$, $|\mathbb{D}_3| = 500k$, and $|\mathbb{D}_4| = 28.5Mio$. The model for $\delta = 2$ was computed in 5 minutes, for $\delta = 3$ in 1 hour, and for $\delta = 4$ in 14 days. Strong connexity was finally achieved for $\delta = 4$, and the properties of the resulting nets are shown in Table 4.2.

Only the model discovered for $\delta = 2$ is shown here, in Figure 4.22, even if not connected. Only degree 2 places have been added to this net, allowing the discovery of sequential processes such as the chute. Areas have been expertly delimited from the labelling of places and transitions to ease the understanding, which is hard when no clue is given due to the size. One can notice the central position of the conveyor of the third station, connected to the gripper, the two presses, and the storage station. The concurrency between these processes can be discovered without trouble.

One input (2B20, inductive sensor of station 2) and one output (2Y04, relay of the conveyor of station 2) could not be connected. There is no clue as to how the two small fragments could be connected to the big ones; a connexion is discovered with degree 4 places, but requires two weeks of calculation.

Computing a monolithic model of a concurrent system therefore shows its limits: it is computationnally expensive, and the resulting model might be hard to understand due to its size and the lack of demarcation. Demarcations could be automatically inferred by a Graph Partitioning algorithm applied to the IPN, but the model would still need being computed.

Net	Simplicity			
	Places	Edges	CNC_{PN}	s.c.
$\mathbb{D}_2$	104	297	1.45	
$\mathbb{D}_3$	115	335	1.55	
$\mathbb{D}_4$	202	699	2.31	✓

Table 4.2: Evaluation of the simplicity metrics on the discovered nets for the MSS.

4.7 Discussion

Regarding previous inquiries dealing with IPN identification, the approach proposed in this section brings the guarantee of fitness (missing in the previous rule-based approach [Estrada-Vargas et al., 2015]), and discovers efficiently concurrency (major difficulty of [Meda-Campana, 2002], [Estrada-Vargas et al., 2014] or [Estrada-Vargas et al., 2015]).

Some approaches of the litterature were discarded for identification of real systems since they did not take the reactive, observable behaviour into account. However, the formulation of the problem in this section does not imply the observable part. Theorem 1 and the principle of the discovery can be compared more thoroughly to different approaches solving similar problem.

Regarding language approaches, it is worth noting that our input is a sequence, and that it is possible to build a language by prefix-closing the sequence:

$$L^{Obs} = \{\varepsilon, s_1, s_1s_2, s_1s_2s_3, \ldots, s_1s_2 \ldots s_{|S|}\}$$

The longest word in this language has length $|S|$, and language approaches based on ILP resolution ([Giua and Seatzu, 2005], [Cabasino et al., 2007]) have theorical exponential complexity regarding this maximal length. Since the length of S reflects the duration of the observation, it is often big. ILP based approaches are computationnaly too hard for our problem. Likewise, it is possible to build a sequential automaton like [Kella, 1971], which has $|S|$ states, and algorithms developped in region theory ([Badouel et al., 2015]) have exponential complexity regarding the number of states.

Besides, language-based approaches aim for maximal precision (even language equalities in *synthesis* approaches). The nets computed are generalized (weighted edges) and contain multiple tokens to achieve this goal; concurrent or sequential processes of a real system are hard to decipher in this class of nets, making the method unfitting for reverse engineering.

Process mining deals with a very close problem. The different cases present in the log can be concatenated to form a sequence, where the firings of transitions correspond to activities. Hence, the approach developped here can also be applied to process mining. Reversely, abstraction-based approaches from process mining aim at finding relations between activities in the log, and can therefore also be applied to firing sequences.

However, fitness is a mandatory quality in our problem (the net must be able to reproduce the sequence), which is not the case in process mining: better have a simpler model that reproduces 80% of the log than a complex model that reproduces 100%. (*normal behaviour could be defined as the 80% most frequently occuring traces* [Van der Aalst, 2011a]). Therefore, most rule-based approaches (such as α[Van der Aalst et al., 2004] or α^{++} [Wen et al., 2007]) do not guarantee fitness, and additional rules are provided to adapt the model if parts are missing (conformance checking). However, they often propose polynomial algorithms, mandatory to deal with the sheer number of activities present in the log.

Typically,in [Tapia-Flores et al., 2014], T-invariants are inferred from the sequence. The identification is based on rules similar as [Estrada-Vargas et al., 2015], using T-invariants to distinguish choices and concurrency, and exploitable in polynomial time. A model adjustment step is required after the identification step, and is mandatory to verify fitness.

Most recent inquiries in process mining deal with the specific class of block-structured workflow nets. In [Leemans et al., 2013] is proposed a discovery algorithm (Inductive miner) running in polynomial time and guaranteeing fitness. However, if the log can not be represented by this restrictive model class, silent transitions are added to ensure fitness, which are incompatible with our problem.

The approach proposed in this chapter is therefore located halfway: no language is required, fitness is guaranteed, and the algorithmic complexity is limited.

4.8 Conclusion

The discovery procedure proposed in this chapter is based on a single, generic caracterization of admissible places to add to the net. Compared to rule-based approaches, the reproducibility of the firing sequence is guaranteed, and compared to language-based approaches, the emphasis is put more on the readability of the model than on its precision. Concurrency can easily be discovered; undeterminism and choices require more computational effort.

The main drawback is the exponential size of the search space; an efficient heuristic is designed in accordance with the reverse engineering objective, to limit the exploration and the computation cost. Nevertheless, as shown on the MSS, monolithic models of big systems are hard to understand due to their sheer size, and are still expensive to compute (14 days of computation to reach a strongly connected, but uncomprehensible model). To deal with such systems, distributed identification should be considered, as shown in next chapter.

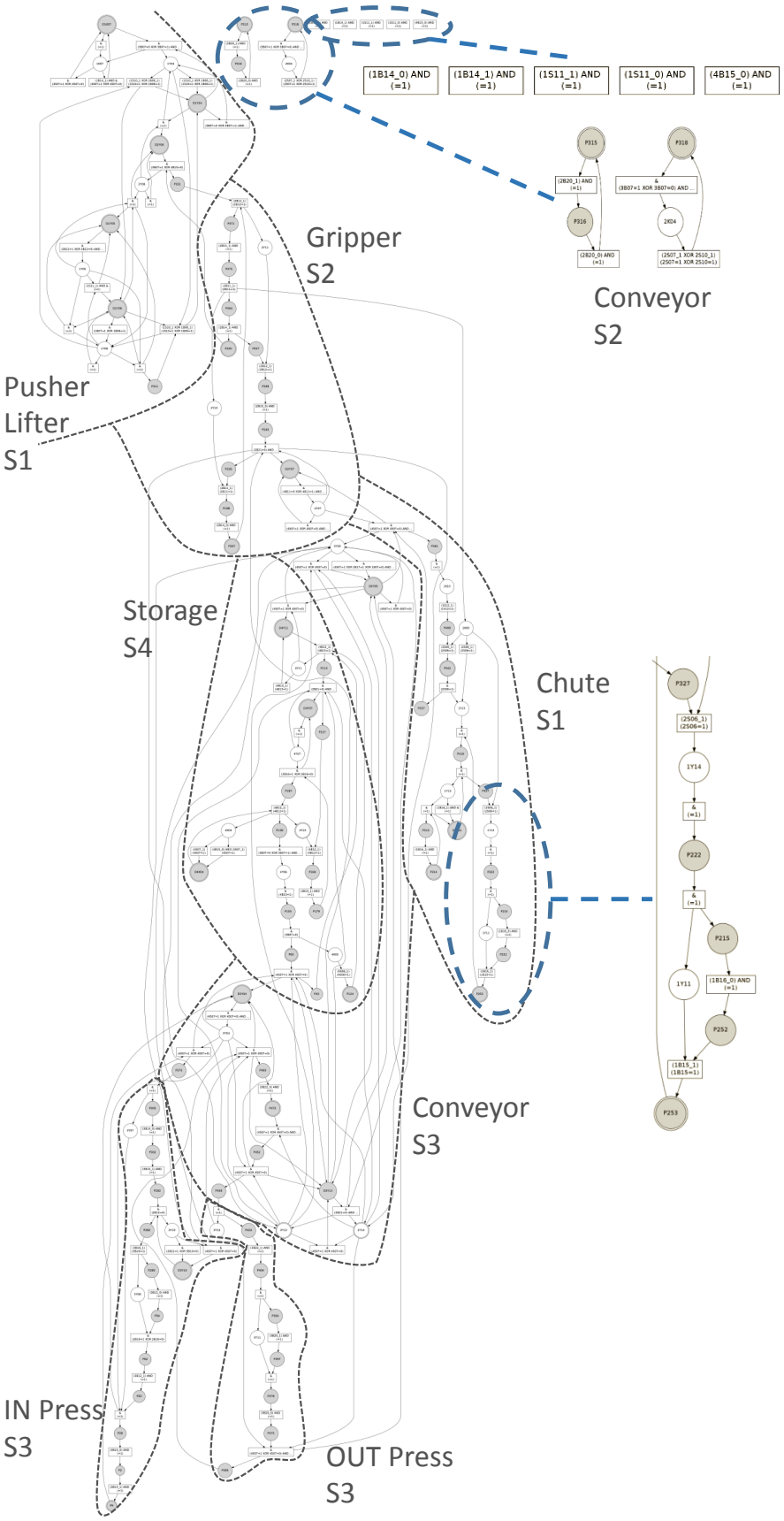


Figure 4.22: Monolithic IPN model obtained after the exploration of $\delta = 2$. 5 transitions and two fragments remain unconnected to the remainder of the net

Chapter 5
Automated partitioning for distributed identification

Introduction

In this chapter, we wish to improve the scalability of the method even further, by exploring partitioning approaches instead of computing monolithic models. Monolithic models are expensive to compute due to the sheer number of transitions, whereas smaller models are easier to compute. Besides, the partition also provides insight on the decomposition of the system into subsystems, by splitting and grouping the components.

Instead of considering the problem at the I/O level, the idea is to use the observable fragments who already provide good insight on related I/Os. The formalization of the problem is proposed, then a clustering approach is presented to provide good partitioning solutions. The method is illustrated using the MSS system.

5.1 Statement of the partitioning problem

5.1.1 Objective of the partitioning

As seen in the previous chapter, a monolithic model is expensive to compute, and hard to read due to its sheer size. Distributing the identification procedure onto subsystems helps reducing both the algorithmic cost and the structural complexity. Each subsystem being smaller than the complete one, the resulting models are easier to compute and to read. The objective is not necessarily to rediscover the monolithic model by composing the distributed models; from the point of view of an engineer, distributed models might as well be a good way of getting insight on the system.

Distributed identification consists in splitting the system into subsystems, and run the identification procedure on each subsystem SUB_k, as illustrated by Figure 5.1. The input of the identification is always an observed vector sequence w_k. The vector sequence w observed on the full system is therefore projected into partial vector sequences w_k corresponding to each SUB_k. The projection function is illustrated by the following example:

Example 5.1 ([Schneider and Litz, 2014]Projection function)**.** *Let* $\{io_1, io_2, io_3\}$ *be the system,* $SUB = \{io_1, io_2\}$ *the considered subsystem, and the vector sequence* w*:*

$$w = \begin{bmatrix}0\\0\\0\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}$$

Then, the projected vector sequence w_{SUB} *is:*

$$w_{SUB} = \begin{bmatrix}0\\0\\-\end{bmatrix}\begin{bmatrix}1\\1\\-\end{bmatrix}\begin{bmatrix}1\\0\\-\end{bmatrix}$$

The first two vectors of w *being identical when considering only the first two I/Os, they are merged into one in the projected sequence* w_{SUB}*.*

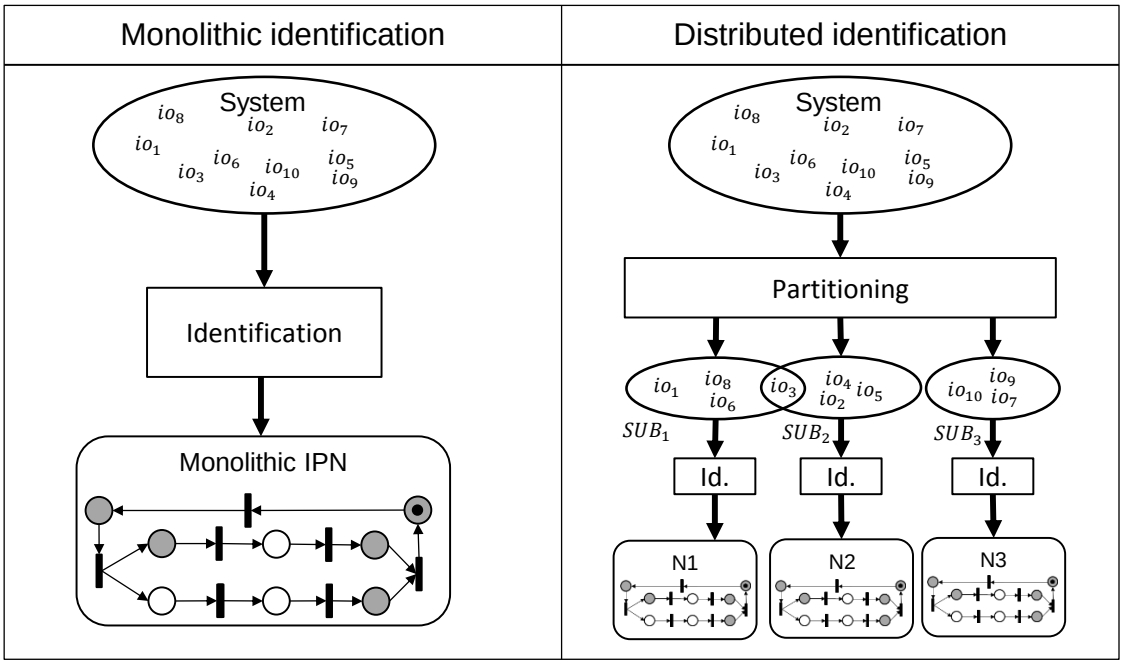


Figure 5.1: Principle of the distributed approach, based on a partition of the system.

Using the projection function, any given partition can be used for a distributed approach, and distributed models can always be computed. Namely, an expert partition for the MSS system is given in Figure 3.2, where the MSS is decomposed into 11 subsystems, all containing from 5 to 8 I/Os (no overlapping of the 73 I/Os). The identification procedure was run on each of the eleven subsystems. The calculation took 3m30s (compared to 14 days for the monolithic model), and the average network complexity of the 11 resulting nets is 1.35 (compared to 2.31 for the monolithic model). Two of the eleven subsystems (Lifter and Transport Station 3) could however not be modelled by strongly

connected nets, hence their might exist a better partition.

Expert partitons are not always available, namely in a blackbox approach. Therefore, the problem of automatic partitioning is adressed. It consists in building a partition of the system on the I/O level. An adequate partition ensures that each subsystem can be identified with qualities relevant to the objective. In [Roth et al., 2010a], the minimization of the exceeding language is pointed as an objective quality related to the objective of fault diagnosis. For reverse engineering, each subsystem should be modelled by a strongly connected IPN, simple to read, and computable in a reasonable time. Furthermore, the subsystems should regroup I/Os that are functionally related in the real system; overlapping of systems is allowed, since some I/Os are located at the interface between two subsystems, and could therefore be shared.

A first formulation of the partitioning problem for reverse engineering is given; since shared-I/Os are allowed, it is more exactly a *cover* problem:

Finding a cover *Consider a DES consisting in m I/Os $\{IO_1, \ldots, IO_m\}$. Compute an I/O-cover $Cover = \{SUB_1, \ldots, SUB_N\}$, with the constraints:*

1. $\forall i \in [\![1, m]\!], \exists SUB_k, IO_i \in SUB_k$
2. *Each model N_k built on a SUB_k is strongly connected*

and optimal regarding the two criteria:

1. *minimize* $CNC_{Avg} = \frac{1}{N} \sum_{k=1}^{N} CNC(N_k)$
2. *minimize* N

The first constraint implies that each I/O belongs to at least one subsystem, and that overlapping is allowed. The second one defines what makes a solution acceptable: since the criterion chosen to stop the unobservable discovery is the strong connexity of the identified net, each subsystem must result in a strongly connected identified net as well.

Numerous covers satisfy these constraints; to choose an adequate partition, optimal criteria are added. The first one aims for simple nets (see Definition 4.6). However, a solution optimal regarding this criterion consists in building a bijection between the I/Os and the subsystems, each subsystem contaning exactly one I/O. Resulting models are presented in Figure 5.2. Therefore, the second criterion is added to find a compromise between the size of the distributed models and their simplicity, as minimizing the number of subsystems implies maximizing the number of I/Os in each subsystem, hence its size. A solution optimal regarding this second criterion only is the monolothic model, provided it can be computed.

Since the optimization problem is multicriteria, there is no single solution that simultaneously optimizes each objective. Instead, the best solutions form a Pareto frontier, whose shape is sketched in Figure 5.3. The two extreme points are known: on one hand

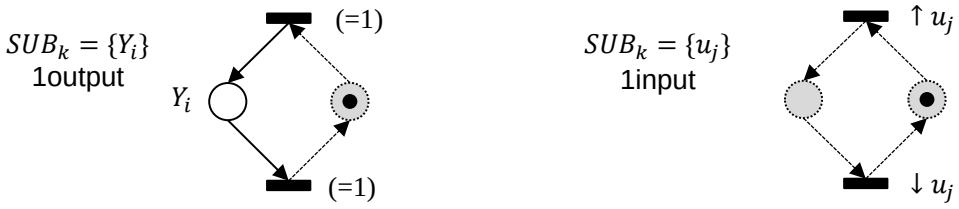


Figure 5.2: Identified net for SUB_k consisting in either 1 output or 1 input

the solution performing a bijection of subsystems onto I/Os (having an average network complexity equal to 1, the minimal achievable), and on the other hand the monolithic solution (only one subsystem). Both these extremes, although Pareto optimal, are not interesting; the choice of a solution among the possible ones will be discussed later. Additionally, notice that the lower the number of subsystems, the more I/Os they contain, and the bigger the resulting models, implying longer computation durations.

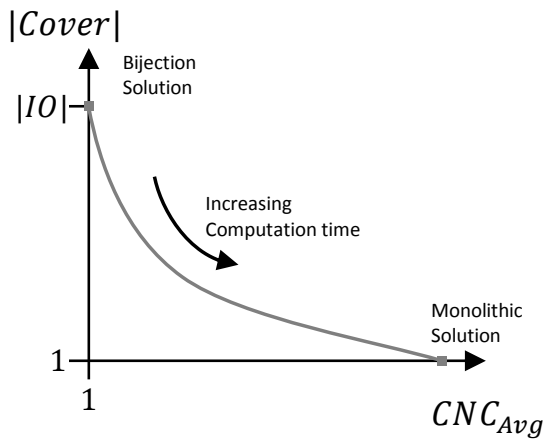


Figure 5.3: Shape of the Pareto frontier of optimal solutions of the cover problem

Notice that the number of partitions is not fixed in this problem. Suppose that the SUB_k are disjoint; in this case, the total number of I/O-partitions is given by Bell's number:

$$B_m = \sum_{k=1}^{m} S(m,k)$$

where $S(m,k)$ is the Stirling number of the second kind:

$$S(m,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}C_k^j j^m$$

Notably, $n = 73$ and $B_{73} = 2.15.10^{77}$ for the MSS system, which means that it is impossible to study all possible partitions. Besides, since overlapping of subsystems is allowed, that number is even higher.

The problem being formulated as an optimization problem, heuristics and optimization techniques have been considered to solve it, and reviewed in next section.

5.1.2 Related work

A pioneer approach for automated I/O-partitioning of a DES is proposed in [Roth et al., 2010a]. It uses simulated annealing as an optimization technique, and is developped to find concurrent behaviour in the observed sequences, and split the system into subsystems with minimal internal concurrency. Minimizing internal concurrency limits the growth of the language with the observations, and enables convergence of L^n_{Obs} for higher values of n. Limiting language growth is consistent with the diagnosis purpose, as a direct consequence is the minimization of false alerts (see Section 1.3.2.1).

The optimization function is designed in that way. H sequences are observed ($p \geq 2$), and $|W^{n,h}_{Obs,SUB_k}|$ is the number of words of length n observed in all sequences up to sequence number h, projected on the I/Os of SUB_k. Given a partition P, the proposed optimization function to minimize is :

$$J^n_1(P) = \frac{1}{N} \sum_{SUB_k} \sum_{h=2}^{H} (\sqrt{h}(|W^{n,h}_{Obs,SUB_k}| - |W^{n,h-1}_{Obs,SUB_k}|))$$

For instance, given a SUB_k, if all sequences after the first one exhibit no new word of length n, *i.e.* all possible interleavings are observed in the first sequence. SUB_k exhibits low concurrenct, and contributes indeed as 0 in the function.

A second optimization function consists in minimizing the average branching degree (BD) of the resulting models:

$$J_2(P) = \sum_{SUB_k} BD(NDAAO_{SUB_k})$$

where $NDAAO_{SUB_k}$ is the automaton identified for the subsystem SUB_k, and

$$BD(NDAAO_{SUB_k}) = \sum_{x \in X} \begin{cases} 0 & \text{if } |f(x)| \leq 1 \\ |f(x)| - 1 & \text{if } |f(x)| > 1 \end{cases}$$

where $|f(x)|$ is the number of transitions leaving state x. Low values of $|f(x)|$ are typical of sequential systems with few choices and interleavings, hence the branching degree also depicts the concurrency.

These criteria are oriented towards minimizing the internal concurrency, which is not mandatory for our problem, as IPN can explicit concurrency while remaining simple to understand, unlike automata. Besides, since the method developped in this thesis abstracts concurrency in the firing functions and non observable places, resulting in compact models. J_1 might be a criterion too sensitive, as illustrated by the next example:

Example 5.2. *Suppose the $H = 3$ following event sequences on $SUB_k = \{u_1, y_1\}$:*

$$E_1 = [\downarrow u_1][\uparrow u_1 \uparrow y_1][\downarrow y_1][\downarrow u_1][\uparrow u_1]$$
$$E_2 = [\downarrow u_1][\uparrow u_1 \uparrow y_1][\downarrow u_1][\downarrow y_1][\uparrow u_1]$$
$$E_3 = [\downarrow u_1][\uparrow u_1 \uparrow y_1][\downarrow u_1][\uparrow u_1][\downarrow y_1]$$

For $n = 2$, $J_1^2(SUB) = \sqrt{2}.3 + \sqrt{3}.2 \simeq 7.7$ and $BD = 2$; the resulting identified automaton is shown in Figure 5.4(a). The two I/Os could be discarded as concurrent and not regrouped. However, notice that input u_1 always causes output y_1. The IPN identified for SUB is shown in Figure 5.4(b), and is satisfying regarding simplicity, therefore the two I/Os should be kept together.

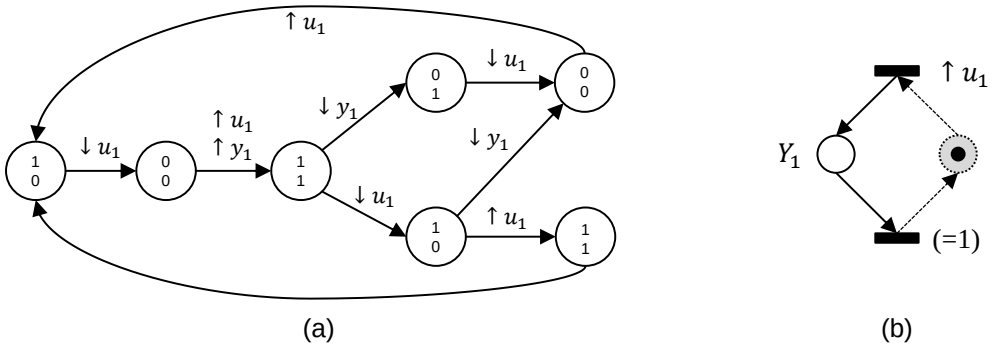


Figure 5.4: (a) The NDAAO and (b) the IPN built from $SUB_k = \{u_1, Y_1\}$

Three important points are highlighted by this example:

- Language-based criteria designed to avoid concurrency are too sensitive to interleaving for our method
- Causal I/Os should not be separated
- The roles of inputs and outputs should be differentiated

A direct consequency is that the observable behaviour, already expressing the causalities, offers a solution to the partitioning problem and provides I/O blocks that should not be separated. This reflexion leads to the reformulation of the problem in section 5.1.3.

The approach proposed in [Roth et al., 2010a] aims at computing directly a partition. An extension is proposed in [Schneider and Litz, 2014]. It consists first in computing numerous acceptable subsets SUB_k, then to synthesise a partition from all computed subsets by solving a Cover Set problem. An overview is shown in Figure 5.5.

Optimal partitioning uses J_1 previously introduced as objective function, and proposes a Hill climbing algorithm to build the subsets in OPT.

Causal partitioning is a first heuristic proposed to group I/Os. The idea is to compute a distance between each I/O pair, then regroup the closest I/Os into subsets. Given H sequences projected on an I/O pair $\{io_1, io_2\}$, and $q \leq H$ the number of occurences of the most frequent sequence, the distance is defined as:

$$d(io_1, io_2) = 1 - \frac{q}{H}$$

Notably, $d = 0$ iff all H projected sequences are identical. Then, fixing a threshold for the distance d_{max}, subsets from CAU are build such that $\forall (io_i, io_j) \in SUB_k^2, d(io_i, io_j) \leq d_{max}$.

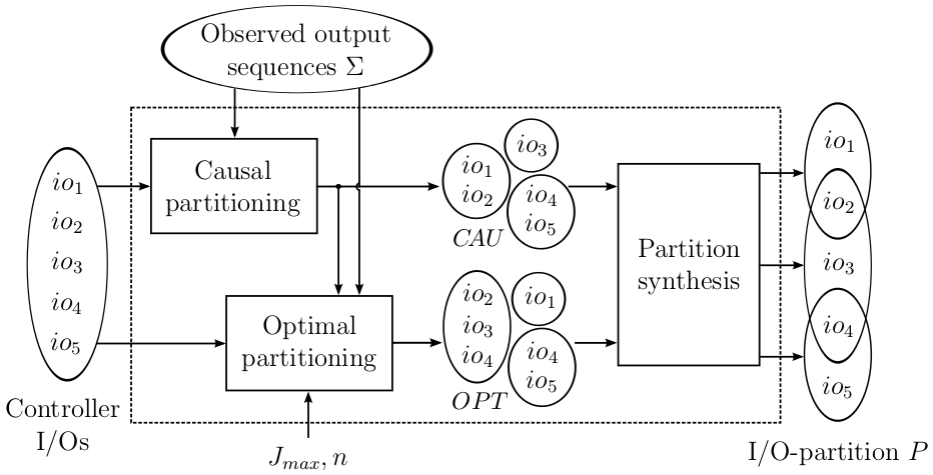


Figure 5.5: Overview of the partitioning approach from [Schneider, 2014]

This distance nevertheless also suffers from the same sensitivity problem, exhibited by Example 5.2. The subsets either from CAU or OPT can break direct causalities between inputs and outputs by separating them, which makes even the causal approach unusable in this thesis.

The problem of partitioning has also been considered in the field of Process Mining (see Section 1.3.3.2), where models are mined from huge flows of data with numerous possible activities. A *Divide-and-Conquer* approach is therefore proposed in [Van der Aalst, 2013b] to ease either conformance checking or process discovery. Regarding the latter, the principle is to split the set of activities into overlapping subsets, then project the log onto the subsets to get sublogs, and perform the discovery on each sublog. Finally, the identified submodels are composed into the full model, *i.e.* including all activities.

Splitting the set of activities is perceived as a Graph Partitioning problem in [Carmona et al., 2009]. First, a Causality Graph is build, based on the causal relationship: the activities are the nodes, and two nodes are connected iff one activity causes the other. Graph partitioning methods can then be used to decompose the graph into subgraphs, each subgraph representing a subset of causal activities. The objective is to minimize the RatioCut, *i.e.* the number of edges that are cut off to break the graph into subgraphs. Each subgraph then corresponds to a set of activities. This method is especially adapted when process mining algorithms exploiting the causal relationship are used, such as α.

In the context of this thesis, such a method can be applied once the observable behaviour is discovered, by using the transitions as nodes. It was however shown in Chapter 2 that the causality relationship is unprecise. Due to massive concurrency and uncomplete observation of the system, concurrent relationships are hard to infer, and transition pairs might be mistakenly classified as causal whereas being actually concurrent. Furthermore, there is no guarantee that the partition formed by any graph partitioning method leads to strongly connected nets, as imposed in our problem.

5.1.3 Mapping I/Os and observable fragments

Since the observable fragments provide the causal I/Os subsets that should not be separated, the partitioning procedure must be done after the computation of these fragments. The idea is now to split the observable fragments into separate observable models, on each of which the unobservable discovery is run. They will be called *blocks*, and the set of blocks must cover all observable fragments, forming a *partition* of the observable model. The number of blocks is *apriori* unknown. The firing sequence S which comes along the observable fragments is global, and must remain consistent with each of the blocks. Therefore the separation of the fragments must not lead to creation, duplication or removal of transitions. Furthermore, the observable places must remain unique as well.

The notions to achieve the decomposition into blocks are now introduced.

Definition 5.1. *An observable fragment F_i consists in a connected component of the observable model. It might either contain places and transitions, or be an isolated transition.*

For instance, the observable model of the MSS in Figure 3.17 consists in 37 fragments, including 24 isolated transitions, and 13 fragments having at least one place.

For the partitioning objective, connected components should not be separated, otherwise direct causalities might be lost. Multiple connected components could however belong to the same subsystem. Besides, observable places are unique, and transitions must not be duplicated. Hence, a fragment can not belong to multiple subsystems. The definitions of blocks, which are models of subsystems, and of a partition ensue:

Definition 5.2. *A block B_k is a non-empty set of observable fragments $\{F_1, \ldots, F_{|B_k|}\}$.*

An observable partition is a set of disjoint blocks $PAR = \{B_1, \ldots, B_{|PAR|}\}$ covering all observable fragments, i.e.

$$\forall F_i, \exists! B_k / F_i \in B_k$$

Before defining the mapping of blocks on I/Os, the classification of I/Os is precised. D is the set of input events that have not been causally associated to any outputevent, i.e. that do not appear in any firing function (See page 48)

Definition 5.3. *The I/Os are split into the output set $\mathbb{Y}$, and the input set $\mathbb{U}$. The input set is further divided into D, the set of unassigned inputs, and $\mathbb{U} \backslash D$. An unassigned input is an input whose events or levels do not appear in any firing function of any output.*

Isolated transitions can only be labelled by unassigned input events. However, these events can be observed with output events as well, meaning that transitions connected to observable places can be labelled with input events both from D and $\mathbb{U} \backslash D$. If these observations are not spurious, the transitions remain after the reduction (see Chapter 3), and the unassigned input can be considered as connected to the simultatneous output. Likewise, if an isolated transition is labelled by two input events, they can be considered connected. Therefore, the set D is further divided:

Definition 5.4. *An unassigned input $u_D \in D$ is said connected iff there exists a transition whose firing function contains u_D, and either connected to an observable place, or labelled by another input event i.e.*

$$\exists t \in T, u_D \in \lambda(t) \wedge ((\exists p \in {}^{\bullet}t \cup t^{\bullet}) \vee (\lambda(t) - u_D \neq \emptyset))$$

The set of connected inputs is noted D_{Conn}. Its complementary, the set of solitary inputs, is noted $D_{Sol} = D/D_{Conn}$.

A direct consequence of this definition is that a solitary input is only associated to two transitions, who are respectively and only labelled by its rising and falling edge. The mapping of fragments and blocks onto the I/O set is now defined.

Definition 5.5. *Let B_k be a block, and F_i an observable fragment. Each place p is associated to an output $\varphi(p)$, and each transition t to the inputs in its firing function $\lambda(t)$. The inputs used in the firing function of a transition t are designed by $\{\lambda(t)\}$. Two mapping function Map_F and Map_B are defined as follows, mapping the elements onto the I/Os:*

$$\begin{aligned} Map_F : \ F_i &\longrightarrow \bigcup_{p \in F_i} \varphi(p) \bigcup_{t \in F_i} \{\lambda(t)\} &\subseteq (\mathbb{Y} \cup \mathbb{U}) \\ Map_B : \ B_k &\longrightarrow \bigcup_{F_i \in B_k} Map_F(F_i) &\subseteq (\mathbb{Y} \cup \mathbb{U}) \end{aligned}$$

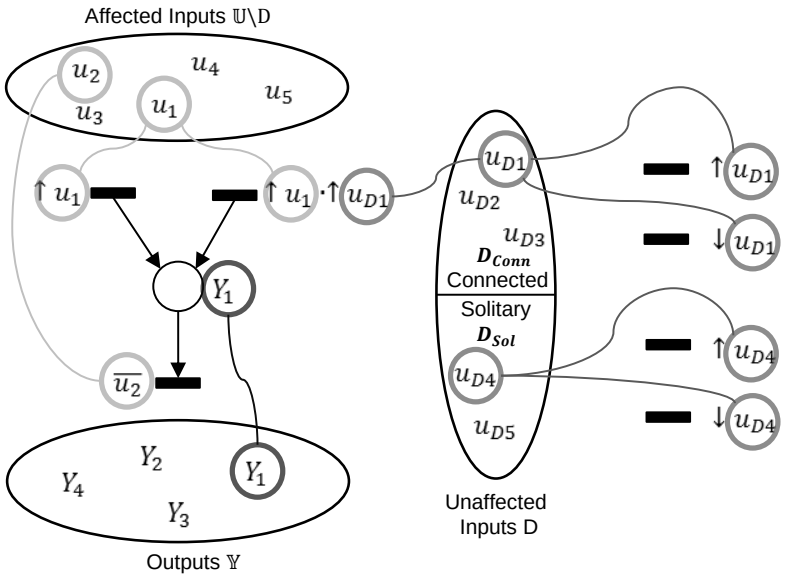


Figure 5.6: Mapping of an observable fragment and isolated transitions on the different I/O sets

The mapping of a fragment is illustrated by Figure 5.6. For instance, F_1 being the left fragment in Figure 5.6, $Map_F(F_1) = \{Y_1, u_1, u_2, u_{D1}\}$. A block is mapped into an I/O subset; however, starting from said I/O subset, the observable model can be

recomputed. To make sure that no transition is altered, the block should be identically rebuilt. This makes the condition of consistency of a block.

Definition 5.6. *A block B_k is consistent if, and only if it is identical to the observable model built from $Map_B(B_k)$*

Naturally, only consistent blocks must be built. A fragment F_i such that $Map_F(F_i) \cap D = \emptyset$ is naturally a consistent block (Type 1). Consistency problems occur when connected inputs (in D_{Conn}) are involved in the mapping of fragments, as illustrated by the next example.

Example 5.3. *Consider the four fragments of Figure 5.7, with $Map_F(F_1) = \{Y_1, u_1, u_{D1}\}$, $Map_F(F_2) = \{Y_2, u_2, u_{D1}\}$,$Map_F(F_3) = Map_F(F_4) = \{u_{D1}\}$. Consider the block $B_1^1 = \{F_1\}$. u_{D1} belongs to $Map_B(B_1^1)$ and is an unaffected input. During the construction of the observable model from $Map_B(B_1^1)$, two transitions corresponding to F_3 and F_4 will be created. Hence B_1^1 is not consistent. However, $B_1^2 = \{F_1, F_3, F_4\}$ is consistent. $B_2 = \{F_2, F_3, F_4\}$ is consistent as well. However, B_1^2 and B_2 share two fragments, and can not make a partition. The only possible partition is directly the block $B_{tot} = \{F_1, F_2, F_3, F_4\}$, which is consistent.*

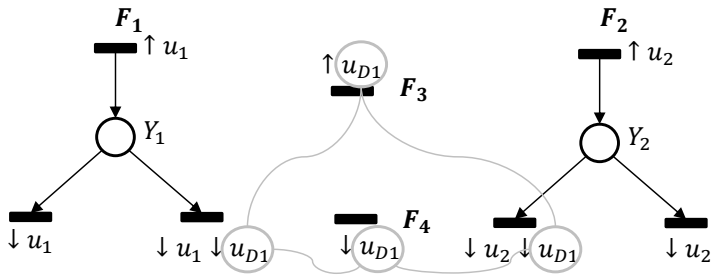


Figure 5.7: Sharing a connected input between two fragments and two isolated transitions.

Consequently, all fragments sharing an input in D_{Conn} are regrouped into the same block (Type 2). Similarly, two isolated transitions sharing a solitary input in D_{Sol} are regrouped in the same block (Type 3). The procedure for building blocks is given by Algorithm 5.1:

Proposition 5.1. *Algorithm 5.1 provides a partition of consistent blocks.*

Proof. Three types of blocks are created by the algorithm. Each Type 3 block B_j built in line 3 consists in a couple of isolated transitions associated to a solitary input u_D, such that $Map_B(B_j) = u_D$. Type 2 blocks built in lines 4-11 are mapped to at least one connected input; said connected inputs do not belong to any other block, avoiding consistency issues. Finally each Type 1 block built in line 12 is built from an observable fragment not mapped to any input in D, hence naturally consistent. □

Algorithm 5.1 Building blocks and a partition

Require: Fragments F_i, mapping functions Map_F, Map_B
Ensure: $PAR = \{B_1, \ldots, B_{|PAR|}\}$ a partition of consistent blocks

```
Build a new block for each pair of transitions sharing an input in D_Sol {Type 3}
for Connected input u_D ∈ D_Conn, not treated yet do
    Find all fragments F_i such that u_D ∈ Map_F(F_i)
    Build B_j = ∪_i F_i; u_D is treated
    while there exists u'_D ∈ Map_B(B_j) not treated yet do
        Find all fragments F'_i such that u_D ∈ Map_F(F_i)
        B_j ← B_j ∪_i F'_i ; u'_D is treated {Type 2}
    end while
end for
Build a new block for each remaining fragment F_i {Type1}
```

Besides being composed of consistent blocks, the partition offers also a solution to the cover problem. Each I/O is attributed to at least one $SUB_k = Map_B(B_k)$ (only outputs in $\mathbb{Y}$ and unassociated inputs in D can not be shared), and the second constraint is verified by the following proposition:

Proposition 5.2. *Let $PAR = \{B_1, \ldots, B_{|PAR|}\}$ be a partition built with Algorithm 5.1. Then, the discovery of the unobservable behaviour of each block B_j leads to a strongly connected model.*

Proof. Type 1 blocks consist in one connected observable fragment. It is sufficient to add the unobservable dual places (see Section 4.4.4) to reach a strongly connected component.

Type 3 blocks consist in two transitions, and are mapped into exactly one input in D_{Sol}. The transitions are connected by two unobservable places, as can be seen in the right net of Figure 5.2.

Finally, type 2 blocks consist in connected fragments, and isolated transitions, which share connected inputs in D_{Conn}. By adding the dual unobservable places, the connected fragments become strongly connected. Then, all transitions sharing an input in D_{Conn} can be connected by two unobservable places which are consistent with the alternance between rising and falling edges leading to one strongly connected component, as illustrated in Figure 5.8. □

Algorithm 5.1 is applied to the observable behaviour of the MSS. 4 pairs of isolated transitions are regrouped with observable fragments, forming 4 Type 2 blocks (B_4, B_8, B_{11}). B_{13} up to B_{20} are Type 3 blocks composed of two isolated transitions, and the remaining fragments build 9 Type 1 blocks, for a total of 21 blocks. The partition is shown in Figure 5.11; grey circles point the regroupments of Type 2 blocks.

This solution is good regarding the minimization of the average complexity, as many fragments are very simple to read after the addition of unobservable behaviour ($CNC_{Avg} = 1.18$). However, the granularity is a bit too fine, as some subsystems have

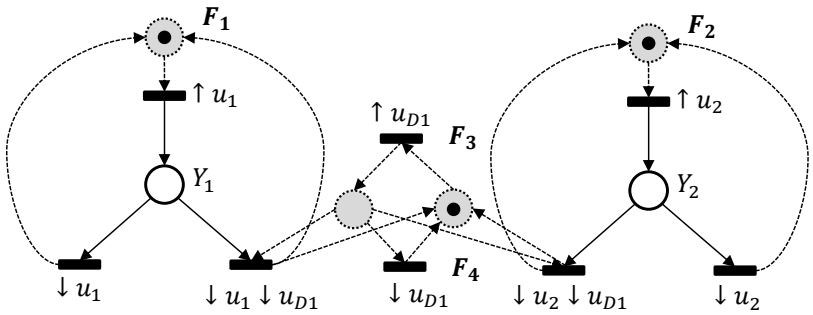


Figure 5.8: Possible unobservable behaviour for a Type 2 Block

very few outputs or solitary inputs. The physical location of the blocks on the MSS is represented in Figure 5.9. Overlappings of inputs were not represented for the sake of readability. Some small blocks are close to one another and should naturally be regrouped.

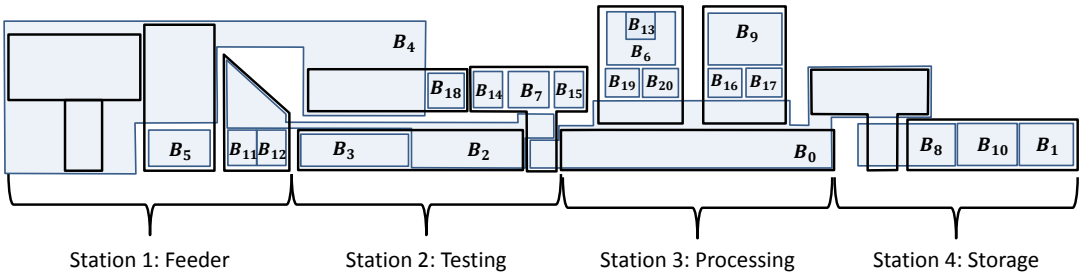


Figure 5.9: Location of the blocks on the MSS

5.1.4 Final formulation

The partitioning problem is reformulated now that a block-composed pre-partition $PAR = \{B_1, \ldots, B_{|PAR|}\}$ is computed from the observable behaviour.

The constraints previously defined on I/Os can be reformulated on the blocks. We aim at finding a repartition of the blocks into subsystems. Each subsystem is a set of blocks, and each block can only belong to one subsystem (to avoid duplicating transitions or places). Finally, each subsystem should be identified as a strongly connected IPN model after the unobservable places are added.

The objective functions remain the same, to ensure each distributed model is balanced between simplicity (minimized with a lot of subsystems) and size (maximized with the monolithic model).

Finding a partition *Consider a set of m blocks $\{B_1, \ldots, B_m\}$. Compute a partition of N subsystems $PAR = \{SSYS_1, \ldots, SSYS_N\}$, with the constraints:*

1. $\forall i \in [\![1, m]\!], \exists! SSYS_k, B_i \in SSYS_k$

2. *The addition of unobservable behaviour to $SSYS_k$ leads to a strongly connected net N_k*

and optimal regarding the two criteria:

1. *minimize* $CNC_{Avg} = \frac{1}{N}\sum_{k=1}^{N} CNC(N_k)$

2. *minimize* N

A solution to this problem is directly a modelling solution as distributed identified nets, and a partition of the I/Os is deduced from the subsystems by the mapping function, solving the initial problem. This new point of view is resumed in Figure 5.10.

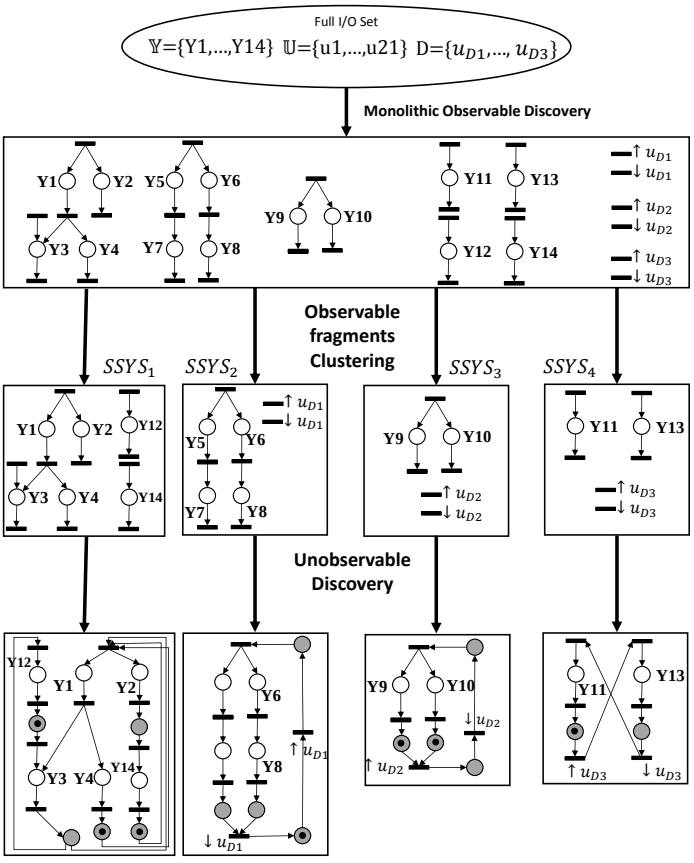


Figure 5.10: Overview of the distributed approach, partitioning taking place after the construction of the observable model

It is not a Bin packing problem ([Korf, 2002]), as the subsystems have no fixed size. This problem can however be viewed as an Exact Set Cover problem ([Knuth, 2000]). Numerous subsystems can be grown out of the blocks B_i. Then, from all the candidate subsystems, an exact cover of PAR, optimal regarding the criteria, can be computed, using for instance Knuth's algorithm ([Knuth, 2000]). However, the number of subsystems that satisfy the constraints is very high (possibly $2^N - 1$), and the exact set cover problem is NP-complete, adding therefore another layer of exponential complexity.

Multiple solutions are satisfying this multi-objective optimization problem, and optimal solutions form a Pareto Frontier regarding the average structural complexity and

the number of subsystems. The extreme solutions are respectively the monolithic model ($SSYS_1 = \{B_1, \ldots, B_m\}$), and the initial set of blocks ($\forall i, SSYS_i = \{B_i\}$).

In the following section, an efficient clustering method is proposed to find 'natural' partitions, and provide a hierarchy of these partitions. A discussion is conducted on the balance between size of the subsystems and the computation time; different variations of the clustering method are proposed depending on the objective of the engineer.

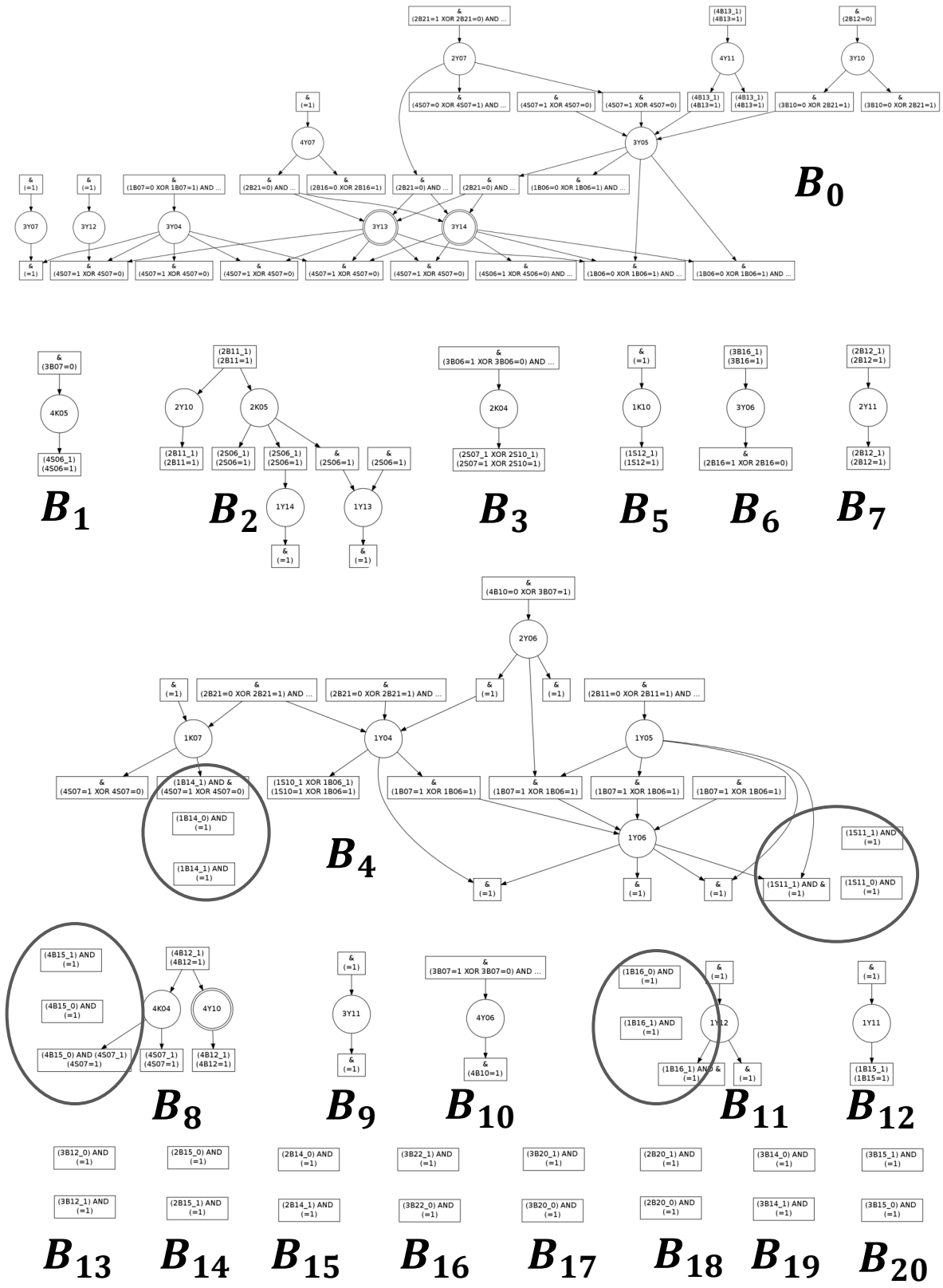


Figure 5.11: The 21 blocks computed for the MSS

5.2 Partitioning by agglomerative hierarchical clustering

The approach proposed is inspired from hierarchical clustering methods used in data mining ([Rokach, 2010]). The objective of clustering is to group objects such that objects in a same group (called a cluster) are more similar than objects belonging to different clusters. The similarity is evaluated through an appropriate metric, corresponding to a measure of the 'distance'between a pair of objects.

Hierarchical clustering aims not only at grouping objects into clusters, but also at providing a hierarchy: a cluster gathers all clusters below it in the hierrachy. Agglomerative clustering is a bottom-up methodology: each object starts in its own cluster, and pairs of clusters are merged while moving up in the hierarchy. The dual top-down approach is called divisive clustering.

In our problem, the objects are subsystems $SSYS_1, \ldots, SSYS_N$. Initially, each subsystem is composed of only one block ($SSYS_i = \{B_i\}$). An agglomerative clustering approach is natural to group the blocks, lower the number of subsystems and satisfy the second objective function. To balance with the first objective function, simplicity should be implied in the similarity metric, so that blocks leading to the most simple models are regrouped.

5.2.1 Similarity and affinity of subsystems

First, the notion of similarity between subsystems is introduced:

Definition 5.7. *Let $SSYS_i, SSYS_j$ be two subsystems. Let N_{ij} be the complete IPN identified after the addition of unobservable behaviour. The similarity Sim of the two subsystems is:*

$$Sim(SSYS_i, SSYS_j) = \begin{cases} CNC(N_{ij}) \text{ if } N_{ij} \text{ is strongly connected} \\ \emptyset \text{ if } N_{ij} \text{ is not strongly connected} \end{cases}$$

When defined, similarity is an indicator of the closeness of subsystems; a low value corresponds to a pair of subsystems whose assembled model is simple to read, hence implying simple operations. The subsystem resulting of the merging of the original pair satisfies the second constraint of the optimization problem (strong connexity).

In the other case, the similarity is undefined ($\emptyset$), as the model resulting of the merging is not strongly connected. Notice that this similarity factor is therefore not a distance. For instance, given subsystems A,B,C, Sim(A,B) and Sim(B,C) being defined, Sim(A,C) might be undefined, unsatisfying the triangular inequality. However, Sim(A∪B,C) is likely to be defined; adequate subsystems might include highly dissimilar subsystems (A,C) who are both similar to a third one (B).

Whenever two systems are similar, they could be merged, and the resulting net would satisfy the constraints. However, to fulfill the first objective function (low average structural complexity), the idea is to merge only the subsystems who are the most

similar, such that structural complexity is minimized at each merging. The affinity of a subsystem is defined as the subsystems it is the most similar to:

Definition 5.8. *Let $SSYS_1, \ldots, SSYS_m$ be m subsystems. The affinity Aff of a subsystem $SSYS_i$ is the set:*

$$Aff(SSYS_i) = \{SSYS_j | Sim(SSYS_i, SSYS_j) = \min_k(Sim(SSYS_i, SSYS_k))\}$$

The affinity of a subsystem might be the empty set, a singleton, or composed of multiple subsystems. An affinity graph is derived from this definition:

Definition 5.9. *Let $SSYS_1, \ldots, SSYS_m$ be m subsystems. The affinity graph A=(V,E) is a directed graph, where the m vertices V represent the m subsystems and the edges represent the affinity, i.e.*

$$(N_i, N_j) \in E \Leftrightarrow SSYS_j \in Aff(SSYS_i)$$

Example 5.4. *Consider 4 subsystems $\{1, 2, 3, 4\}$ such that $Sim(1, 2) = Sim(1, 3) = Sim(1, 4) = \emptyset$, $Sim(2, 3) = Sim(2, 4) = 1.25$ and $Sim(3, 4) = 1.15$. The corresponding affinity graph is presented in Figure 5.12: 1 is an isolated node, 2 has two successors, and finally 3 and 4 are eachothers unique respective affinity, forming a 2-cycle. 2 and 3 should be merged, and the similarity between the resulting subsystem 2,3 and 1 computed, to decide whether further merging is acceptable.*

Sim	1	2	3	4
1	-	∅	∅	∅
2	∅	-	1,25	1,25
3	∅	1,25	-	1,15
4	∅	1,25	1,15	-

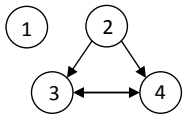


Figure 5.12: Similarity table and Affinity graph deduced for Example 5.4

Subsystems to be merged in priority are the length-2 directed cycles in the affinity net: they involve two subsystems such that each subsystem is the most similar to the other. The subsystems can be iteratively merged, the similarity recomputed at each step, until no more merging is possible. The convergence is ensured due to helpful properties of affinity graphs:

Proposition 5.3. *The maximal length of chordless[1] directed cycles in an affinity graph A is 2*

Proof. Suppose there exists a chordless directed cycle of length $n \geq 3$, let $V_1, V_2, \ldots, V_k$ be the concerned nodes, such that $V_{i-1} \rightarrow V_i \rightarrow V_{i+1}$ and $V_k \rightarrow V_1$. Since the edge

[1] A chordless cycle in a graph is a cycle such that no two vertices of the cycle are connected by an edge that does not itself belong to the cycle.

starting from V_i is oriented towards V_{i+1}, necessarily

$$Sim(SSYS_i, SSYS_{i+1}) < Sim(SSYS_i, SSYS_{i-1})$$

otherwise, V_i would point to V_{i-1}. By repeating for all V_i, it comes

$$Sim(SSYS_k, SSYS_1) < Sim(SSYS_{k-1}, SSYS_k) < \cdots < Sim(SSYS_1, SSYS_2)$$

However, this would mean that the edge $V_1 \rightarrow V_2$ should not exist, as the affinity of $SSYS_1$ can not be $SSYS_2$, hence a contradiction. Necessarily, there can not exist a chordless cycle of length longer than 2. □

Proposition 5.4. *There exists a length-2 directed cycle in any strongly connected component of A not reduced to a node.*

Proof. If said strongly connected component consists in two nodes, it is directly a length-2 directed cycle. If it consists in three nodes or more, since the component is strongly connected, each node possesses a path to reach itself back. Let k be the minimal length of these paths over all nodes. Necessarily there exists a chordless cycle of length k (if there was a chord, k would be lower). According to the previous Proposition, k=2. □

Proposition 5.5. *Let $A = (V, E)$ be an affinity graph. There exists a length-2 directed cycle in A, iff E is not empty*

Proof. The implication $\Rightarrow$ is obvious.

If E is not empty, with the previous Proposition, it remains to prove that there exists a strongly connected component composed of at least two nodes in the graph. Suppose there is not, then there exists a node N_i with at least one entering edge, and no leaving edge. However, the entering edge implies that the set $\{SSYS_j | Sim(SSYS_i, SSYS_j) < \infty\}$ is not empty. Necessarily, the affinity of $SSYS_i$ is not the empty set as well, and N_i must have at least one leaving edge, hence the contradiction. □

Cycles can be found as long as there are edges in the affinity graph. The full agglomerative procedure is exposed by Algorithm 5.2. At each step, the affinity graph is studied; a length-2 directed cycle is picked in each strongly connected component (at least two nodes) of the graph. The nodes of the cycles are merged and the affinity graph recomputed. The procedure is repeated until there are no more edges in the graph. In the worst case, convergence is achieved when there remains exactly one node, which corresponds to the full system.

The costliest operation is the computation of the affinity graph (lines 2,7), which requires the evaluation of all similarity values. An upper bound of the number of similarity values to compute during the discovery is given by the following proposition:

Proposition 5.6. *Consider a system with n subsystems. To run Algorithm 5.2, the maximal number of similarity values to compute is $(n-1)^2$.*

Algorithm 5.2 Agglomerative clustering of subsystems

Require: Blocks $B_1, \ldots, B_n$

Ensure: $PAR = \{SSYS_1, \ldots, SSYS_m\}$ a partition.

1: Compute the initial partition $PAR = \{\{B_1\}, \ldots \{B_n\}\}$
2: Compute the affinity graph $A = (V, E)$ related to PAR
3: **while** $E \neq \emptyset$ **do**
4: Pick a length-2 directed cycle $(SSYS_i, SSYS_j)$ in each strongly connected component of A
5: Merge each pair of subsystems into a new one $SSYS_{ij}$
6: Update the partition PAR
7: Update the affinity graph
8: **end while**

Proof. Given the n initial subsystems, there are initially $n(n-1)/2$ similarity values to compute to build the first affinity graph. Then, the worst case is the following: at each step, only two nodes of the graph are merged. After the first loop, $n-2$ subsystems are unchanged, and one is new, hence $n-2$ new similarity values to compute. After the second loop, it remains $n-3$ subsystems are unchanged, hence $n-3$ new values, etc. The total number is therefore:

$$\frac{n(n-1)}{2} + (n-2) + (n-3) + \cdots + 1 = \frac{n(n-1)}{2} + \frac{(n-1)(n-2)}{2} = (n-1)^2$$

□

Example 5.5 (Example 5.4 cont.)**.** *The agglomerative clustering is illustrated by Figure 5.13. From the similarity table of Figure 5.12, 3-4 is the only strongly connected component. The nodes are merged, and the similarity recomputed. Sim(1,2) is already known, Sim(1,3∪4)=∅, and Sim(2,3∪4)=1.3. 2-3∪4 is a new strongly connected component, and merged. Finally Sim(1,2∪3∪4)=∅, and there is no more edge in the affinity graph, stopping the clustering. Figure 5.13(b) shows a hierarchical representation. Each layer is a solution of the problem, and each layer exhibits a different number of subsystems. The middle layer (1,2,3-4) has three subsystems, whereas the top one (1,2-3-4) has only two.*

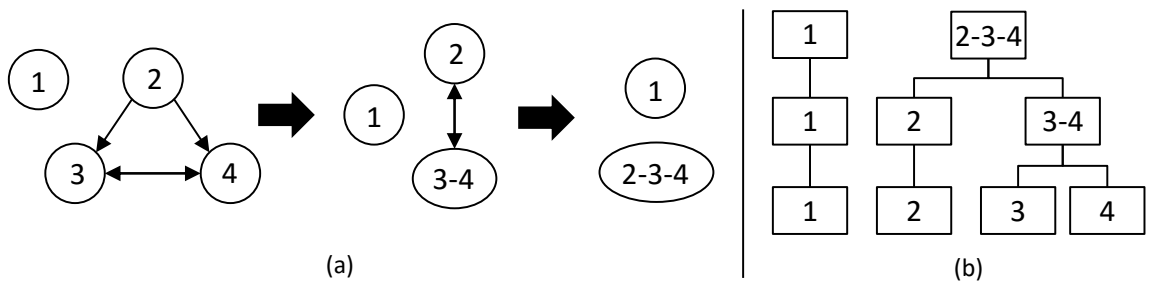


Figure 5.13: (a) Evolution of the affinity graph along the clustering; (b) Hierarchical representation

The main advantage of the approach is to compute a full hierarchy. Suppose that the expert decides a solution with N' subsystems is not distributed enough: it suffices to go down in the hierarchy to find an already computed solution with more subsystems. Reversely, if there are too many subsystems, it suffices to go up in the hierarchy to find a coarser solution.

However, the computation of the full hierarchy is worrisome. First, if there exists a monolithic, strongly connected model of the full system, then Algorithm 5.2 does not stop until said model is reached, whereas the whole point of the distributed approach is to avoid computing the monolithic model.

Furthermore, the update of the affinity graph (line 7 of Algorithm 5.2) implies to compute similarity values. The only way of deciding that a similarity value is undefined is by running the full exploration, and concluding at the end that no strongly connected model exists (Section 4.4.3, a backup threshold was even set to ensure the halting of the computation). Likewise, the computation of similarities might be expensive: consider B_0 and B_4 from Figure 5.11, who respectively have 28 and 22 transitions. Computing $Sim(B_0, B_4)$ took 4 hours. To ensure the efficiency of the clustering, a limitation must be introduced.

5.2.2 Limited clustering

The principle of limited clustering consists in slightly altering the definition of similarity, to ease its calculation. Additional rules are introduced to decide quickly if the similarity is worth computing, or if it is undefined ($\emptyset$). By increasing the number of undefined similarities, the number of non-empty affinities drops, and the lack of edges in the affinity graph (stopping criterion of Algorithm 5.2) is reached after less computations, avoiding convergence to the monolithic model. Two alternate definitions of similarity are therefore proposed, based on the introduction of thresholds; the remainder of the clustering approach remains the same. The two propositions are first presented in this section, then their application is evaluated in the next one.

5.2.2.1 By size of subsystems: $|T|$-clustering

A first proposition consists in limiting the number of transitions in a given model, as the complexity of discovering the unobservable behaviour is exponential regarding the number of transitions. The number of transitions of the full system is known and fixed. The initial subsystems have been designed in order that each transition belongs to exactly one subsystem. If the sum of transitions of two subsystems is over a given threshold, they are then considered dissimilar.

Definition 5.10 ($|T|$-similarity)**.** *Let $SSYS_i, SSYS_j$ be two subsystems, having respectively $|T_i|$ and $|T_j|$ transitions, and $N_{ij} = (P_{ij}, T_{ij}, W_{ij})$ be the complete IPN identified after the addition of unobservable behaviour ($|T_{ij}| = |T_i| + |T_j|$) . Let $|T|_{Lim}$ be a maximal*

number of transitions allowed. The $|T|$-similarity $Sim_{|T|}$ of the two subsystems is:

$$Sim_{|T|}(SSYS_i, SSYS_j) = \begin{cases} CNC(N_{ij}) & \text{if } N_{ij} \text{ is strongly connected} \\ \emptyset & \text{if } N_{ij} \text{ is not strongly connected} \\ & \text{or } |T_{ij}| > |T|_{Lim} \end{cases}$$

Since the total number of transitions $|T|$ is known and fixed, this threshold can also be used to control the number of subsystems of the final partition. If the engineer aims for a solution with N subsystems, by assuming the subsystems have similar sizes, a threshold can be set at $|T|_{Lim} = \lfloor |T|/N \rfloor$.

Without any *apriori*, given that the initial number of subsystems n is known, the engineer can aim for $n/2$ subsystems, *i.e.* set the threshold to $|T|_{Lim} = \lfloor 2|T|/n \rfloor$. If he/she desires more subsystems, he/she can pick a solution already computed. If he/she desires less subsystems, he/she can then aim for $n/4$ subsystems ($|T|_{Lim} = \lfloor 4|T|/n \rfloor$), continue the computation from the last solution, and repeat the procedure until a satisfying solution is reached.

The main advantage of this approach is that the number of subsystems can be controlled. However, some blocks contain too many transitions (see B_0 or B_4 of the MSS), and their similarity values remain undefined until the threshold is high, missing potentially simple models. Furthermore, the computation time is not controlled, and some similarity values can be unpredictabily expensive to compute, despite a reasonable number of transitions.

5.2.2.2 By computation time: time-clustering

To control computation time, it is natural to introduce a computation threshold for the evaluation of each similarity.

Definition 5.11 (time-similarity)**.** *Let $SSYS_i, SSYS_j$ be two subsystems, and N_{ij} be the complete IPN identified after the addition of unobservable behaviour. Let t_{Lim} be the maximal computation time allowed. The time-similarity Sim_t of the two subsystems is:*

$$Sim_t(SSYS_i, SSYS_j) = \begin{cases} CNC(N_{ij}) & \text{if } N_{ij} \text{ is s.c. and computed in less than } t_{lim} \\ \emptyset & \text{if } N_{ij} \text{ is not strongly connected} \\ & \text{or } N_{ij} \text{ is not computed in less than } t_{lim} \end{cases}$$

Setting the threshold t_{lim} limits the computation of any similarity value. t_{lim} can therefore be chosen, based on an upper bound of the number of similarity values to compute (Proposition 5.6). The engineer can allow a total computation time t; given n initial subsystems, the threshold can be set at $t_{lim} = t/(n-1)^2$. This is a lower threshold, ensuring that the total computation time does not exceed t; the actual value of the computation time should actually be far lower.

The main advantage of this approach is that a solution is guaranteed to be obtained

quickly, as computation time is controlled. Furthermore, all the subsystems are sollicited and can be merged, compared to the $|T|$-threshold. However, there is no explicit link between the computation time and the number of subsystems reached when the algorithm terminates. If the number of subsystems is too high, the algorithm can be restarted by increasing the threshold, but it is impossible to determine which increase is required.

5.2.2.3 Choosing a limited clustering approach

The approaches are complementary, as the advantages of one are the drawbacks of the other. In both cases, if the granularity of the system is too coarse (too few subsystems), previous, finer solutions have already been calculated and are available in the hierarchy.

They could be used depending on the objective of the engineer:

- If he/she has an *apriori* on the system, regarding the number of subsystems to discover or their similar sizes, $|T|$-based clustering seems appropriate, despite computation time. The granularity of the system can be finely tuned.
- If he/she wants to obtain quickly a model to get quick insight, and not update it afterwards, time-based clustering seems better.

Both procedures are illustrated on the MSS in the next section.

5.2.3 Results and interpretation

As a recall, the MSS consists in 73 I/Os (43 inputs and 30 outputs), and the observed data consisted in a sequence w of length 63797 I/O vectors. After the computation and simplification of the observable behaviour, the observable model consists in a net of 30 observable places and 101 transitions. The input of the partitioning are these observable fragments, which are regrouped in 21 blocks (B_0 to B_{20}, shown in Figure 5.11). All computations are ran on a laptop (Intel® Core™ i5-3380M CPU @ 2.90GHz x4, 8Go RAM).

5.2.3.1 $|T|$-clustering

The initial number of subsystems being $n = 21$, and the total number of transitions being $|T| = 101$, a first threshold is set at $|T|_{Lim} = \lfloor 2 \times 101/21 \rfloor = 9$. Notice that B_0, B_4 and B_2 have respectively 28,22 and 8 transitions; they can not be merged with any other block without exceeding the threshold. The evolution of the affinity graph throughout the clustering is shown in Figure 5.14.

In the first graph, B_0, B_4 and B_2 are isolated, and there are 4 fragments who exhibit 6 strongly connected components. The length-2 cycles chosen for merging are circled by dotted lines. Notice that the affinities change during the procedure. For instance, the affinity of subsystem B_8 switched from $\{B_1, B_{10}\}$ to B_{17} in Step 2, then to $\emptyset$ in Step 3.

The algorithm stops at the fourth step, where a solution with 9 subsystems is proposed. The computation time was 25 minutes.

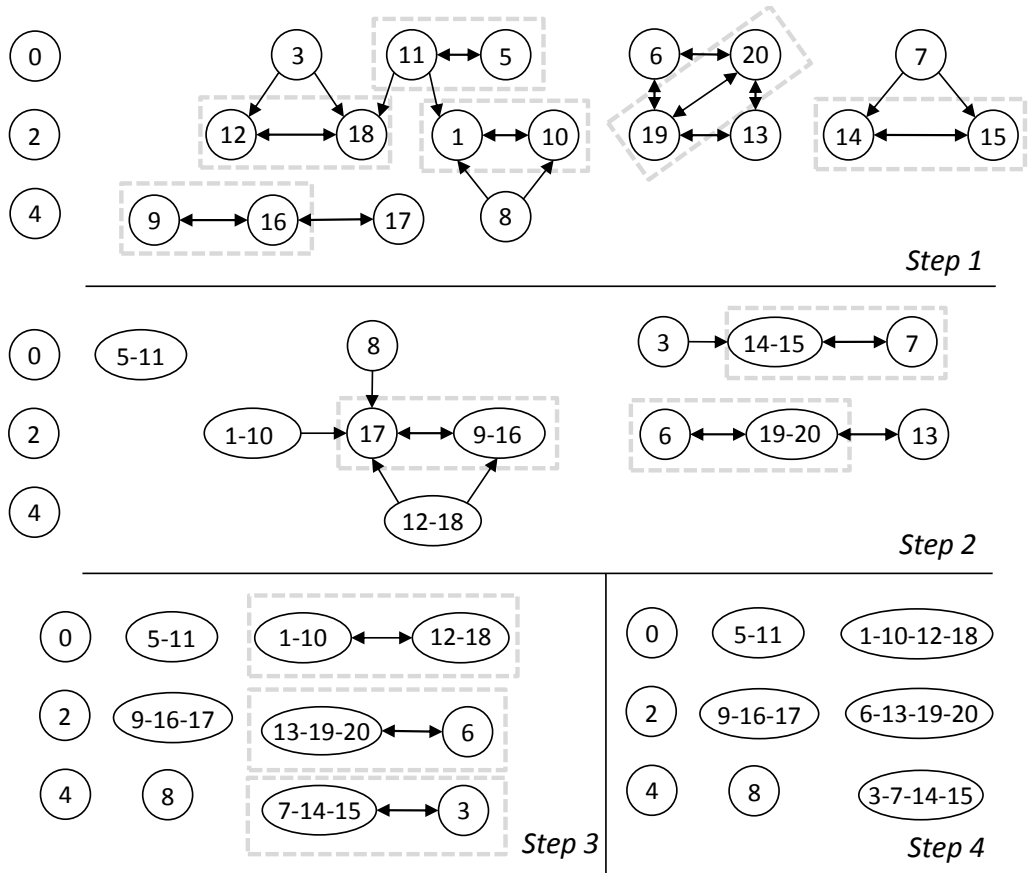


Figure 5.14: Successive affinity graphs, first run of the clustering, with $|T|_{Lim} = 9$

Compared to the solution based on an expert partitioning, this one has fewer subsystems (9 against 11), similar average network complexity (1.35 in both cases), and all models are strongly connected (2 are not in the expert partitioning). This solution is therefore better in every aspect than the expert one. The clusters are consistent with the physical subsystems of the chain; for insance, cluster $\{B_6, B_{13}, B_{19}, B_{20}\}$ corresponds exactly to the input press.

Suppose now that the engineers wants less subsystems. The threshold is doubled, becoming $|T|_{Lim} = 18$. B_0 and B_4 can still not grow. Starting from the last affinity graph of the first run (Step 4), the clustering is ran a second time, and the successive graphs are shown in Figure 5.15. Three steps are required to reach a partition in 6 subsystems, and took 50 additional minutes of computation. The structural complexity naturally degraded to $CNC_{Avg} = 1.50$.

The interpretation of these subsystems on the MSS is shown in Figure 5.16. The six subsystems computed after the second run are represented, and the dotted lines show where to split these subsystems to obtain the 9-subsystems solution computed after the first run. For instance, $SSYS_2$ is split by the blue dotted line into $\{B_9, B_{16}, B_{17}\}$ and B_8, which were two subsystems of the first solution. $SSYS_5$ has components in Stations

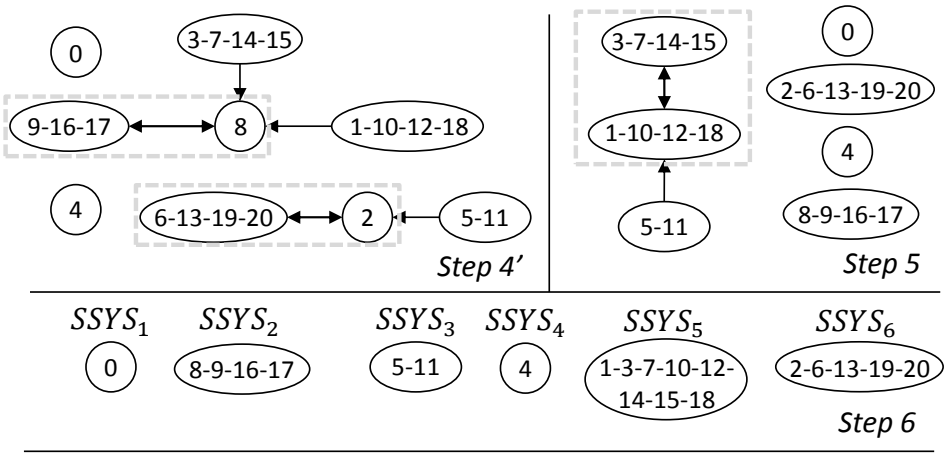


Figure 5.15: Successive affinity graphs, second run of the clustering, with $|T|_{Lim} = 18$

1,2, and 4, which makes it a surprising subsystem. It is likely that the part of Station 4 (B_1, B_{10}) should be connected to either $SSYS_2$ or $SSYS_1$, but these systems could not grow due to the limited size. $|T|$-clustering might not be adapted here, as the initial blocks have irregular sizes.

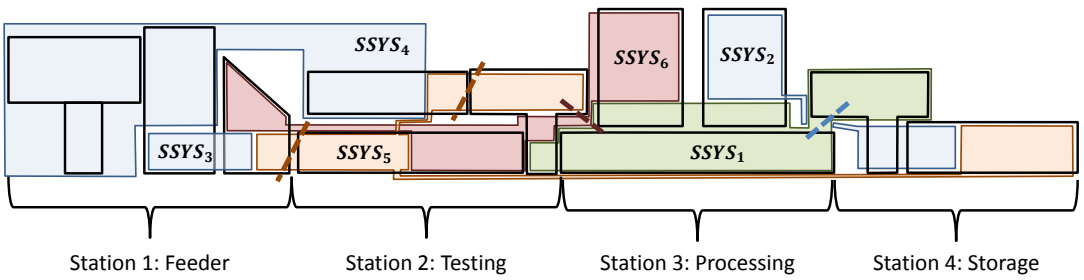


Figure 5.16: Location of the six computed subsystems on the MSS, $|T|_{Lim} = 18$

5.2.3.2 time-clustering

To set the threshold, we considered maximum 2 hours of computation. Since n=21, there are at most 400 similarity values to compute, thus resulting in an upper bound of 18s for t_{lim}, rounded at 20s. In practice, the procedure took 53 minutes to compute. It required 7 steps to reach an affinity graph with no edges, the resulting graphs are presented in Figure 5.18. In the first affinity graph, all blocks are connected, unlike $|T|$-clustering; B_0 and B_2, despite their sizes, end up in $SSYS_1$, meaning that this subsystem can be quickly identified (in less than 20s), even though it contains 40 transitions. Such a subsystem can not be discovered by $|T|$-clustering.

The resulting partitioning consists in 6 subsystems, and $CNC_{Avg} = 1.383$, which makes it slightly more complex than the expert partition, but for half the number of subsystems, and with only strongly connected nets. The interpretation on the MSS in shown in Figure 5.17. The blocks who are physically close to each other are noticeably regrouped into the same subsystem. Compared to the decomposition of Figure 5.16,

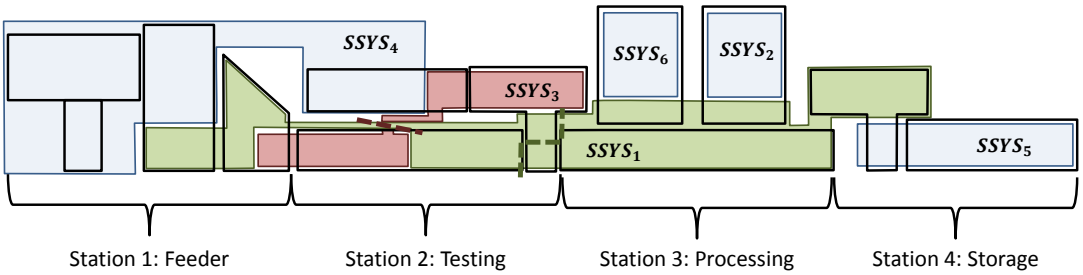


Figure 5.17: Location of the six computed subsystems on the MSS, $t_{Lim} = 20s$

this partition seems even more natural, as the subsystems are not limited in size (and the initial blocks have irregular sizes).

$SSYS_1$ is specially notable, as it implies outputs from all stations. It represents the succession of operations a gear always undergoes when it starts to be treated in station 1, up to its grabbing in section 4. The two presses ($SSYS_2$ and $SSYS_6$) are not always used (depending on the material), and are therefore independent subsystems (expressing the recipes imply lots of choices, which increase the structural complexity when represented). The storage unit ($SSYS_5$) depends likewise on the material. $SSYS_4$ is a subsystem who aliments the chain with gears, and can naturally be separated from the main process. It overlaps the testing station, due to a testing cylinder being often operated simultaneously with a magnet of the feeder. This is probably a spurious correlation, but occuring too frequently to be removed. Finally, $SSYS_3$ is an assembly of two smaller subsystems: a part of station 1, and the gripper of station 2. These are subsystems who are often idle, waiting for a chariot to be available; they can be considered as satellites at the service of the main process. If the procedure had been stopped at step 6 (Figure 5.18), subsystems 3 and 1 would be split in two following the dotted line.

As example, the IPN models of the solution obtained by time-clustering are given in Figures 5.20 and 5.21. When looking at $SSYS_3$, it can be noted that the IPN contains the two fragments that were not connected in Figure 4.22. Namely, the pre-and post transitions of 2K04 are connected to unobservable places P67, P70 and P84, which are all of degree 4. Computing degree 4 is hard for the monolithic model, but not for the reduced subsystems.

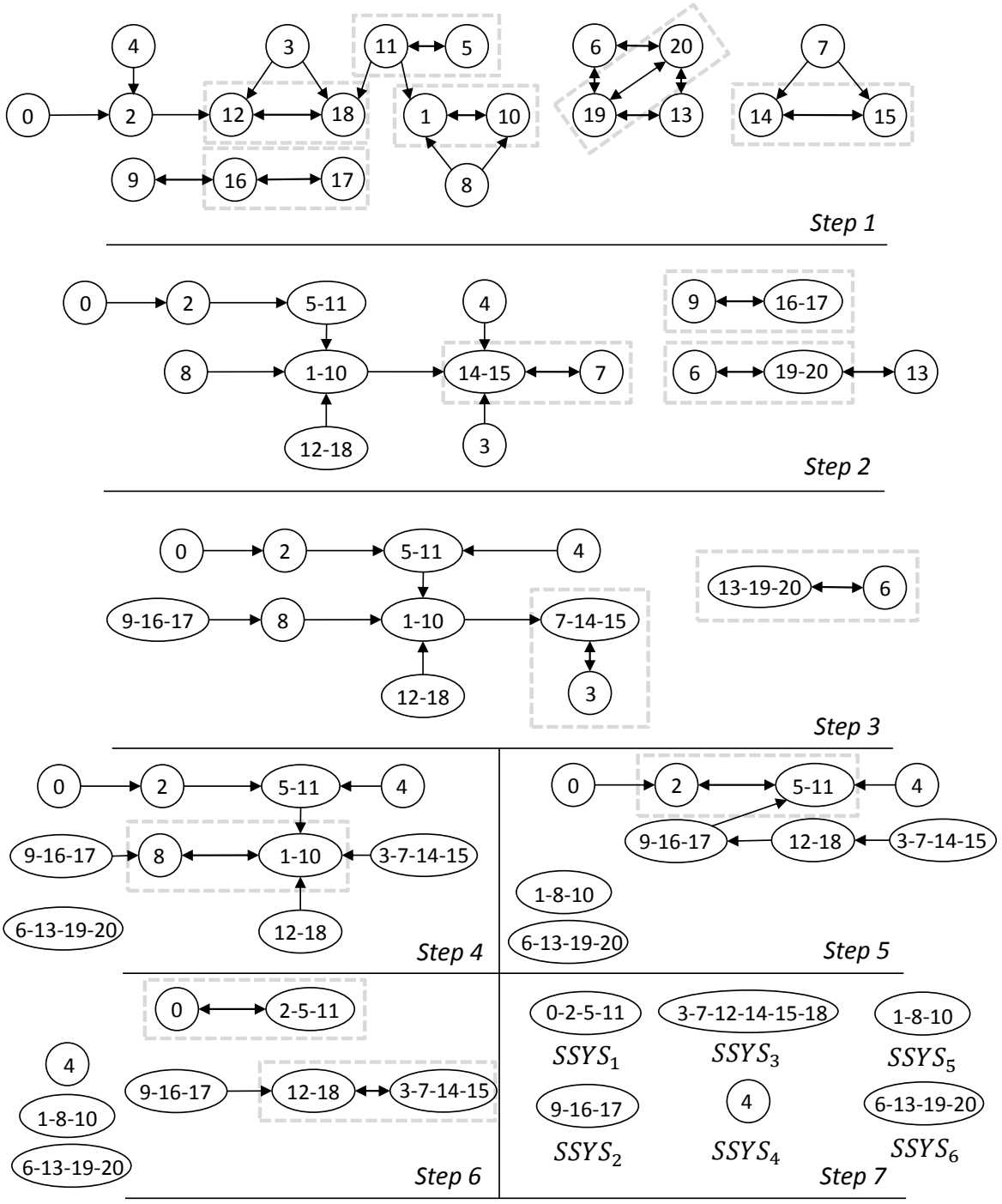


Figure 5.18: Succesive affinity graphs computed for the MSS, $t_{Lim} = 20s$

By varying the time threshold, the granularity of the partition can be controlled. An augmentation of the time threshold is correlated with moving from the initial distributed solution towards the monolithic one, as shown in Figure 5.19. This graph plots different partitioning solutions regarding the number of subsystems and the average structural complexity. As reference solutions, the monolithic model (computed in Chapter 4), the initial block distribution (21 subsystems), and the expert partition (Figure 3.2, 11 subsystems) are plotted. Multiple solutions obtained by time-clustering are given as well, by varying the threshold. Notice that there is no strict monotony, as the solution computed for $t_{lim} = 10s$ is strictly worse than $t_{lim} = 20s$, despite being computed quicker.

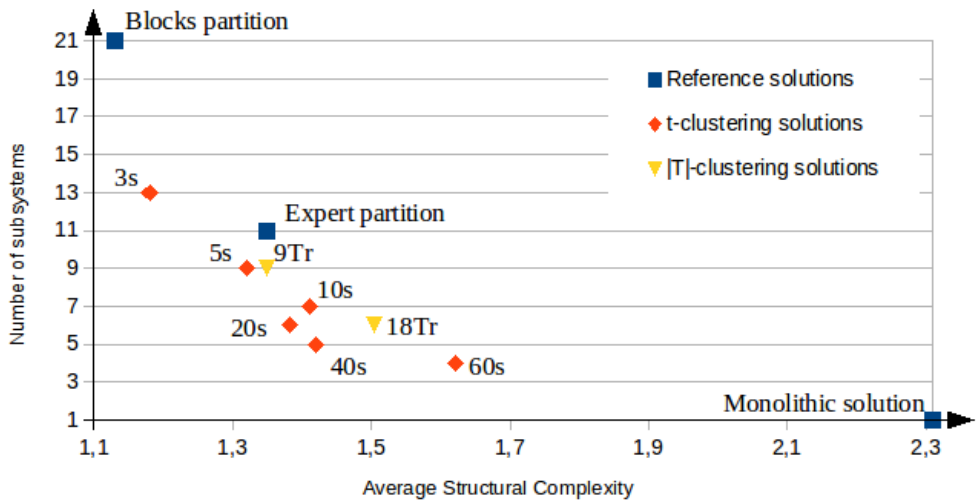


Figure 5.19: Evaluation of different partitions computed with the clustering approach

5.3 Conclusion

This chapter proposed a distributed approach to avoid computing a monolithic model, who is neither easy to understand, nor costless to compute. The problem has been set first as an optimization problem to find an adequate cover of the I/Os, such that the resulting distributed models are simple to understand. Using the observable fragments, which can be computed with the improvements of Chapter 3, the problem has been reformulated as a new optimization problem of clustering the fragments, with the same objective; the I/O cover ensues from the partition. An algorithm inspired from clustering methods is proposed to agglomerate the fragments into clusters. To avoid computation issues, it is proposed to limit the growth of the systems either by size or computation time, leading in reasonable time to simple, distributed models. Due to the lack of a single objective criterion to assess the quality of the model (such as exceeding language for fault diagnosis), the choice of the method is left up to the engineer, who can also control the granularity of the distribution at his taste.

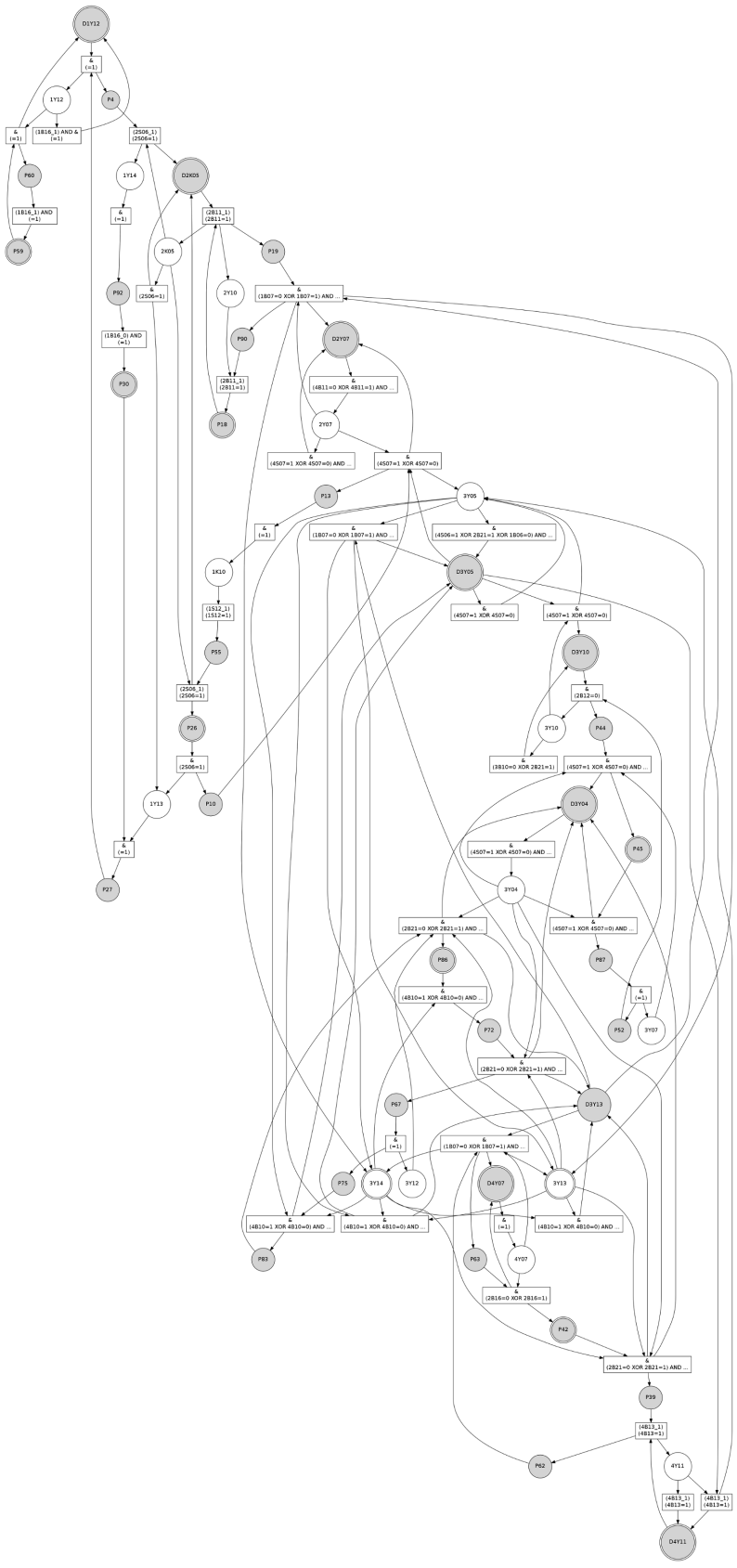


Figure 5.20: IPN identified for SYS_1, biggest subsystem of the MSS

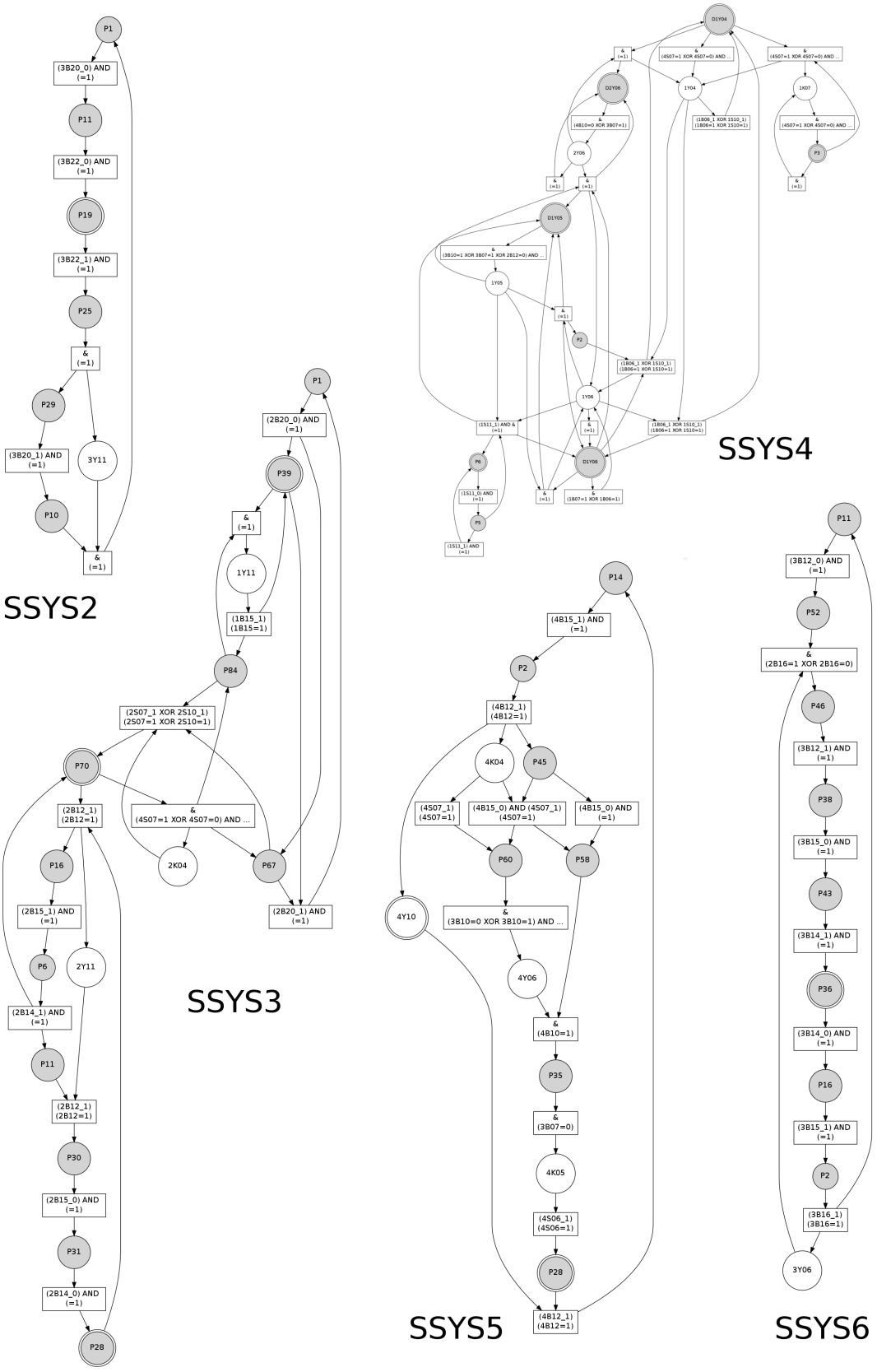


Figure 5.21: IPNs identified for the remaining subsystems of the MSS

Conclusion and outlooks

Discrete Event Systems identification is a young research field; this thesis contributes to its radiance, with the objective of making identification an experimental, performing modelling method. A first challenge is the gap between technology and abstract models: for instance, defining events from input/output signals or including the technological specificities of the controller in the models. Another challenge is scalability; realistic systems are composed of dozens or hundreds of I/Os, and often exhibit massive concurrency, leading to a state-space explosion, common issue when modelling DES.

An approach developed in a previous thesis ([Estrada-Vargas, 2013]) was proposed to build Interpreted Petri Nets from a sequence of I/O vectors observed during the operation of a real system. The approach is devoted to closed-loop systems controlled by a PLC, taking up the first challenge. The resulting model is compact, explicits the reactive behaviour of the controller through the direct I/O causalities, and agregates the unobservable state evolutions. It satisfies a reverse-engineering objective. However, the second challenge is still troublesome for this method. Massive concurrency namely implies the impossibility of achieving a complete observation, hindering the unobservable discovery.

This thesis tackles the scalability challenge by proposing three contributions, to improve the previous inquiries.

Chapter 3 proposed to include another technological specificity of the controller in the identification procedure, to improve the computation of the observable behaviour. Synchronization has been identified as critical when dealing with massively concurrent systems, provoking spurious simultaneous observations of input and output events. A spurious-concurrency-blocking filter is proposed to separate these observations from actual I/O causalities. A reduction procedure then detects spurious unfrequent transitions, removes them from the net and replaces them in the firing sequence by an equivalent set of firings. This contribution greatly improves the construction of the observable model, making it compact and easily computable despite the concurrency.

Chapter 4 proposed a new approach to discover the unobservable behaviour, based on a single theorem. Its strength lies in its genericity, ensuring a net always fitting the observation, and its ability to easily discover concurrency, despite the uncomplete observation. However, its weakness is the exponential size of the search space, which is mitigated in practice by an adapted search heuristic. This contribtion enables the construction of complete, compact, explicit, monolithic IPN models for systems of realistic sizes. These models might nevertheless remain hard to read due to their sizes.

Chapter 5 finally proposed a distributed approach to avoid computing a monolithic model. Using an expert partitioning of the system is possible, but in the extreme case of a blackbox approach, no partitioning is provided. The task of finding an adequate partitioning is formulated as a multi-objective optimization problem: the resulting distributed models offer a compromise between simplicity and size. The filtered observable behaviour is exploited to provide an initial solution, and a clustering algorithm is proposed to merge the subsystems and find good partitionings, variations of the algorithm being proposed to the engineer depending on its objective (quick model, or controlled number of subsystems).

The proposed partitioning approach is promising, since it uses the knowledge obtained from the construction of the observable behaviour. For now, the clustering approach requires the computation of numerous models; it might be interesting to propose criteria or objective functions computable from the I/O sequence or the firing sequence instead, to ease the computation. Another direction could be explored in a greybox approach, if an expert partitioning is available: said partitioning could be corrected and improved by the knowledge brought by the observable fragments. Finally, the objective functions were chosen accordingly to the reverse engineering purpose of the net, and can be changed to fit a different purpose. Namely, to use the identified Petri Net for fault diagnosis (see [Basile, 2014]), the function(s) proposed in [Schneider and Litz, 2014] could be combined with the observable behaviour to obtain distributed nets adapted to fault diagnosis. For such a purpose, a new heuristic might also be required to discover the unobservable part.

The approach proposed provides an engineer with one, or multiple, simple, global, comprehensive model(s) of the system. However, as seen in the development, specific behaviours remain hard to infer. Specially, temporizations exist in the system, and are agregated into unobservable places by the current approach. Hence, their representation in the net is not explicit. Time intervals can be inferred on the fly while building the PN fragments (like in [Basile et al., 2016c]), or added to the full structure after its computation (like in [Schneider et al., 2012]). Said intervals could be associated to places to represent either the duration of activation of an actuator (observable places), or the waiting time in a state of the controller (unobservable places). However, most of the intervals inferred do not represent functional temporizations implemented in the controller, and postprocessing them to extract the true temporizations might be tedious, if not impossible. Instead, a greybox approach should be promising. The knowledge of the components involved in a temporization should help its discovery, then explicit representation in the net.

Likewise, other untimed behaviours such as counters or self-loops, remain hard to infer. Data mining methods could be used to automatically discover patterns symptomatic of these behaviours in the firing sequence, leading to the definition of new relations, with their associated unobservable places. However, most of the relations discovered might

be irrelevant or redundant. A greybox approach should be considered as well to discover these specific behaviours. A global approach can be run blackbox, discovering direct I/O causalities, sequentialities, concurrency and partitioning, then the models can be locally complemented at the will of the engineer by locally injecting knowledge.

Finally, the reflexion should be prolonged on devoting these reverse-engineered nets to certification or reimplementation. To certify a system, exhaustive test might be costly, whereas identification can provide a model whose properties can be verified for a lower cost, given that the model is guaranteed precise. A first global model can be obtained through blackbox passive observation, then locally completed by active identification. Similarly to test approaches, the idea is to force some inputs and observe the response of some outputs. The identified model can be corrected according to these new observations. Missing behaviours, notably implemented for safety purposes, can be observed that way. To avoid any deterioration, active identification should nevertheless be performed only in greybox, or with a simulated operating part. To extend the purpose of the identified nets, active greybox identification seems to be the direct follow-up of passive blackbox identification.

Bibliography

[Aguilar, 2011] Aguilar, J. (2011). The identification of discrete-event dynamic systems based on the evolutionary programming. *International Journal of Knowledge-based and Intelligent Engineering Systems 15*, pages 43–53.

[Angluin, 1982] Angluin, D. (1982). Inference of reversible languages. *Journal of the Association for Computing Machinery, 29(3)*, pages 741–765.

[Angluin, 1987] Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation 75*, pages 87–106.

[Badouel et al., 1995] Badouel, E., Bernardinello, L., and Darondeau, P. (1995). Polynomial algorithms for the synthesis of bounded nets. *Lecture Notes in Computer Science, 915*, pages 647–679.

[Badouel et al., 1997] Badouel, E., Bernardinello, L., and Darondeau, P. (1997). The synthesis problem for elementary net systems is np-complete. *Theoretical Computer Science, 186(12)*, pages 107–134.

[Badouel et al., 2015] Badouel, E., Bernardinello, L., and Darondeau, P. (2015). *Petri Net Synthesis*. Springer-Verlag.

[Basile, 2014] Basile, F. (2014). Overview of fault diagnosis methods based on petri net models. *Proceedings of the IEEE European Control Conference (ECC), Strasbourg, France*, pages 2636–2642.

[Basile et al., 2012] Basile, F., Chiacchio, P., and Coppola, J. (2012). Active identification of petri nets models. *Proceedings of the 11th Workshop on Discrete Event Systems (WODES12), Guadalajara, Mexico*, pages 278–285.

[Basile et al., 2015] Basile, F., Chiacchio, P., and Coppola, J. (2015). Real time identification of time petri net faulty models. *Proceedings of the 2015 IEEE International Conference on Automation Science and Engineering (CASE), Gothenburg, Sweden*, pages 208–285.

[Basile et al., 2016a] Basile, F., Chiacchio, P., and Coppola, J. (2016a). Faulty model identification in deterministic labeled time petri nets. *Proceedings of the 13th Workshop on Discrete Event Systems-WODES 2016, Xi'an, China*, pages 486–492.

[Basile et al., 2016b] Basile, F., Chiacchio, P., and Coppola, J. (2016b). Identification of labeled time petri nets. *Proceedings of the 13th Workshop on Discrete Event Systems-WODES 2016, Xi'an, China*, pages 478–485.

[Basile et al., 2016c] Basile, F., Chiacchio, P., and Coppola, J. (2016c). Identification of time petri nets models. *IEEE Transactions on Systems, Man and Cybernetics: Systems.*

[Basile et al., 2011] Basile, F., Chiacchio, P., Coppola, J., and De Tommasi, G. (2011). Identification of petri nets using timing information. *Proceedings of the 3rd International Workshop on Dependable Control of Discrete Systems (DCDS'11), Saarbrücken, Germany*, pages 156–163.

[Bergenthum et al., 2007] Bergenthum, R., Desel, J., Lorenz, R., and Mauser, S. (2007). Process mining based on regions of languages. *Business Process Management*, pages 375–383.

[Biermann, 1972] Biermann, A. (1972). On the inference of turing machines from sample computations. *Artificial Intelligence 3*, pages 181–198.

[Biermann and Feldman, 1972] Biermann, A. and Feldman, J. (1972). On the sysnthesis of finite-state machines from samples of their behavior. *IEEE Transactions on Computers, Vol. 21*, pages 592–597.

[Cabasino et al., 2014] Cabasino, M., Giua, A., Hadjicostis, C., and Seatzu, C. (2014). Fault model identification and synthesis in petri nets. *Discrete Event Dynamic Systems*, pages 1–22.

[Cabasino et al., 2015] Cabasino, M.-P., Darondeau, P., Fanti, M.-P., and Seatzu, C. (2015). Model identification and synthesis of discrete-event systems. In *Contemporary Issues in System Science and Engineering*. IEEE-Wiley.

[Cabasino et al., 2007] Cabasino, M.-P., Giua, A., and Seatzu, C. (2007). Identification of petri nets from knowledge of their language. *Discrete Event Dynamic Systems, 17(4)*, pages 447–474.

[Carmona et al., 2009] Carmona, J., Cortadella, J., and Kishinevsky, M. (2009). Divide-and-conquer strategies for process mining. *Business Process Management (BPM 2009), LNCS 5701*, pages 327–343.

[Cassandras and Lafortune, 2008] Cassandras, C. and Lafortune, S. (2008). *Introduction to Discrete Event Systems, Second Edition*. Springer.

[Chikofsky and Cross II, 1990] Chikofsky, E.-J. and Cross II, J.-H. (1990). Reverse engineering and design recovery: A taxonomy. *IEEE Software*, pages 13–17.

[David and Alla, 1994] David, R. and Alla, H. (1994). Petri nets for modeling of dynamic systems- a survey. *Automatica Vol.30 No.2*, pages 175–202.

[Desel and Esparza, 1995] Desel, J. and Esparza, J. (1995). *Free Choice Petri Nets.* Cambridge University Press.

[Desel and Reisig, 1996] Desel, J. and Reisig, W. (1996). The synthesis problem of petri nets. *Acta Informatica 33*, pages 297–315.

[Dotoli et al., 2010] Dotoli, M., Fanti, M., Mangini, A., and W.Ukovitch (2010). Identification of the unobservable behaviour of industrial automation systems by petri nets. *Control Engineering Practice, 9(9)*, pages 958–966.

[Dotoli et al., 2008] Dotoli, M., Fanti, M., and Mangini, A. M. (2008). Real time identification of discrete event systems using petri nets. *Automatic, 44(5)*, pages 1209–1219.

[Esparza et al., 2010] Esparza, J., Leucker, M., and Schlund, M. (2010). Learning workflow petri nets. *Applications and Theory of Petri Nets,LNCS 6128*, pages 206–225.

[Estrada-Vargas, 2013] Estrada-Vargas, A. (2013). *Black-box identification of automated discrete event systems.* PhD thesis, ENS Cachan, France.

[Estrada-Vargas et al., 2010] Estrada-Vargas, A.-P., Lesage, J.-J., and Lopez-Mellado, E. (2010). A comparative analysis of recent identification approaches for discrete-event systems. *Mathematical Problems in Engineering, vol. 2010, Article ID 453254, 21 pages.*

[Estrada-Vargas et al., 2014] Estrada-Vargas, A.-P., Lesage, J.-J., and Lopez-Mellado, E. (2014). Input-output identification of controlled discrete manufacturing systems. *International Journal of System Science 45, 3*, pages 456–471.

[Estrada-Vargas et al., 2015] Estrada-Vargas, A.-P., Lesage, J.-J., and Lopez-Mellado, E. (2015). A black-box identification method for automated discrete-event systems. *IEEE Transactions on Automation Science and Engineering*, pages 1–16.

[Fanti and Seatzu, 2008] Fanti, M. and Seatzu, C. (2008). Fault diagnosis and identification of discrete event systems using petri nets. In *Proceedings of the 9th International Workshop on Discrete Event Systems, Göteborg, Sweden*, pages 432–435.

[Garcia-Valles and Colom, 1999] Garcia-Valles, F. and Colom, J.-M. (1999). Implicit places in net systems. In *Proc. of the 8th International Workshop on Petri Nets and Performance Models*, pages 104–113.

[Gill, 1966] Gill, A. (1966). Realization of input-output relations by sequential machines. *Journal of the Association for Computing Machinery,13*, pages 33–42.

[Girault and Valk, 2003] Girault, C. and Valk, R. (2003). *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications*. Springer-Verlag.

[Giua and Seatzu, 2005] Giua, A. and Seatzu, C. (2005). Identification of free-labeled petri nets via integer programming. In *Proc of the 44th IEEE Conference on Decision and Control, and the European Control Conference*, pages 7639–7644.

[Gold, 1967] Gold, E. M. (1967). Language identification in the limit. *Information and Control, 10*, pages 447–474.

[Goldsmith et al., 2008] Goldsmith, J., Hagen, M., and Mundhenk, M. (2008). Complexity of dnf minimization and isomorphism testing for monotone formulas. *Information and Computation 206*, pages 760–775.

[Grinchtein et al., 2005] Grinchtein, O., Jonsson, B., and Leucker, M. (2005). Inference of timed transition systems. *Electronic Notes in Theoretical Computer Science, vol. 138(3)*, page 87–99.

[Heun and Vairavan, 1976] Heun, K. and Vairavan, K. (1976). The realization of consistent input-output sequences by finite state machines. *Information and Control 31*, pages 97–106.

[Hiraishi, 1992] Hiraishi, K. (1992). Construction of a class of safe petri nets by presenting firing sequences. *Proceedings of the 13th International Conference on Application and Theory of Petri Nets, vol. 616 of Lectures Notes in Computer Sciences, Sheffield, UK*, pages 244–262.

[IEC 61131-1..IEC 61131-8, 2010] IEC 61131-1..IEC 61131-8 (2000-2010). *Programmable Controllers Part 1 to 8*. International Electrotechnical Commission.

[Jantzen, 1987] Jantzen, M. (1987). Language theory of petri nets. *Petri Nets: Central Models and Their Properties, Lecture Notes in Computer Science 254*, pages 397–412.

[Jarvis, 2010] Jarvis, D. (2010). An identification technique for timed event systems. *Proceedings of the 10th International Workshop on Discrete Event Systems (WODES2010), Berlin, Germany*, pages 181–186.

[Kella, 1971] Kella, J. (1971). Sequential machine identification. *IEEE Transactions on Computers*, pages 332–338.

[Klein, 2005] Klein, S. (2005). *Identification of Discrete event systems for fault detection purposes*. PhD thesis.

[Knuth, 2000] Knuth, D. (2000). Dancing links. *Millenial Perspectives in Computer Science*, pages 187–214.

[Korf, 2002] Korf, R. (2002). A new algorithm for optimal bin packing. In *Eighteenth national conference on Artificial intelligence*, pages 731–736.

[Ladiges et al., 2015] Ladiges, J., Haubeck, C., Fay, A., and Lamersdorf, W. (2015). Learning behaviour models of discrete event production systems from observing input/output signals. In *15th IFAC/IEEE/IFIP/IFORS Symposium on Information Control Problems in Manufacturing (INCOM)*, volume 48 of *IFAC-PapersOnLine*, page 1565–1572.

[Lassen and van der Aalst, 2009] Lassen, K. and van der Aalst, W. (2009). Complexity metrics for workflow nets. *Information and Software Technology, 51(3)*, pages 610–626.

[Leemans et al., 2013] Leemans, S., D.Fahland, and Van der Aalst, W. (2013). Discovering block-structured process models from event logs. *Application and Theory of Petri Nets and Concurrency, LNCS 7927*, pages 311–329.

[Leemans et al., 2014] Leemans, S., D.Fahland, and Van der Aalst, W. (2014). Discovering block-structured process models from incomplete event logs. *Application and Theory of Petri Nets and Concurrency, LNCS 8489*, pages 91–110.

[Lefebvre and Leclerq, 2011] Lefebvre, D. and Leclerq, E. (2011). Stochastic petri net identification for the fault detection and isolation of discrete event systems. *IEEE Transactions on Systems, Man and Cybernetics–Part A: Systems and Humans, Vol 41, No 2*, pages 213–225.

[Lorenz et al., 2007] Lorenz, R., Mauser, S., and Juhas, G. (2007). How to synthesize nets from languages - a survey. In *2007 Winter Simulation Conference, Washington D.C.*, pages 637–647.

[Maier, 2014] Maier, A. (2014). *Identification of Timed Behavior Models for Diagnosis in Production Systems.* PhD thesis, University of Paderborn, Germany.

[Mazurkiewicz, 1995] Mazurkiewicz, A. (1995). Introduction to trace theory. *The Book of Traces*, pages 3–41.

[McCabe, 1976] McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering (2)*, pages 308–320.

[Mealy, 1955] Mealy, G. (1955). A method for synthesizing sequential circuits. *Bell System Technical Journal*, pages 1045–1079.

[Meda et al., 1998] Meda, M., Ramirez, A., and Malo, A. (1998). Identification in discrete event systems. *IEEE Int. Conf. on Systems Man and Cybernetics, San Diego , USA*, pages 747–745.

[Meda-Campana, 2002] Meda-Campana, M. (2002). *On-line Identification of Discrete Event Systems: Fundamentals and Algorithms for the Synthesis of Petri Net Models.* PhD thesis, Cinvestav Guadalajara, Mexico.

[Meda-Campana and Lopez-Mellado, 2001] Meda-Campana, M. and Lopez-Mellado, E. (2001). A passive method for on-line identification of discrete event systems. *Proceedings of the 40th IEEE Conference on Decision and Control, Orlando, Florida USA*, pages 4990–4995.

[Mendling et al., 2007] Mendling, J., Reijers, H., and Cardoso, J. (2007). What makes process models understandable ? *Lecture Notes in Computer Science Volume 4714*, pages 48–63.

[Moore, 1956] Moore, E. (1956). Gedanken-experiments on sequential machines. *Automata Studies,Annals of Mathematical Studies (Princeton, N.J.: Princeton University Press (34)*, pages 129–153.

[Munoz et al., 2014] Munoz, D., Correcher, A., Garcia, E., and Morant, F. (2014). Identification of stochastic timed discrete event systems with st-ipn. *Mathematical Problems in Engineering, vol. 2014, Article ID 835312, 21 pages.*

[Ould El Medhi et al., 2006] Ould El Medhi, S., Leclerq, E., and Lefebvre, D. (2006). Petri nets design and identification for the diagnosis of discrete event systems. *2006 IAR Annual Meeting, Nancy, France.*

[Pascoe, 1966] Pascoe, T. L. (1966). Allocation of resources – cpm. *Revue Francaise de Recherche Opérationelle 38*, pages 31–38.

[Petri, 1962] Petri, C. (1962). *Kommunication mit Automaten.* PhD thesis.

[Prähofer et al., 2013] Prähofer, H., Schatz, R., and Grimmer, A. (2013). Reverse engineering and visualization of the reactive behavior of plc applications. *11th IEEE International Conference on Industrial Informatics (INDIN), Bochum, Germany*, pages 564–571.

[Prähofer et al., 2014] Prähofer, H., Schatz, R., and Grimmer, A. (2014). Behavioral model synthesis of plc programs from execution traces. *IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, Spain*, pages 1–6.

[Rauterberg, 1992] Rauterberg, M. (1992). A method of a quantitative measurement of cognitive complexity. *Human-Computer Interaction: Tasks and Organisation.*, pages 295–307.

[Rokach, 2010] Rokach, L. (2010). A survey of clustering algorithms. *Data Mining and Knowledge Discovery Handbook*, pages 269–298.

[Roth, 2010] Roth, M. (2010). *Identification and Fault Diagnosis of Industrial Closed-Loop Discrete Event Systems*. PhD thesis.

[Roth et al., 2009a] Roth, M., Lesage, J.-J., and Litz, L. (2009a). Distributed identification of concurrent discrete event systems for fault detection purposes. In *Proceedings of the European Control Conference 2090 (ECC 2009), Budapest, Hungary*, pages 2590–2595.

[Roth et al., 2009b] Roth, M., Lesage, J.-J., and Litz, L. (2009b). A residual inspired approach for fault localization in des. In *Proceedings of the 2nd IFAC Workshop on Dependable Control of Discrete Event Systems(DCDS'09),Bari, Italy.*, pages 347–352.

[Roth et al., 2010a] Roth, M., Lesage, J.-J., and Litz, L. (2010a). Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems. *Proc. of the 2010 American Control Conference (ACC), Baltimore, MD, USA*, pages 2601–2606.

[Roth et al., 2010b] Roth, M., Lesage, J.-J., and Litz, L. (2010b). Identification of discrete event systems - implementation issues and model completeness. In *Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics (ICINCO) Funchal, Portugal.*

[Saives and Faraut, 2014] Saives, J. and Faraut, G. (2014). Automated generation of activities of daily living. In *Proceedings of the 12th International Workshop on Discrete Event Systems-WODES 2014, Paris, France*, pages 13–20.

[Saives et al., 2015a] Saives, J., Faraut, G., and Lesage, J.-J. (2015a). Identification of discrete event systems unobservable behaviour by petri nets using language projections. In *2015 European Control Conference (ECC).*

[Saives et al., 2015b] Saives, J., Pianon, C., and Faraut, G. (2015b). Activity discovery and detection of behavioral deviations of an inhabitant from binary sensors. *IEEE Transactions on Automation Science and Engineering, 12 (4)*, pages 1211–1224.

[Schneider, 2014] Schneider, S. (2014). *Automatic Modeling and Fault Diagnosis of Timed Concurrent Discrete Event Systems*. PhD thesis, University of Kaiserslautern, Germany.

[Schneider and Litz, 2014] Schneider, S. and Litz, L. (2014). Automatic partitioning of des models for distributed fault diagnosis purposes. *Proceedings of the 12th International Workshop on Discrete Event Systems-WODES 2014, Paris, France*, pages 21–26.

[Schneider et al., 2012] Schneider, S., Litz, L., and Lesage, J.-J. (2012). Determination of timed transitions in identified discrete-event models for fault detection. *51st IEEE Conference on Decision and Control, Maui, HI, United States*, pages 5816–5821.

[Shahbaz and Groz, 2009] Shahbaz, M. and Groz, R. (2009). Inferring mealy machines. *Proceedings of the FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, LNCS 5850*, pages 207–222.

[Soo and Jung-Mo, 1992] Soo, L. and Jung-Mo, Y. (1992). An empirical study on the complexity metrics of petri nets. *Microelectron. Reliab., Vol. 32, No. 3*, pages 323–329.

[Tapia-Flores et al., 2014] Tapia-Flores, T., Lopez-Mellado, E., Estrada-Vargas, A.-P., and Lesage, J.-J. (2014). Petri net discovery of discrete event processes by computing t-invariants. In *2014 IEEE 19th Conference on Emerging Technologies and Factory Automation (ETFA).*

[Tarjan, 1972] Tarjan, R. E. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing, vol. 1, no 2*, pages 146–160.

[Van der Aalst et al., 2005] Van der Aalst, W., Alves de Medeiros, A., and Weijters, A. (2005). Genetic process mining: An experimental evaluation. *Applications and Theory of Petri Nets, LNCS 3536*, pages 48–69.

[Van der Aalst, 2011a] Van der Aalst, W.-M.-P. (2011a). Do petri nets provide the right representational bias for process mining? In *Proc. of the Workshop Applications of Region Theory 2011*, pages 85–94.

[Van der Aalst, 2011b] Van der Aalst, W.-M.-P. (2011b). *Process Mining, Discovery, Conformance and Enhancement of Business Processes.* Springer-Verlag Berlin Heidelberg.

[Van der Aalst, 2013a] Van der Aalst, W.-M.-P. (2013a). Discovering petri nets from event logs. *Transactions on Petri Nets and Other Models of Concurrency VII, LNCS 7480*, pages 372–422.

[Van der Aalst, 2013b] Van der Aalst, W.-M.-P. (2013b). A general divide and conquer approach for process mining. *2013 Federated Conference on Computer Science and Information Systems (FedCSIS), Krakow, Poland*, pages 1–10.

[Van der Aalst et al., 2004] Van der Aalst, W.-M.-P., Weijters, A.-J.-M.-M., and Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, pages 1128–1142.

[Van Dongen et al., 2009] Van Dongen, B., Alves de Medeiros, A., and Wen, L. (2009). Process mining: Overview and outlook of petri net discovery algorithms. *ToPNoC II, LNCS 5460*, pages 225–242.

[Van Hee et al., 2011] Van Hee, K. M., Liu, Z., and Sidorova, N. (2011). Is my event log complete? — a probabilistic approach to process mining. *Fifth International Conference on Research Challenges in Information Science (RCIS), Gosier, Guadeloupe*, pages 1–12.

[Veelenturf, 1978] Veelenturf, L. (1978). Inference of sequential machines from sample computations. *IEEE Transactions on Computers, C-27(2)*, pages 167–170.

[Veelenturf, 1981] Veelenturf, L. (1981). An automata theoretical approach to developing learning neural networks. *Cybernetics and Systems, 12*, pages 179–202.

[Wen et al., 2007] Wen, L., Van der Aalst, W.-M.-P., Wang, J., and Sun, J. (2007). Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery 15(2)*, pages 145–180.

Résumé en langue française

Introduction

L'identification est une méthode de modélisation expérimentale, basée sur l' observation des signaux d'un système existant. Pour la classe spécifique des Systèmes à Evénements Discrets, modélisés par un espace d'état discret et une fonction de transition sollicitée sur occurrence d'événements discrets et asynchrones, la modélisation par connaissance est à ce jour la technique la plus répandue. Cette thèse contribue au développement de l'identification et à son affirmation comme alternative viable à la modélisation experte.

Dans une optique de rétro-ingénierie, l'identification est adaptée pour fournir des modèles simples et compréhensibles de systèmes complexes. La classe de modèles retenues pour la modélisation est celle des Réseaux de Petri Interprétés (RdPI). Ces modèles permettent de représenter explicitement, grâce à l'interprétation, à la fois les causalités directes entre entrées et sorties du système, dit comportement *observable*, et les évolutions d'état interne, dit comportement *non observable*.

On s'intéresse ici à des systèmes en boucle fermée, constitués d'un processus manufacturier et d'un contrôleur de type Automate Programmable Industriel (API). L'identification est conduite en boîte-noire, seuls les signaux logiques d'entrées/sorties étant mesurables, et passive, pour éviter toute détérioration du processus observé.

Quelques challenges majeurs sont à relever. Le caractère expérimental de l'identification implique la prise en compte de contraintes technologiques, liées à la nature des signaux d'entrée/sortie ou au fonctionnement du contrôleur. Il est également impossible d'observer l'intégralité des comportements existants du système bouclé en un temps fini ; une contrainte physique supplémentaire est donc l'incomplétude de l'observation. Enfin, la taille des systèmes est un challenge majeur. La taille des espaces d'états décrivant un SED est souvent sujette à l'explosion combinatoire, notamment lorsque de nombreux sous-systèmes évoluent en parallélisme total. Ces phénomènes de parallélisme s'amplifient avec la taille du système, et constituent une limitation à la taille des systèmes pouvant être modélisés ou identifiés.

Cette thèse s'attaque notamment à ce dernier challenge. Elle s'inscrit dans la continuité de travaux proposés dans une thèse antérieure ([Estrada-Vargas, 2013]). Dans ces derniers, une approche en deux temps permet de découvrir successivement les comportements observable, puis non observable, d'un SED, et d'en restituer un modèle RdPI compact et expressif. Cette approche montre ses limites lorsque la taille et la complex-

ité du système augmentent. Les contributions présentées dans ce document visent à repousser ces limites, et sont les suivantes:

- Permettre la construction d'un modèle monolithique en :
  - Filtrant et réduisant les effets du parallélisme dans la construction du comportement observable
  - Proposant une nouvelle approche robuste à la concurrence pour la découverte du comportement non observable
- Proposer une approche distribuée, par un partitionnement automatique du système en sous-systèmes de taille réduite.

Passage à l'échelle du comportement observable

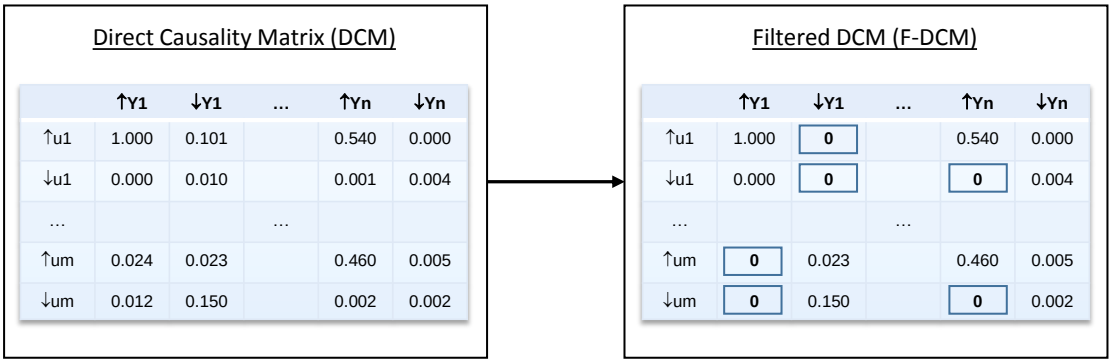
La découverte du comportement observable est fondé sur la construction d'une matrice de causalité à partir de la séquence observée w. De relations causales entre les entrées et les sorties sont découvertes, et traduites en fragments de RdPI ; la séquence observée w est traduite en une séquence de tir S compatible avec le modèle. L'efficace de la méthode dépend toutefois de la qualité des données observées. La concurrence massive d'un système complexe induit une forme de bruit dans les données observées, sous forme de synchronisations par le contrôleur d'événements indépendants. Ce phénomène a pour conséquences:

- une hausse du coût de calcul des fonctions de tir exprimant les causalités entrées/sorties
- la création de transitions redondantes dans les fragments de RdPI construits.

Conséquemment, deux améliorations visent à améliorer la qualité de l'observation et à limiter l'impact de la concurrence :

- Figure Fr.1. Un filtre vient détecter et retirer les observations simultanées d'événements d'entrées et de sorties décorellés. Son principe est basé sur le théorème d'absorption dans les algèbres booléennes. Tout événement d'entrée, ou combinaison d'événements, qui n'est pas une condition suffisante de l'événement de sortie, est dû à une observation parasite. Les cases de la matrice de causalité correspondant à de telles observations sont annulées. Les effets du filtrage sont alors une réduction drastique du coût de calcul des fonctions de tir, ainsi qu'une amélioration de leur qualité.
- Figure Fr.2. Une procédure de réduction vient détecter et retirer les transitions redondantes. Est considérée redondante une transition au nombre d'occurences trop faible dans la séquence de tir (inférieur au nombre d'exécutions du système durant l'observation), et qui soit réductible, c'est-à-dire qui puisse être remplacée par un

ensemble de transitions équivalentes en terme de comportement. Lorsqu'une transition est retirée, toutes ses occurrences dans la séquence de tir S sont remplacées



	↑Y1	↓Y1	...	↑Yn	↓Yn
↑u1	1.000	0.101		0.540	0.000
↓u1	0.000	0.010		0.001	0.004
...			...		
↑um	0.024	0.023		0.460	0.005
↓um	0.012	0.150		0.002	0.002

	↑Y1	↓Y1	...	↑Yn	↓Yn
↑u1	1.000	0		0.540	0.000
↓u1	0.000	0		0	0.004
...			...		
↑um	0	0.023		0.460	0.005
↓um	0	0.150		0	0.002

Figure Fr.1: Principe du filtre : Annuler les cases de la matrice correspondant à des observations parasites

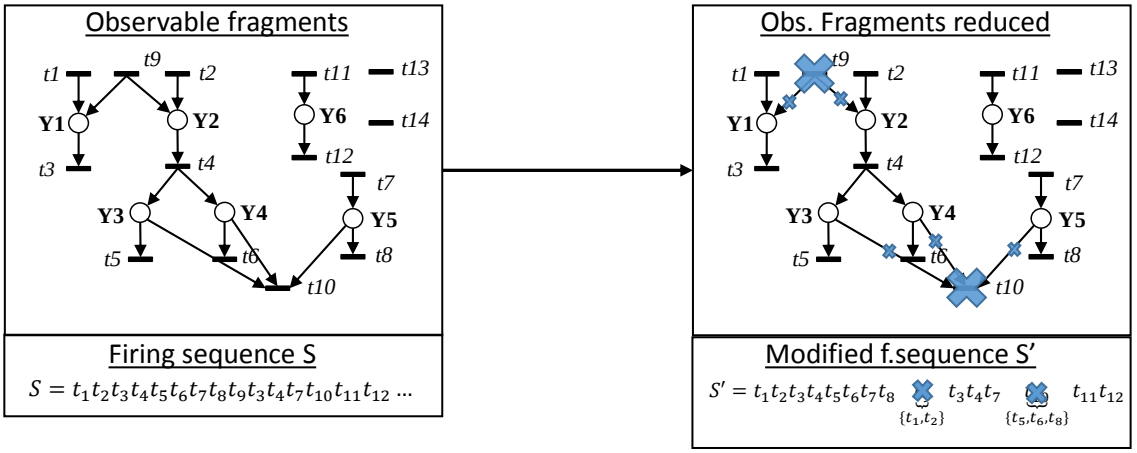


Figure Fr.2: Principe de la réduction : Suppresion des transitions, et remplacement dans S

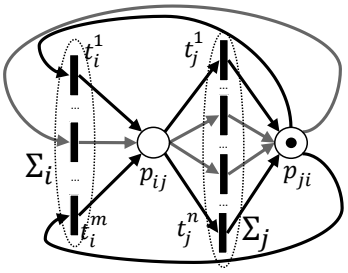
Ces améliorations ont été appliquées à un système automatisé présent au LURPA (73 E/S). Grâce à elles, le comportement observable est désormais intégralement identifié (toutes les fonctions de tir sont calculables), et le nombre de transitions est limité (réduit de 295 à 101 par la procédure de réduction).

Découverte du comportement non observable

On s'intéresse ici à la découverte de places non observables à partir de la séquence de tir S, de sorte que le RdPI final puisse la reproduire, et soit sauf.

Par le biais de projections sur sous-alphabets disjoints de l'alphabet complet des transitions, des motifs tels que celui de la figure Fr.3 sont découverts. La relation de

mutuelle dépendance $\leftrightarrows$ est ainsi définie sur l'espace des paires d'alphabets disjoints. A la relation $\Sigma_i \leftrightarrows \Sigma_j$ sont associées les deux places de la figure Fr.3. Si l'on dispose déjà d'un RdPI solution, l'ajout de ces deux places à celui-ci forme un nouveau RdPI, dont il est prouvé qu'il est également sauf et reproduit S. A partir d'un réseau initial, on peut donc itérativement construire des solutions garanties bonnes, en étudiant des projections et en rajoutant les places associées quand une mutuelle dependance est découverte.



$$\Sigma_i \leftrightarrows \Sigma_j$$
$$\Pi(S) = t_i^1 t_j^1 t_i^2 t_j^2 t_i^3 t_j^3 \ldots$$

Figure Fr.3: Motif projeté caractérisant une mutuelle dépendance, et places associées

Néanmoins, étant donné le nombre de transitions $|T|$, le nombre total de projections à étudier est approximativement $3^{|T|}$. De plus, l'étude de toutes les projections permet de garantir l'obtention de la solution la plus précise (exhibant le moins de langage excédentaire), mais souvent au prix de la compréhensibilité du réseau. Les deux modèles de la Figure Fr.4 ont été identifiés sur le même système et sont tous deux solutions. Le premier est construit à partir d'une étude limitée des projections, tandis que le second est construit par une étude complète de l'espace de recherche. Ce dernier possède le moins de langage excédentaire mais est complexe à comprendre.

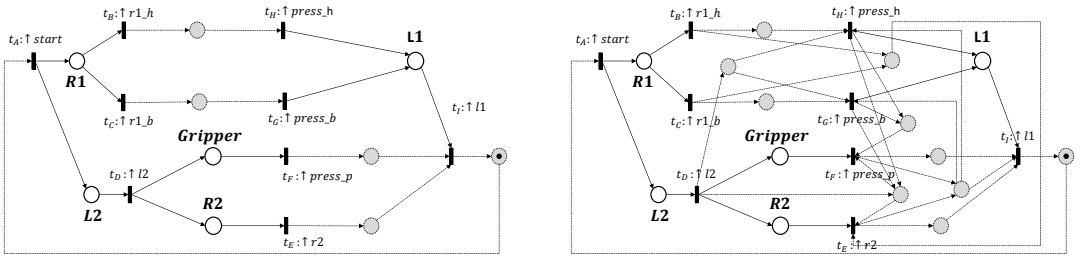


Figure Fr.4: Deux RdPI identifiés pour le même système : (a) est simple, mais génère plus de langage excédentaire que (b)

Une heuristique est proposée pour atteindre des solutions telles que le réseau (a), en procédant à une exploration limitée de l'espace de recherche. La taille des alphabets étant liée au degré des places non observables associées δ, on procède à une étude par valeurs croissantes de δ. Les séquentialités pures sont découvertes dans un premier temps ($\delta = 2$), puis les conflits ($\delta \geq 3$), qu'on suppose de faible degré pour des systèmes manufacturiers. L'exploration est stoppée quand un réseau fortement connexe est atteint après l'étude d'une valeur de δ. Les places rajoutées étant de faible degré, le réseau est

simple à lire. La taille des espaces d'état restreints est raisonnable, réduisant le coût de calcul (pour $\delta << |T|$, la taille d'un espace restreint est $|T|^{\delta}$). La méthode proposée permet ainsi de calculer des modèles monolithiques garantis justes, à temps de calcul raisonnable.

L'application de cette méthode au système automatisé conduit à la valider, mais également à montrer que les modèles monolithiques ne sont pas les plus adaptés à la compréhension des systèmes de taille conséquente, et peuvent devenir coûteux à construire.

Partitionnement automatique pour l'identification distribuée

Il peut être intéressant pour un ingénieur de disposer d'une décomposition en sous-systèmes du système complet, et d'avoir ainsi un aperçu du fonctionnement de chaque sous-système. Une approche distribuée est ainsi envisagée, la méthode d'identification étant applicable à tout sous-système. L'objectif est de trouver une décomposition du système proposant un compromis entre taille des sous-systèmes et simplicité des RdPI distribués ; un problème d'optimisation est ainsi formulé.

Pour éviter de séparer des entrées et sorties identifiées causales durant la construction du comportement observable, ce dernier sert de base à l'approche. Les fragments observables sont ainsi considérés comme des sous-systèmes élémentaires, formant ainsi une première solution. Puis, un algorithme de clustering fusionne itérativement les fragments, de façon à augmenter la taille des sous-systèmes et à construire une hiérarchie ascendante (Figure Fr.5). L'augmentation de la complexité structurelle des sous-réseaux identifiés est limitée à chaque itération.

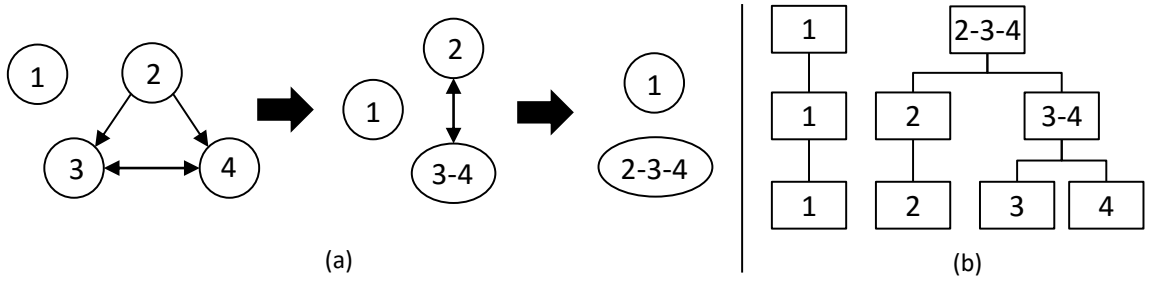


Figure Fr.5: Exemple de l'approche par clustering : Evolution d'un graphe d'affinité (a), et construction de la hiérarchie (b)

Il est difficile de prévoir le coût de la calcul total, qui peut exploser. Pour limiter ce dernier, deux limitations sont proposées. Si les fragments initiaux sont de taille similaire, il est possible de limiter la taille (en nombre de transitions) des modèles identifiés. Sinon, il est également possible de limiter le temps de calcul alloué au calcul de chaque sous-modèle. La décomposition identifiée pour un coût de calcul limité à $t = 20s$ sur le système test est montrée en Figure Fr.6. Les fragments identifiés sont interprétables, et le partitionnement est cohérent avec la répartition physique des composants dans le

système.

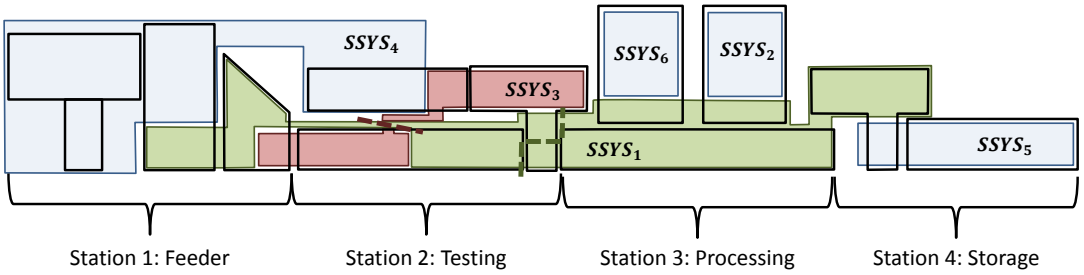


Figure Fr.6: Localisation des six sous-systèmes obtenus par time-clustering sur la MSS, $t_{Lim} = 20s$

Conclusion

Les apports proposés dans cette thèse permettent d'obtenir des modèles RdPI lisibles de systèmes, considérés comme complexes par leur taille et leur comportement. Dans un premier temps, l'approche monolithique est améliorée à la fois par le développement de techniques de filtrage limitant l'impact de la concurrence, et par une nouvelle approche de découverte de comportement non observable garantissant la justesse du modèle identifié. Dans un second temps, une approche de clustering permet le partitionnement automatique du système complexe en sous-systèmes, exploitables par une approche distribuée.

Pour prolonger ces travaux, une approche boîte grise pourrait être considérée afin de découvrir des comportements non observables spécifiques (compteurs, boucles,...), par fouille de données par exemple. Les temporisations fonctionnelles du système pourraient également être découvertes et représentées dans une telle approche. Dans un contexte boîte grise, l'identification active peut également être envisagée, permettant de découvrir des comportements admissibles supplémentaires, la boîte grise permettant de limiter le risque d'endommagement. Avec ces nouvelles directions, de nouvelles applications aux modèles identifiés peuvent être envisagées, telles que la certification de systèmes, ou la réimplémentation de contrôleurs.

Appendix A
Assessing the quality of an identified Petri net

A.1 Precision

The closeness, in terms of behaviour, of an identified net to an observed firing sequence, can be qualified and quantified by using languages. Recall that the observation is finite and incomplete (Section 1.2.2), whereas the language generated by a net is infinite as soon as it exhibits cyclicity. To compare the languages, the length n of words is considered, and the languages are parametrized by n.

Consider a firing sequence $S = s_1 s_2 \dots s_{|S|}$ and a net $N = (G, M_0)$. The definition of the language generated by N was given by Definition 1.12. With the inclusion of n, the following languages are firstly considered:

Definition A.1. *The language of length n generated by a Petri net N , i.e. the set of words of length n generated by N, or identified language is:*

$$L^n_{Id}(N) = \{w \in T^* |\ |w| = n\ \wedge \exists M \in R(N),\ M_0 \xrightarrow{w} M\}$$

The language of length n, observed in a firing sequence S, is:

$$L^n_{Obs}(S) = \{s_1 s_2 \dots s_n\}$$

The exceeding language of length n is the difference of these two sets. The exceeding language of length up to n is therefore:

$$L^n_{Exc}(N, S) = \bigcup_{i=1}^{n} L^i_{Id}(N) \setminus L^i_{Obs}(S)$$

Following this definition, given an integer n, there is only one word of this length included in the sequence. It comes that $|L^n_{Exc}(N, S)| = |L^n_{Id}| - 1$. A precise net has few exceeding language, so aiming for precision means decreasing the size of the identified language. These definitions are often used in the litterature, but inadapted for systems exhibiting concurrency, as illustrated by the next example:

Example A.1. *Consider two sequences:*

$$S = t_2t_3t_4\ t_5t_1\ t_2t_4t_3\ t_5t_1\ t_3t_2t_4\ t_5t_1\ t_3t_4t_2\ t_5t_1\ t_4t_3t_2\ t_5t_1\ t_4t_2t_3\ t_5t_1$$
$$S' = t_5t_1\ t_2t_3t_4\ t_5t_1\ t_2t_4t_3\ t_5t_1\ t_3t_2t_4\ t_5t_1\ t_3t_4t_2\ t_5t_1\ t_4t_3t_2\ t_5t_1\ t_4t_2t_3$$

The two sequences represent the same system, but the observation began at different times of its operation. Let N of Figure A.1 be the identified net for S, while N' is the net identified for S'.

N exhibits the parallelism observed between transitions t_2, t_3, t_4. However, the exceeding language L^n_{Exc} plotted in Figure A.1 is not zero at least for the first values of n. Given that for small values of n, there is already exceeding behaviour, the model could be discarded as bad even though it mirrors perfectly the behaviour included in the sequence.

N' exhibits the same parallelism, but the exceeding language is equal to zero for n up to 2, since the initial marking has changed.

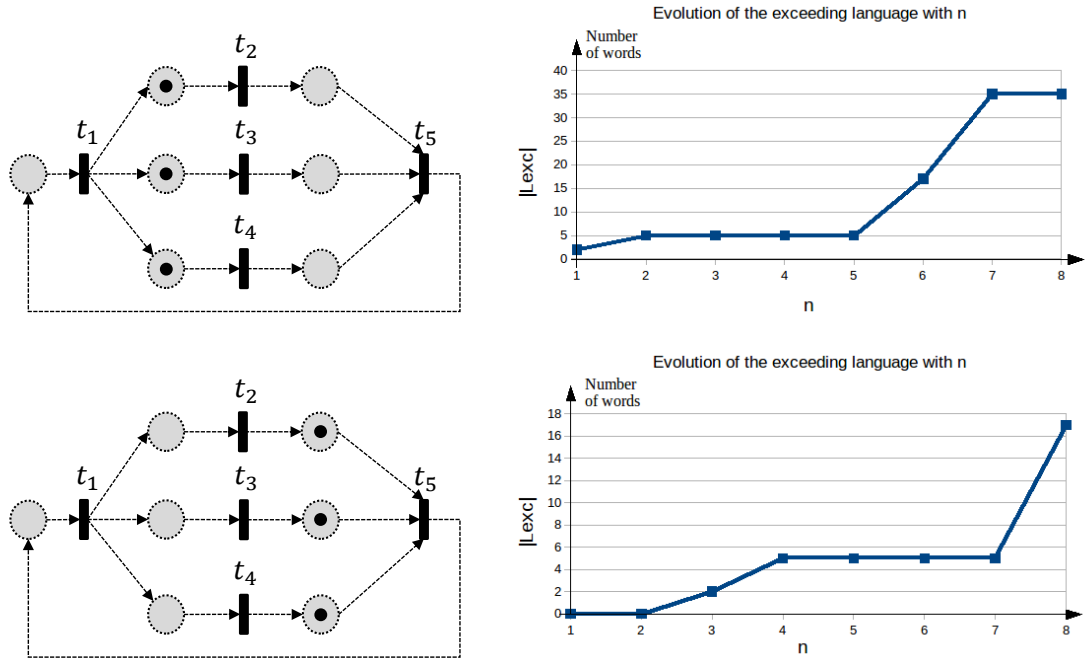


Figure A.1: First row: N identified for S, and its exceeding language; Second row: N' identified for S', and its exceeding language

Basically, the same net structure exhibiting concurrency can be seen as either precise or unprecise depending on the initial marking. To better express the closeness of an identified model to a sequence, namely when parallelism is involved, alternative definitions are proposed. The identified language accounts for all accessible markings instead of only the initial one. To be consistent, the observed language is defined by all substrings of length n, as if a sliding window of length n was moved along the firing sequence.

Definition A.2 (Alternative languages)**.** *The language of length n generated by a Petri*

net N , i.e. the set of words of length n generated by N, or identified language is:

$$L^n_{Id2}(N) = \{w \in T^* | \ |w| = n \ \wedge \exists M' \in R(N), \ M \xrightarrow{w} M'\}$$

The language of length n, observed in a firing sequence S is:

$$L^n_{Obs2}(S) = \bigcup_{1 \leq t \leq |S|-n+1} s_t.s_{t+1} \ldots s_{t+n-1}$$

The definition of exceeding language remains the same.

The results of applying this definition of language to Example A.1 is presented in Figure A.2. There is no exceeding language for the first values (n=1,2,3); therefore the identified net can be considered quite fitting the sequence. The size of the exceeding language increases for higher values of n because the sequence S is short and does not exhibit that much behaviour.

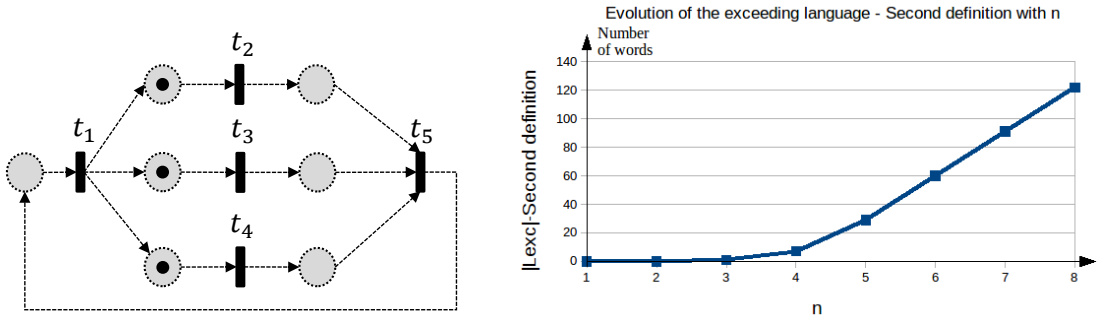


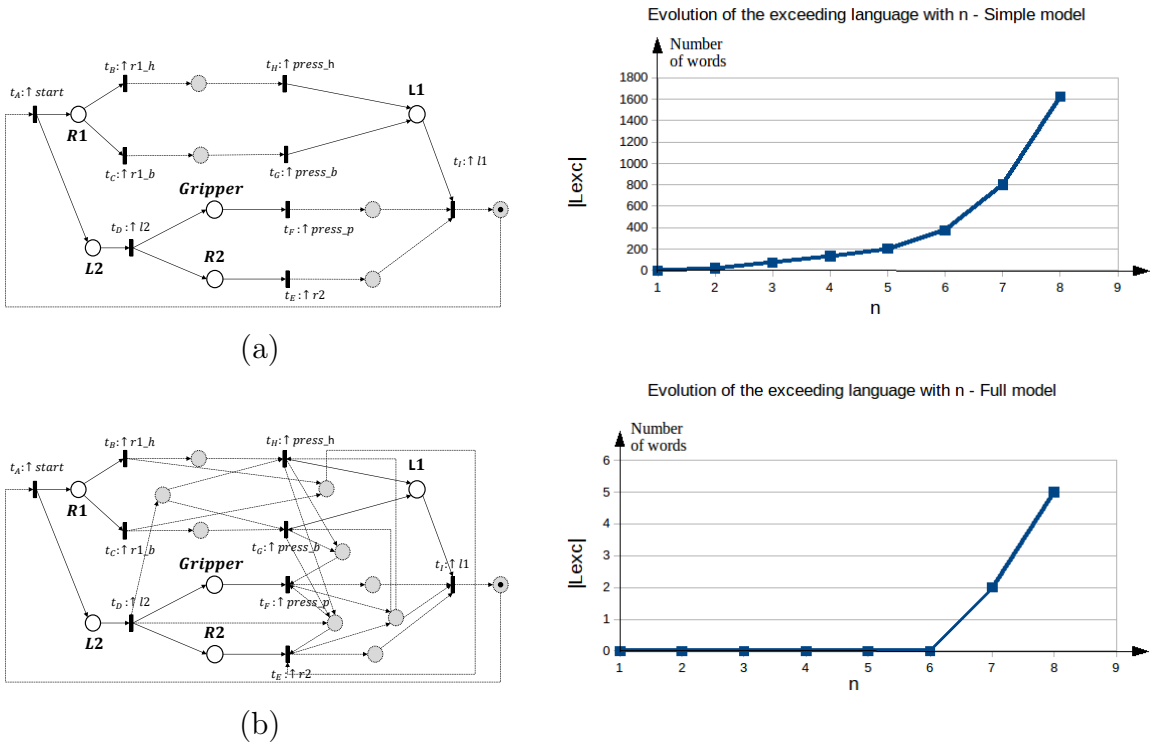
Figure A.2: The net N identified and its exceeding language according to Definition A.2

Definition A.2 will from now on be the reference for any qualification of the behaviour of the identified net. If the problem was a pure synthesis problem, the only objective of any procedure would be to minimize L^n_{Exc} for all values of n, even nullify L^n_{Exc} for all values of n up to a n_{max} that should be maximal. However, in our reverse-engineering approach, understandability of the resulting net is also of importance.

A.2 Complexity metrics

A Petri net consists of two parts : a structure G and an initial marking M_0. The structure can be considered as the static part of the net, whereas the initial marking unlocks dynamical properties of the system, expressed by the reachability graph.

Therefore, different metrics are considered for the structural and the dynamic complexity. They are evaluated on the example presented in the main corpus of the thesis, recalled here in Figure A.3.



(a)

(b)

Figure A.3: (a) A simple solution, with a lot of exceeding language ; (b) A complex solution, with no exceeding language up to $n = 6$

A.2.1 Structural Complexity

Most metrics assessing the structural complexity of a Petri net, or more generally of an oriented graph, are issued from the field of software engineering. Expressing fragments of code as an oriented graph is a way of visualizing and controlling the flow of operations [McCabe, 1976]; a measure of the complexity of the graph is a measure of the complexity of the associated code, and a way of prediciting where eventual mistakes can be made.

Let $G = (V, E)$ be an oriented graph, with V the set of nodes and E the set of edges. Metrics assessing the size of the graph (number of edges $|E|$ and number of nodes $|V|$) are too simple to assess the structural complexity of the graph and are left apart

A first simple and elegant metric, called Coefficient of Network Complexity (CNC) has been introduced by Pascoe ([Pascoe, 1966]): it is the ratio edges/nodes of the graph:

Definition A.3 (Coefficient of Netwotk Complexity)**.** *The original definition [Pascoe, 1966] for a graph $G = (V, E)$:*

$$CNC = \frac{|E|}{|V|}$$

This definition can be translated to a Petri net structure $G = (P, T, W)$, where P and T mirror the set of nodes V, and W is the set of edges mirroring E

$$CNC_{PN} = \frac{|W|}{|P| + |T|}$$

CNC expresses the mean number of edges entering a node (or equivalently, the mean number of edges leaving a node); it does however not make a distinction between places and transitions. The authors of [Soo and Jung-Mo, 1992] consider the case of Petri nets only, and propose two additional structural metrics taking into account the bipartite nature of the graph: the Average Transition Output(ATO) and Average Place Output(APO)

Definition A.4 (Average Transition and Place Output)**.** *Let $G = (P, T, W)$ be a Petri net structure.*

Average Transition Output:

$$ATO = \frac{1}{|T|} \sum_{t \in T} |t^\bullet|$$

Average Place Output:

$$APO = \frac{1}{|P|} \sum_{p \in P} |p^\bullet|$$

Given a transition, ATO expresses the mean number of aval places, hence the mean number of parallel processes that the transition starts. Given a place, APO expresses the mean number of aval transitions, hence the number of conflictuous processes that the place enables when filled with a token. Both these metrics express structural complexity. Furthermore, it can be noted that:

$$\sum_{t \in T} |t^\bullet| + \sum_{p \in P} |p^\bullet| = |W|$$

Therefore:

$$|T|.ATO + |P|.APO = (|P| + |T|).CNC_{PN}$$

CNC_{PN} can be considered as the barycenter of both ATO and APO, and is thus a relevant measure to express the structural complexity as well.

Another widely recognized complexity measure is the cyclomatic complexity, first introduced by McCabe in [McCabe, 1976]. It represents the number of independant paths in the graph, and is defined as following:

Definition A.5 (Cyclomatic Complexity)**.** *Let $G = (E, V)$ be an oriented graph, and p its number of strongly connected components. Then the cyclomatic complexity is :*

$$CC = E - V + 2p$$

Let $G = (P, T, W)$ be a Petri net, and sc its number of strongly connected components. Then the cyclomatic complexity is :

$$CC_{PN} = |W| - (|P| + |T|) + 2.sc$$

Both CC and CNC involve the number of places, transitions and edges, the main difference being that CC also involves the number of strongly connected components. Trying to understand multiple Petri nets as once, namely all possibilities for parallelism is a complicated task. However, if two processes of a system are totally parallel (*i.e.* independant), they can be considered as two subsystems and be identified separately. For instance in example 4.5, the two chariots can roll independantly, but are synchronised by the start button: they can hence not be considered as independant subsystems. Should the transfer dock not exist, they would be fully independant systems, and the identification procedure could be run separately on each of them.

A reasonable assumption is therefore that the identified model should consist in one strongly connected net to be relevant. If *sc* is set up to 1 in CC_{PN}, CNC_{PN} and CC_{PN} express roughly the same variations. Therefore, CNC_{PN} should be a good enough metric for the purpose of evaluating the structural complexity of an identified Petri net. This opinion is for instance shared by the authors of [Mendling et al., 2007], that consider this metric (called *average connector degree*) as relevant to relate the understandability of the net. In the field of cognitive science, the author of [Rauterberg, 1992] has evaluated that both CNC and CC are relevant to evaluate the complexity of a given task (namely interacting with a database management system).

Example A.2. *Looking back at the two identified nets of Figure A.3. The values of the different structural metrics presented in this section are given in Table A.1 Indeniably, whichever the chosen metric, the second net is evaluated more complex than net (a).*

Metric	Net (a)	Net (b)
|T|	9	9
|P|	10	15
|W|	22	42
CNC_{PN}	1.16	1.82
CC_{PN}	5	21
APO	1.1	1.26
ATO	1.1	2.55

Table A.1: Comparison of structural complexity metrics on Example 4.5

A.2.2 Dynamic Complexity

Petri net dynamics are closely related to the language. Given a marking M_0 and a marking M reachable from M_0, the number of transitions enabled from M gives the evolution possibilities of the net state. To have access to the mean number of enabled transitions, the building of the reachability graph is required:

Definition A.6 (Branching Degree(BD))**.** *Let $N = (G, M_0)$ be a Petri net and R(N)=(S,E) be its reachability graph (with E the set of edges and S the set of states). Then, the*

Branching Degree defined as following represents the mean number of transitions enabled at any state:

$$BD = \frac{1}{|S|} \sum_{s \in S} outdeg(s) = \frac{|E|}{|S|}$$

where $outdeg(s)$ is the out-degree of state s, i.e. the transitions enabled from the marking M associated to s.

Recall definition A.2 of the language generated by the net. Given a marking M, consider the set of words of length n that lead to M. Since BD transitions are enabled in this state, these words can be prolongated with BD different transitions, leading to BD words of length $n+1$ for each word leading to M. This being true for each marking, each word of length n can roughly create BD words of length $n+1$.

The following approximation can be made : $|L^{n+1}(N)| \simeq |L^n(N)|.BD$; given that $|L^1(N)| = |T|$, it comes $|L^n(N)| \simeq |T|.BD^{n-1}$. This approximation shows that BD is linked to the language generated by the net. Looking back at example 4.5, the second net generates less language than the first one. BD(a)=42/21=2, whereas BD(b)=15/12=1.25. The dynamic is indeed less complex in the second net, but much more difficult to understand. BD can therefore not be used to assess of the understandability of the net.

BD is basically CNC applied to the reachability graph. Similarly, other structural metrics can be applied to the reachability graph. For instance, the authors of [Lassen and van der Aalst, 2009] used the cyclomatic complexity (CC) and concluded as well that a small reachability graph is not correlated to an understandable net.

To capture the essence of the dynamic of the net, one should be able to picture the flow of tokens during the execution of a sequence. Therefore, the average number of tokens can be more representative of the dynamic of a graph; it mirrors the number of processes simultaneously running.

Definition A.7. *Let $N = (G, M_0)$ be a Petri net and R(N)=(S,E) be its reachability graph. The Average Number of Tokens (AToks) in the graph is defined as following:*

$$AToks = \frac{1}{|S|} \sum_{s \in S} \left(\sum_{p \in [\![1,|P|]\!]} M_s(p) \right)$$

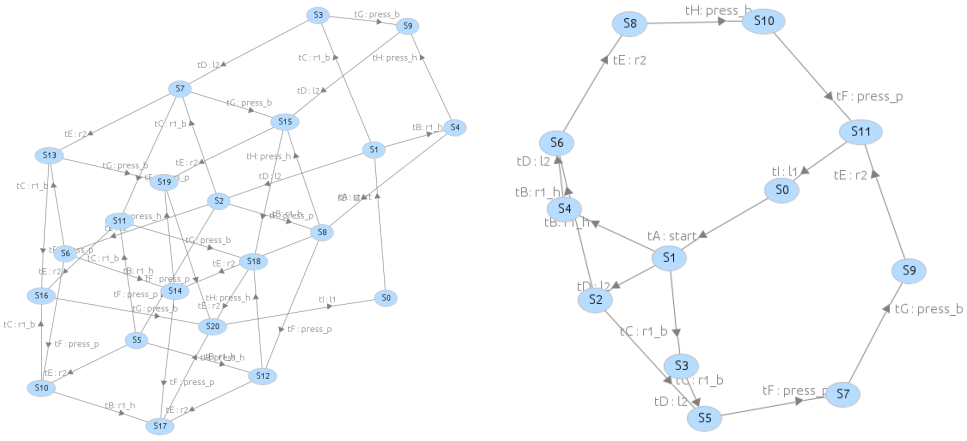
The authors of [Soo and Jung-Mo, 1992] consider this criterion, with the size of the reachability graph, as the best metric of dynamic complexity. However, notice that the solutions of the identification problem are 1-bounded. Since there is at most 1 token in a place, n tokens in the net represent n concurrent processes that are simultaneously running. Therefore, AToks represents the mean number of concurrent processes in the system, hence a difficulty of understanding induced by the complexity of the physical system, but not by the model.

Example A.3. *Looking back at the two identified nets of Figure A.3, their reachability*

graphs are given in Figure A.4 The values of the different dynamic metrics presented in this section are given in Table A.2.

Metric	Net (a)	Net (b)
|S|	21	12
|E|	42	15
ATok	2.7	4,16
BD	2	1.25
CC	23	5

Table A.2: Comparison of dynamic complexity metrics on Example 4.5



(a) (b)

Figure A.4: Reachability graphs of the nets of Figure A.3

BD and CC applied to the reachability graph assess that net (b) is less complex than net (a), which is logical given that the second net generates less language that the first one. However, net (b) is hard to understand, which is mirrored by ATok. Evaluating the behaviour of the net requires to follow the flow of less tokens in net (a) than in net (b)

Dynamic complexity metrics are related to the inherent complexity of the system, not of its model representation; whereas structural complexity metrics are related to the model. Therefore, only a structural complexity metric is kept to assess the understandability of a model. Namely, CNC is chosen.

Appendix B
Proofs

Proposition 3.5 (Chapter 3, Page 78) *Let $\{t_i\}_{1 \leq i \leq n}$ be a reduction of t. Then, there exists a firing sequence σ, containing exactly one firing of each transition of $\{t_i\}$, that always results in the same marking evolution as the firing of t, i.e.:*

$$\exists \sigma = t_1 t_2 \dots t_n \mid \forall (M, M'), M \xrightarrow{t} M' \iff M \xrightarrow{\sigma} M'$$

σ is called a replacement for t

Proof. The equality of the markings reached by firing t or σ is a consequence of Proposition 3.4. It remains to prove firability.

Firability $\Leftarrow$ Suppose that $M \xrightarrow{\sigma} M'$, and that t is not firable from M. Let $p \in {}^\bullet t$ be a place such that $M(p) = 0$, forbidding the firing of t. The event $\downarrow \varphi(p)$ was however generated by the firing of σ. Let M_{int} be a marking such that:

$$M \xrightarrow{\sigma-} M_{int} \xrightarrow{\sigma+} M'$$

with $\sigma - \sigma+ = \sigma$ and $M_{int}(p) = 1$. $\downarrow \varphi(p)$ is generated by the firing of $sigma+$. However, since $M(p) = 0$, $\uparrow \varphi(p)$ must occur in $\sigma-$. Consequently, both $\downarrow \varphi(p)$ and $\uparrow \varphi(p)$ belong to $RI(\{t_i\})$, which is impossible since both the rising and the falling edge of an output event can not belong to a RI set. Contradiction reached. Necessarily, t is firable in M

Firability $\Rightarrow$ By recursivity. Suppose that $M \xrightarrow{t} M'$, and that t_1 is not firable in M. Let $p \in {}^\bullet t_1$ be a place such that $M(p) = 0$. $\downarrow \varphi(p) \in RI(t_1)$, and $RI(t_1) \subset RI(t)$. Therefore, $\downarrow \varphi(p) \in RI(t)$, and $p \in {}^\bullet t$. However, $M \xrightarrow{t}$, hence $M(p) = 1$, and a contradiction. Necessarily, t_1 is firable in M.

Suppose now that $M \xrightarrow{t_1 \dots t_{i-1}} M''$, and that t_i is not firable in M''. By the same reasoning, a place p is exhibited such that $M''(p) = 0$, and verifying $p \in {}^\bullet t$, hence $M(p) = 1$. p was necessarily emptied by the firing of a transition t_j in $\{t_1, \dots, t_{i-1}\}$. Therefore, $\downarrow \varphi(p) \in RI(t_j)$. However, $\downarrow \varphi(p) \in RI(t_i)$ as well,

and by definition of a reduction, $RI(t_j) \cap RI(t_i) = \emptyset$, hence the contradiction. Necessarily, t_i is firable in M''.

□

Proposition 4.4 (Chapter 4, Page 92) *Let Σ_n be a size n alphabet ($n \geq 2$). The number of couples of disjoint, non-empty subsets of Σ_n, noted P_n, is given by the formula:*

$$P_n = 3^{n-2} + \sum_{k=0}^{n-3} 3^k (2^{n-k-1} - 1)$$

Proof. The proof is conducted by recurrence

For $n = 2$, $P_2 = 3^0 = 1$. There are only two non-empty disjoint subsets of Σ_2, both of size 1; hence only one couple. The formula is true for $n = 2$

Suppose that P_n is true.

Let Σ_{n+1} be an alphabet of size $n+1$ and t_{n+1} such that $\Sigma_{n+1} = \Sigma_n \cup \{t_{n+1}\}$, with Σ_n a size n alphabet. There are P_n non-empty disjoint subsets of Σ_n. Let (Σ_i, Σ_j) be one of these couples. By adding t_{n+1}, three non-empty disjoint subsets of Σ_{n+1} can be built:

- $(\Sigma_i \cup \{t_{n+1}\}, \Sigma_j)$
- $(\Sigma_i, \Sigma_j \cup \{t_{n+1}\})$
- (Σ_i, Σ_j)

For each couple in Σ_n, three couples in Σ_{n+1} can be built.

It remains to count all couples involving the singleton $\{t_{n+1}\}$: $(\{t_{n+1}\}, \Sigma'_n)$, where Σ'_n is a non-empty subset of Σ_n. Exactly $(2^n - 1)$ of these subsets exist. Consequently:

$$P_{n+1} = 3P_n + (2^n - 1)$$

Replacing P_n by its hypothetic expression:

$$P_{n+1} = 3\left[3^{n-2} + \sum_{k=0}^{n-3} 3^k (2^{n-k-1} - 1)\right] + (2^n - 1)$$

i.e.:

$$P_{n+1} = 3^{n-1} + \sum_{k=0}^{n-3} 3^{k+1} (2^{n-k-1} - 1) + (2^n - 1)$$

After reindexing k in the sum:

$$P_{n+1} = 3^{n-1} + \sum_{k=1}^{n-2} 3^k (2^{n-k} - 1) + (2^n - 1)$$

Including $(2^n - 1)$ in the sum:

$$\boxed{P_{n+1} = 3^{n-1} + \sum_{k=0}^{n-2} 3^k (2^{n-k} - 1)}$$

The formula is proven correct for $n+1$, therefore it is true for all $n \geq 2$. □

Proposition 4.5(Chapter 4, Page 96) *Let $N = \{(P, T, C), M_0\}$ be a PN, and $L_{M_0}(N) = \{S | M_0 \xrightarrow{S}\}$ the identified language, i.e. the (infinite) set of sequences firable from the initial marking. Let $N' = \{(P', T, C'), M'_0\}$ be a PN constructed from N by adding a place p with its edges $i[T]$ and $o[T]$, such that $c_p[T] = o[T] - i[T]$, and its initial marking m_p, i.e. :*

$$P' = P \cup \{p\}$$
$$C' = \begin{bmatrix} C \\ c_p[T] \end{bmatrix}$$
$$M'_0 = \begin{bmatrix} M_0 \\ m_p \end{bmatrix}$$

Then $L_{M'_0}(N') \subseteq L_{M_0}(N)$.

Proof. Suppose that $L_{M'_0}(N') \not\subseteq L_{M_0}(N)$.

Then $\exists w \in L_{M'_0}(N'), w \notin L_{M_0}(N)$. Let w be such a sequence in $L_{M'_0}(N')$, w is not firable in N. Therefore it can be written $w = \sigma_1 t \sigma_2$, $(\sigma_1, \sigma_2) \in (T^*)^2$, with $\sigma_1 \in L_{M'_0}(N') \cap L_{M_0}(N)$, and t the first transition of w firable in N' but not in N.

Let M_{σ_1} (resp. M'_{σ_1}) be the marking reached in N (resp. N') after the firing of σ_1:

$$M_{\sigma_1} = M_0 + C\sigma_1$$
$$M'_{\sigma_1} = \begin{bmatrix} M_0 \\ m_p \end{bmatrix} + \begin{bmatrix} C \\ c_p[T] \end{bmatrix} \sigma_1 = \begin{bmatrix} M_{\sigma_1} \\ m_p + c_p[T]\sigma_1 \end{bmatrix}$$

Hence $\forall p_k \in P, M_{\sigma_1}[p_k] = M'_{\sigma_1}[p_k]$ (unaffected places keep the same marking).

t is not firable in N from M_{σ_1}. Necessarily, there exist a place in amount of t, $p_t \in \{p_i \in P | C[p_i, t] < 0\}$, such that:

$$M_{\sigma_1}[p_t] < -C[p_t, t]$$

However, t is firable in N' from M'_{σ_1}, hence all places in amount of t, $\{p_i \in P' | C'[p_i, t] < 0\}$ satisfy $M'_{\sigma_1}[p_i] \geq -C'[p_i, t]$, namely :

$$M'_{\sigma_1}[p_t] \geq -C'[p_t, t]$$

However $p_t \in P$, hence $M_{\sigma_1}[p_t] = M'_{\sigma_1}[p_t]$ and $C[p_t, t] = C'[p_t, t]$. The two inequalities are contradictory. t, then w can not exist, therefore $L_{M'_0}(N') \subseteq L_{M_0}(N)$. □

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Identification Comportementale Boîte-noire de Systèmes à Evénements Discrets par Réseaux de Petri Interprétés

Mots clés : *Identification, Systèmes à Evénements Discrets, Systèmes réactifs, Réseaux de Petri Interprétés, Rétro-ingénierie, Partitionnement automatique.*

Résumé : Cette thèse contribue à l'identification de modèles compacts et expressifs de Systèmes à Evénements Discrets (SED) réactifs, à des fins de rétro-conception ou de certification. L'identification est passive, et boîte-noire, la connaissance se limitant aux signaux d'entrées/sorties du système. Les Réseaux de Petri Interprétés (RdPI) permettent de modéliser à la fois le comportement observable (causalités entrées/sorties observées directement), et le comportement non observable du système (évolutions de variables internes). Cette thèse vise à identifier des modèles RdPI à partir d'une séquence observée de vecteurs entrées/sorties. Notamment, l'enjeu étant de traiter des systèmes concurrents de taille réaliste, l'approche développée permet le passage à l'échelle de résultats précédents. La construction de la partie observable est d'abord améliorée par l'ajout d'un filtre. Celui-ci détecte et supprime les synchronisations parasites causées par le contrôleur en présence de systèmes concurrents. Une nouvelle approche est ensuite proposée pour découvrir la partie non observable, basée sur l'utilisation de projections, et garantissant la reproductibilité du comportement observé malgré la concurrence. Une heuristique permet de construire un modèle satisfaisant pour la rétro-ingénierie, à coût de calcul limité. Enfin, une approche distribuée est proposée pour réduire davantage le coût de calcul, en partitionnant automatiquement le système en sous-systèmes. L'effet cumulatif de ces contributions est illustré par l'identification de RdPI sur un système de taille raisonnable, validant leur efficacité.

Title : Blackbox Behavioural Identification of Discrete Event Systems by Interpreted Petri Nets

Keywords : *Identification, Discrete Event Systems, Reactive systems, Interpreted Petri Nets, Reverse engineering, Automated partitioning*

Abstract : This thesis proposes a method to identify compact and expressive models of closed-loop reactive Discrete Event Systems (DES), for reverse-engineering or certification. The identification is passive, and blackbox, accessible knowledge being limited to input/output signals. Interpreted Petri Nets (IPN) represent both the observable behaviour (direct input/output causalities) and the unobservable behaviour (internal state evolutions) of the system. This thesis aims at identifying IPN models from an observed sequence of I/O vectors. The proposed contributions extend previous results towards scalability, to deal with realistic systems who exhibit concurrency. Firstly, the construction of the observable part of the IPN is improved by the addition of a filter limiting the effect of concurrency. It detects and removes spurious synchronizations caused by the controller. Then, a new approach is proposed to improve the discovery of the unobservable part. It is based on the use of projections and guarantees the reproduction of the observed behaviour, despite concurrency. An efficient heuristic is proposed to compute a model adapted to reverse-engineering, limiting the computational cost. Finally, a distributed approach is proposed to further reduce the computational cost, by automatically partitioning the system into subsystems. The efficiency of the cumulative effect of these contributions is validated on a system of realistic size.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France