

UNIVERSITY OF VERONA

DEPARTMENT OF COMPUTER SCIENCE

GRADUATE SCHOOL OF NATURAL SCIENCE AND ENGINEERING

DOCTORAL PROGRAM IN COMPUTER SCIENCE

CYCLE 35

Decomposition of sequential and concurrent models

S.S.D. ING-INF/05

Coordinator: _____

Prof. Ferdinando Cicalese

Tutor: _____




Prof. Tiziano Villa

Doctoral Student: _____

Dott. Viktor Teren

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License, Italy. To read a copy of the licence, visit the web page:

<http://creativecommons.org/licenses/by-nc-nd/3.0/>

-  **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
-  **NonCommercial** — You may not use the material for commercial purposes.
-  **NoDerivatives** — If you remix, transform, or build upon the material, you may not distribute the modified material.

Decomposition of sequential and concurrent models— VIKTOR TEREN

PhD Thesis

Verona, 28 dicembre 2023

Abstract

Finite State Machines (FSMs), transition systems (TSs) and Petri nets (PNs) are important models of computation ubiquitous in formal methods for modeling systems. Important problems involve the transition from one model to another. This thesis explores Petri nets, transition systems and Finite State Machines decomposition and optimization. The first part addresses decomposition of transition systems and Petri nets, based on the theory of regions, representing them by means of restricted PNs, e.g., State Machines (SMs) and Free-choice Petri nets (FCPNs). We show that the property called “excitation-closure” is sufficient to produce a set of synchronized Petri nets bisimilar to the original transition system or to the initial Petri net (if the decomposition starts from a PN), proving by construction the existence of a bisimulation. Furthermore, we implemented a software performing the decomposition of transition systems, and reported extensive experiments. The second part of the dissertation discusses Multiple Synchronized Finite State Machines (MSFSMs) specifying a set of FSMs synchronized by specific primitives: Wait State and Transition Barrier. It introduces a method for converting Petri nets into synchronous circuits using MSFSM, identifies errors in the initial approach, and provides corrections.

Abstract (italian)

Le macchine a stati finiti (FSM), sistemi di transizioni (TS) e le reti di Petri (PN) sono importanti modelli formali per la progettazione di sistemi. Un problema fondamentale è la conversione da un modello all'altro. Questa tesi esplora il mondo delle reti di Petri e della decomposizione di sistemi di transizioni. Per quanto riguarda la decomposizione dei sistemi di transizioni, la teoria delle regioni rappresenta la colonna portante dell'intero processo di decomposizione, mirato soprattutto a decomposizioni che utilizzano due sottoclassi delle reti di Petri: macchine a stati e reti di Petri a scelta libera. Nella tesi si dimostra che una proprietà chiamata "chiusura rispetto all'eccitazione" (excitation-closure) è sufficiente per produrre un insieme di reti di Petri la cui sincronizzazione è bisimile al sistema di transizioni (o rete di Petri di partenza, se la decomposizione parte da una rete di Petri), dimostrando costruttivamente l'esistenza di una bisimulazione. Inoltre, è stato implementato un software che esegue la decomposizione dei sistemi di transizioni, per rafforzare i risultati teorici con dati sperimentali sistematici. Nella seconda parte della dissertazione si analizza un nuovo modello chiamato MSFSM, che rappresenta un insieme di FSM sincronizzate da due primitive specifiche (Wait State - Stato d'Attesa e Transition Barrier - Barriera di Transizione). Tale modello trova un utilizzo significativo nella sintesi di circuiti sincroni a partire da reti di Petri a scelta libera. In particolare vengono identificati degli errori nell'approccio originale, fornendo delle correzioni.

Acknowledgments

I want to express my deep gratitude for the support and encouragement that I received during my PhD journey. Without the guidance, patience, and expertise of many who supported me, completing this thesis would not have been possible.

I am profoundly thankful to my supervisor, Tiziano Villa, whose expertise, understanding, and especially patience were fundamental during these years and significantly enriched my experience. I extend my deepest gratitude to Jordi Cortadella, my supervisor at Universitat Politècnica de Catalunya. His profound knowledge and guidance were not only crucial during my time there, but also significantly influenced my work before and after my stay abroad. I also extend my gratitude to the reviewers, Alex Yakovlev and Luciano Lavagno, for their insightful and valuable suggestions during the thesis revision. Special thanks to Valentina Napoletani for her contributions to my research on MSFSM.

I express my sincere thanks to the University of Verona for providing an excellent research environment and financial support during my Ph.D. studies.

Despite facing the challenges of remote work due to the Covid-19 pandemic for a significant portion of my Ph.D., I am grateful to the wonderful colleagues for all the fun we had in the last years.

My deepest gratitude goes to my mother for her unwavering support and continuous encouragement throughout my years of study and the thesis-writing process. This achievement would not have been possible without her. Thank you. I also want to thank other family members, in particular my brothers, for believing in me during these years.

I extend my thanks to my friends. Michael for for having been my moral support for a very long time. Nadir, Victor, and Derrick for the fun times we shared. Christian, thank you for our endless and stimulating dialogues. I also want to express my appreciation to Lia, Diana, Francesca, and many others.

Lastly, I want to acknowledge my roommate and friend, Milana, for bringing brightness and enthusiasm into my life during my stay in Barcelona.

With heartfelt thanks,
Viktor

Contents

Part I Introduction and preliminaries

1	Introduction	3
1.1	Practical application of Petri net decomposition	3
1.2	Overview	8
2	Preliminaries	13
2.1	Finite State Machines and Transition systems	13
2.2	Petri Nets and Signal Transition Graphs	15
2.3	Region theory and Petri net synthesis	19
2.3.1	Definitions and conversion flow overview	19
2.3.2	Petri net synthesis	21
2.3.3	Minimal pre-regions generation	21
2.3.4	Label Splitting	24
2.3.5	Redundant pre-regions removal	25
2.3.6	Minimal pre-regions merging	26
2.4	Binary Decision Diagram (BDD)	26

Part II Decomposition based on regions theory

3	Decomposition into sets of synchronizing PNs	31
3.1	Evolution of the decomposition	31
3.2	Decomposition based on theory of regions	32
3.3	Composition of PNs and equivalence to the original TS/PN	34
3.3.1	Safe composition of unsafe PNs	34
3.3.2	Proof of Theorem 4	37
3.4	What happens if excitation-closure is not satisfied?	39
3.5	What about the decomposition into sets of synchronizing Marked Graphs?	40

4	Transition System decomposition into sets of synchronizing SMs	43
4.1	Sequential SM search	44
4.1.1	Generation of a set of SMs with excitation closure	45
4.1.2	Removal of the redundant SMs	47
4.1.3	Merge between regions preserving the excitation closure	48
5	Transition System decomposition into sets of synchronizing FCPNs	53
5.1	Overview	53
5.2	Sequential FCPN search	55
5.3	Decomposition into k FCPNs simultaneously	56
5.3.1	Example of k FCPN simultaneous decomposition	57
5.4	Additional constraint: safeness for each FCPN	61
5.5	Decomposition optimization	62
5.6	Decomposition into a set of synchronizing ACPNs	65
6	Experimental results	67
6.1	SMs	67
6.1.1	Creation of a new mixed strategy	71
6.1.2	Simultaneous SM search	72
6.1.3	SMs without guarantee the safeness of single components	73
6.2	FCPNs	73
6.2.1	FCPNs without guarantee the safeness of single components	73
6.2.2	Safe FCPNs	75
6.2.3	Reset of the learned clauses	79

Part III Multiple Synchronized FSMs

7	MSFSM model	83
7.1	Other models of concurrency	85
7.1.1	Communicating Sequential Processes (CSP)	85
7.1.2	Synchronous Languages and their representations	86
7.1.3	Why MSFSMs?	88
7.2	Creation of synchronous circuits with MSFSMs	89
7.2.1	PN to MSFSM transformation flow	90
7.2.2	Original models with the addition of the clock	94
7.3	Synchronous elastic circuits	98
7.4	MSFSM to PN conversion flow	98
7.4.1	MSFSM to S-component mapping	98
7.4.2	S-component merging to PN conversion	99
7.5	The role of the Wait State synchronization primitive	101
7.5.1	Issues with the MSFSM to PN conversion flow	101
7.5.2	Revised Wait State in the MSFSM to PN conversion flow	101

7.5.3 Self-loops in the PN to MSFSM conversion flow	103
7.6 A simple use case	104

Part IV Conclusion

8 Final considerations	109
8.1 Main contributions of this thesis	109
8.2 Considerations about the decomposition based on regions theory.....	109
8.3 Considerations about MSFSM model.....	110
8.4 PN decomposition: MSFSM vs. Regions Theory	111
References	115
List of Figures	119
List of Tables	123
List of Acronyms	125

Introduction and preliminaries

Introduction

In the realm of system design, verification, and synthesis, Petri nets have emerged as a pivotal model, supporting a wide range of methodologies and applications. Their graphical nature, combined with their mathematical rigor, offers a unique blend of intuitiveness and precision, making them an invaluable tool for capturing and analyzing the dynamic behavior of systems. The ability of Petri nets to represent concurrency, synchronization, and conflicts gives them a distinct advantage in modeling complex systems, especially in today's era, where parallelism and interaction are at the heart of most technological advancements.

The significance of Petri nets extends beyond providing specifications; they serve as a foundation for various algorithms and techniques aimed at ensuring the correctness, reliability, and efficiency of systems. From the early stages of design, when conceptual models are crafted, to the advanced phases of verification and synthesis, when the system's behavior is rigorously tested and optimized, Petri nets play a crucial role in guiding the process and ensuring its success.

Recognizing the profound influence and adaptability of Petri nets in these domains, I focused this thesis on the decomposition of Petri nets and transition systems. For those who are not familiar with Petri nets and transition systems, please read the chapter on preliminaries. One may ask: "Why is decomposition very important?" The reasons are numerous: e.g., simpler analysis and optimization of single components and reusability. If we start the decomposition from a transition system, the identification of parallel operations is also done automatically.

1.1 Practical application of Petri net decomposition

To better grasp the relevance of decomposition, let us delve into an example: a car-producing factory. Fig. 1.1 represents the initial sketch with a transition system representing the main flow for the production of only one car and with revenues that match the expenses for production and shipment: a comprehensive representation for a high-level description of a sequential flow. Increasing the level of detail, we have the transition system in Fig. 1.2, with the representation of different parts and materials. Here, we can see how the number of states explodes since we have some highly concurrent parts, indeed, the PN representation perfectly fits in this case (Fig. 1.3). In particular, we can avoid a huge state explosion if we adopt the model to represent

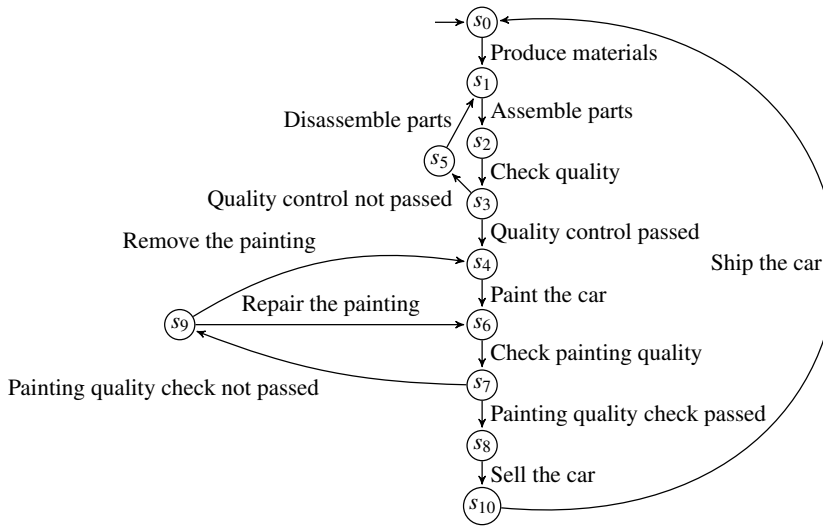


Fig. 1.1: Transition system representing a car manufacturing system with a low level of detail.

the concurrent manufacture of n cars. Thanks to the PN structure, it is sufficient to increase the number of initial tokens. If we further expand our model with new features that expand the size of the PN, like: n different types of cars, different painting colors, the allocation of earnings from car sales to reward investors and expand the factory, then the PN would become quite difficult to analyze. To address this problem of state explosion, the decomposition of the Petri net would be essential. Fig. 1.4 represents a decomposition, divided into the different views of the manufacturing process, with the possibility of easily modifying or expanding different aspects of the entire system, without having to directly understand the PN representing the overall behavior with an intricate “spaghetti” structure. The components derived from the decomposition are synchronized following the rules of parallel composition “||” of reachability graphs¹: given two Petri nets with a common label, the two PNs can evolve independently until they reach the common event, which should fire at the same time in both PNs (a formal description is provided in the chapter on preliminaries). Looking at the decomposition result in Fig. 1.4(a), we can see a simplified flow for the production and shipment of a generic car. Fig. 1.4(b) shows the parts composition of a “Model 1” car. As mentioned previously, analysis becomes easier; by reusing parts, we can add a new car model by slightly changing the PN representing the composition of the “Model 1” car, while keeping unchanged the other PNs, if we already modeled the production of the required car components. Fig. 1.4(c) shows the color configurability for each car model, isolating this aspect from the others, which could be considered as part of the user configuration experience. Figs. 1.4(d) and 1.4(e) describe the parts production processes. Fig. 1.4(f) represents the resource management aspect, integrating also the activation of “Energy supply”

¹ The reachability graph of a Petri net is a directed graph, $G = (V, E)$, where each node, $v \in V$, represents a reachable marking and each edge, $e \in E$, represents a transition between two reachable markings. The set of reachable markings can be infinite. [1]

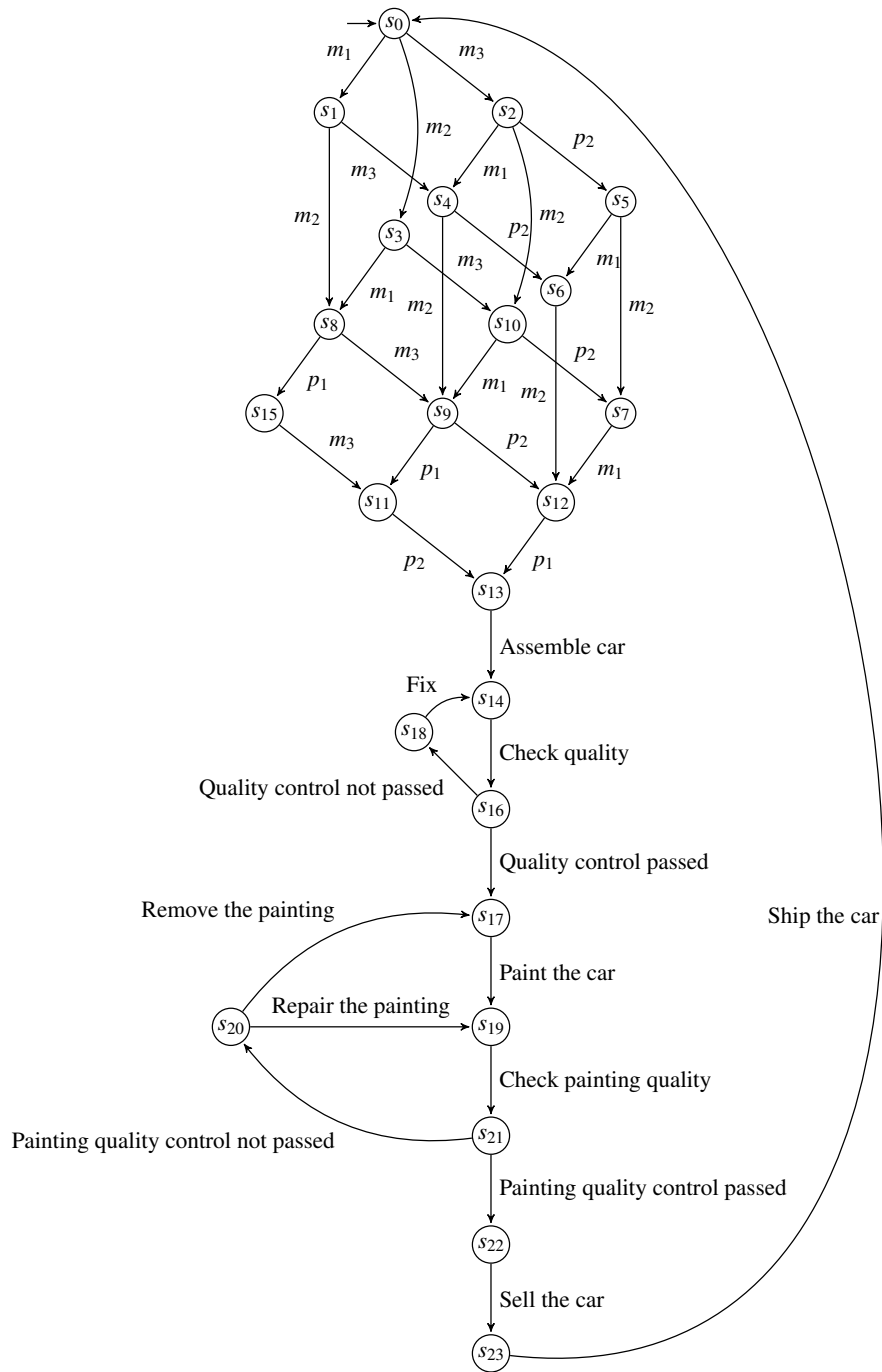


Fig. 1.2: Transition system representing a car manufacturing system with a higher level of detail, representing the production of single materials and car parts where m_1, m_2 and m_3 represent the production of three different materials and p_1 and p_2 represent the production of car parts.

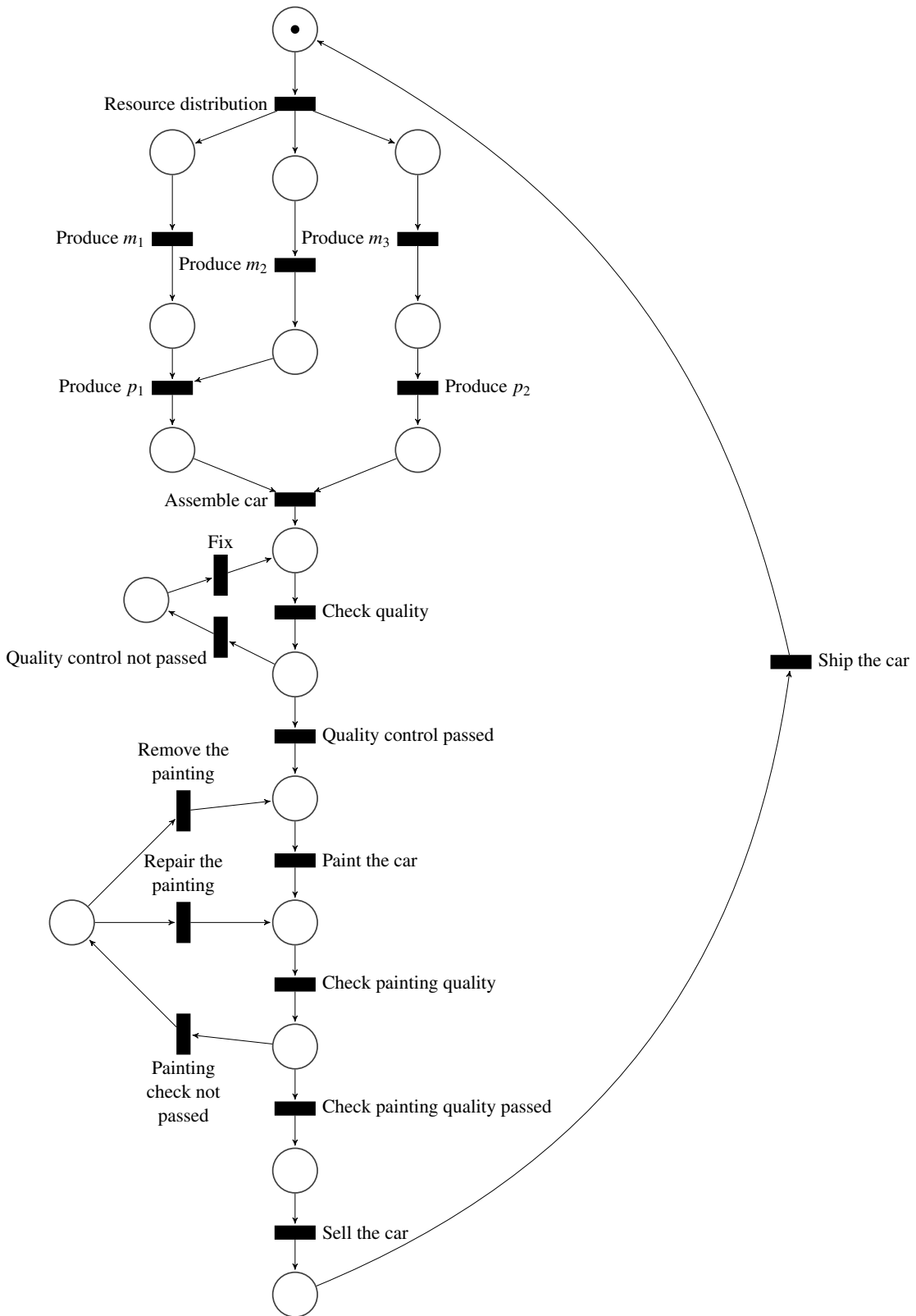


Fig. 1.3: Petri net representation of the car factory of Fig. 1.2.

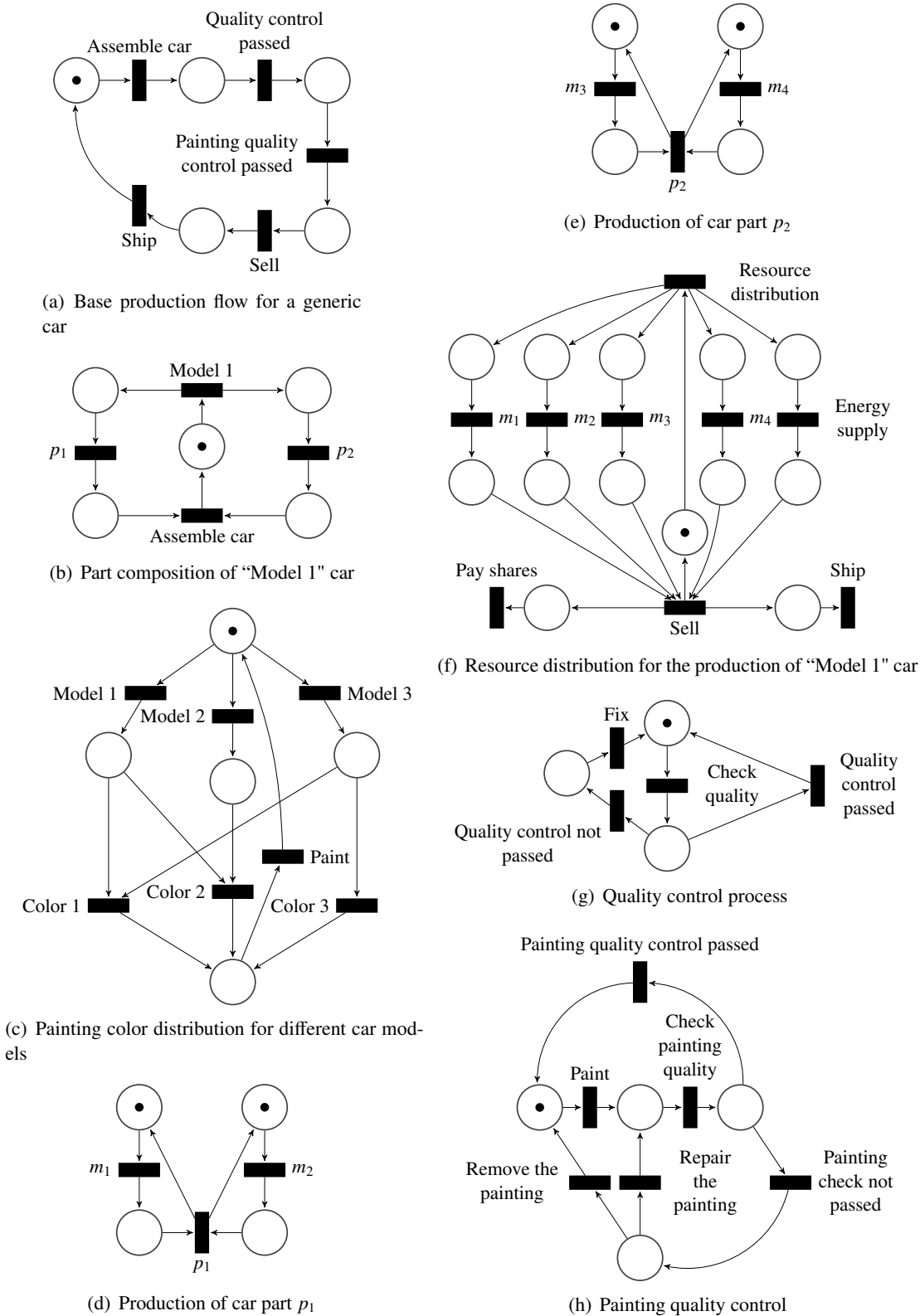


Fig. 1.4: Petri net decomposition representing an extended version of the car factory in Fig. 1.3.

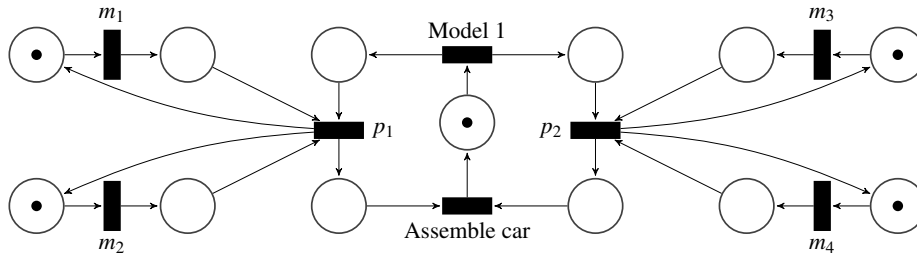


Fig. 1.5: Petri net representing the composition of PN in Figs. 1.4(b), 1.4(d) and 1.4(e).

transition, to be synchronized with an additional PN managing the energy supply in all parts of the factory. Lastly, Figs. 1.4(g) and 1.4(h) represent two quality control processes.

Looking at the decomposition result, we can easily find an analogy with relational databases: usually instead of creating a database with only one table containing everything, the information is split into different tables searching to minimize the information redundancy which is inevitable for a correct database design. PN decomposition is similar: instead of keeping a single PN containing everything, we perform a decomposition. The size of the decomposition is larger than the original PN because of the inevitable redundancy necessary to synchronize the derived components. From the other side, like in relational databases, the entire picture results more comprehensible; performing changes, and gathering information with respect to a certain aspect results easier, since we can synchronize only the PN of our interest to gather high-level information, exactly as in case of relational databases. If everything is working, a further check could be done on the PN representing the composition of all components. As long as we compose a couple of small PN, the operations could be done manually, but when all the derived components are involved, it is time to adopt a software for the verification process. Looking at Fig. 1.4 suppose that we want a detailed flow of the steps to assemble a car of “Model 1”, considering only the materials used for the car part production: it is sufficient to combine the PN of Figs. 1.4(b), 1.4(d) and 1.4(e). The result can be observed in Fig. 1.5. In this case we can appreciate the possibility to keep only the meaningful information, removing everything unnecessary in the current circumstances.

1.2 Overview

The recently discussed example underscores the critical role of the Petri net decomposition in Business Process Management (BPM)², as evidenced by multiple studies, especially those of Van Der Aalst [3–7], calling BPM the “Killer App” for Petri nets [8]. A concrete application that should benefit from PN decomposition in BPM is process mining [9–14]. The aim is to

² Business process management (BPM) is a discipline that uses various methods to discover, model, analyze, measure, improve and optimize business processes. A business process coordinates the behavior of people, systems, information, and things to produce business outcomes in support of a business strategy. Processes can be structured and repeatable, or unstructured and variable. Though not required, technologies are often used with BPM. BPM is the key to aligning IT (Information technology) / OT (Operational technology) investments with business strategy. [2]

mine comprehensive Petri nets for a better visualization of spaghetti models obtained by process mining. Creating a transition system from logs allows one to better understand a process, giving also the possibility to parallelize the latter. Say that we mined some traces and represented them as a transition system, then the decomposition of the transition system (presented in the first part of my thesis) splits the TS as different concurrent flows, which can be analyzed separately. Furthermore, since each derived PN is concurrent with the others, parallelization of concurrent processes becomes easier. In most cases, the decomposition starts from a Petri net that represents the whole behavior of the system [9–11]. Instead of creating a PN from event logs, we can easily create a transition system [15, 16] and directly decompose it with the software that I created during my Ph.D.: “Seto”³ [17]: a C++ based software, incorporating different external libraries like *networkx* [18] for graph analysis, used to calculate Maximal Independent Sets (MIS)⁴, *PBLib* [19] allowing the resolution of the Boolean satisfiability problem (SAT)⁵ constraints, *Petrify* [20] for further synthesis analysis, *mcr12* [21] for bisimulation check, and finally other utilities (e.g., for the export or import of files compatible with *graphviz*⁶ [22] open source software, used for PN/LTS visualization).

The purpose of the decomposition algorithms provided by our software is not to replace the current methods, since the decomposition result does not always give additional useful data with respect to monolithic models, but often it does. It is a new tool that should be used in synergy with established process mining techniques, without replacing them. The application of the proposed decomposition algorithms to process mining is still an open question, left as a suggestion for future research.

The minimization process has substantial importance in enhancing efficiency and clarity in various domains, particularly in process mining and circuit design. In the field of process mining, minimization of PN decompositions is crucial. Process mining inherently deals with complex, real-world data and event logs, translating them into comprehensive models. As these models often represent intricate system behaviors, they can become extremely convoluted. By minimizing PN decompositions, we can distill these complex structures into more manageable, simplified forms without losing essential information. This practice not only makes the models easier to understand and analyze, but also significantly reduces computational overhead, facilitating more efficient data processing and analysis. Additionally, minimized PNs help to identify and eliminate redundancies and inefficiencies within the system, providing clearer insight into process optimization opportunities.

However, in the realm of circuit design, the importance of minimization cannot be overstated. The design and operation of electronic circuits require high precision, efficiency, and reliability. In this thesis, the MSFSM⁷ model [23–25] was analyzed with the purpose of de-

³ Available at <https://github.com/viktorteren/Seto>.

⁴ Given an undirected graph $G = (V, E)$, an **independent set** is a subset of nodes $U \subseteq V$ such that no two nodes in U are adjacent. An independent set is maximal if no node can be added without violating independence.

⁵ Boolean Satisfiability or simply SAT is the problem of determining if a Boolean formula is satisfiable or unsatisfiable. Satisfiable: If the Boolean variables can be assigned values such that the formula turns out to be *true*, then we say that the formula is satisfiable, otherwise it is unsatisfiable, also called UNSAT.

⁶ Available at <https://graphviz.org/>.

⁷ Multiple Synchronized Finite State Machine

signing synchronous circuits. By engaging in the minimization of this structure, designers can achieve more streamlined, cost-effective, and efficient circuit designs. This reduction directly impacts on the physical space required for hardware, material costs, and energy consumption, while maintaining the functional integrity and performance of circuits. In addition, simplified models are more accessible for future troubleshooting and scalability, making them preferable in rapidly evolving technological applications.

In both contexts, the underlying goal remains consistent: to enhance the clarity, efficiency, and maintainability of complex systems. Whether by facilitating a detailed analysis of the data obtained by process mining or by optimizing the physical and functional aspects of circuit design, the minimization of PN decompositions and MSFSMs serves as a foundational strategy in managing complexity across disciplines.

This thesis is structured into two primary parts, each focusing on distinct but interconnected aspects of Petri nets and their applications. The initial segment delves into the decomposition of transition systems and Petri nets into a synchronous product of Petri nets, applying specific constraints. The constraints most frequently analyzed concern various subclasses of Petri nets, notably State Machines (SMs) and Free-Choice Petri nets (FCPNs) (see the preliminaries). The decomposition process is based on the theory of regions [26], a cornerstone of the Petri net synthesis for decades. This approach not only facilitates a versatile decomposition process, but also enables to prove the existence of a bisimulation between the synchronized Petri nets derived and the original LTS or Petri net. This method proves to be more robust compared to the relations between languages, as documented in previous studies [27].

One notable advantage of this methodology is the simplification that it brings to the visualization and analysis of highly complex Petri nets, especially Free-choice Petri nets. These nets, characterized by their structural properties, simplify the representation by capturing essential elements like causality, choice, and concurrency, thus avoiding convoluted intertwined representations. This clarity in visualization, coupled with the ability to depict more complex sequential behaviors than standard State Machines, enhances their analytical accessibility, keeping the complexity moderate since several important problems can be solved in polynomial time [28, p. 403]. In our pursuit of efficiency and minimality, the proposed flow inherently incorporates dual minimization procedures (removal of redundant components and minimization of those remaining). Although some redundancy can aid component readability, excessive repetition can clutter the representation with superfluous information. Our strategy prioritizes minimality, accepting slightly more challenging readability to avoid unnecessary redundancy.

The subsequent part of the thesis transitions to the exploitation of the MSFSM model, enabling the construction of synchronous circuits from Petri nets through the continued application of PN decomposition. Indeed, the decomposition of Petri nets is ready to make significant contributions to the fields of circuit analysis and design. The MSFSM model is designed to initiate a polynomial synthesis flow originating from Free-Choice or Asymmetric-Choice Petri nets (ACPNs)⁸. This phase of research led to the identification of an inconsistency within the established model, leading to the suggestion of a solution that currently stands as a conjecture.

⁸ For ACPN definition see the preliminaries chapter.

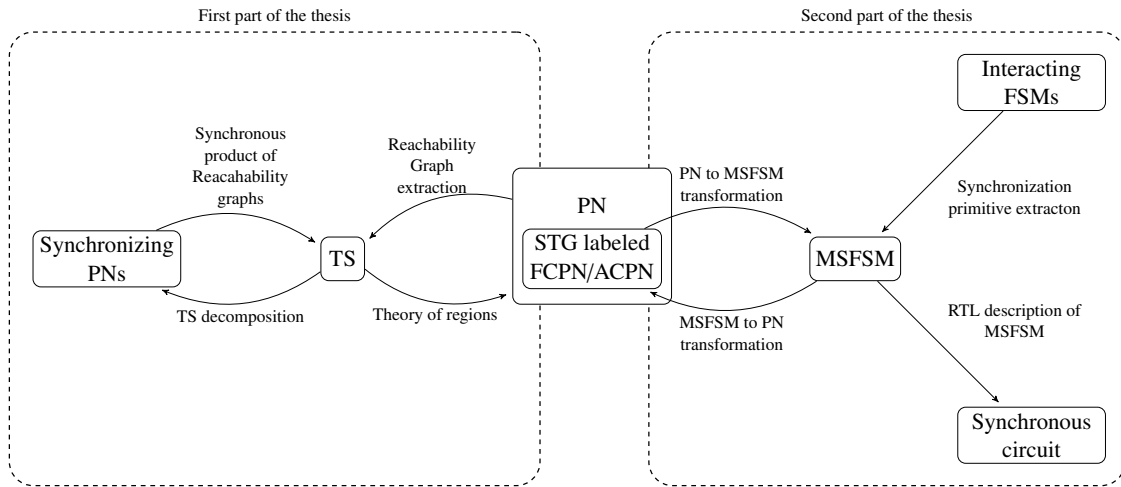


Fig. 1.6: Relationships among the models discussed in this thesis.

Complementing the text, Fig. 1.6 offers a concise visual representation of the multifaceted flows of the thesis. The left half of the picture highlights the TS decomposition, integral to the research, within a broader flow that encompasses the reachability graph extraction from a Petri net. Interestingly, the decomposition process is reversible. By executing the synchronous product of the reachability graphs of the synchronizing Petri nets, we can revert to a bisimilar transition system (potentially distinct from the original). Subsequently, the theory of regions facilitates the derivation of a singular, bisimilar Petri net. The figure's right half illustrates the two principal transformation flows pertinent to the second part of the thesis. The main flow initiates from an STG⁹ labeled ACPN or FCPN, culminating in a synchronous circuit, while the alternative flow merges a set of interacting FSMs into a coherent Petri net.

⁹ A Signal Transition Graph (STG) $G = (V, E)$ is an interpreted subset of marked graphs, i.e. Petri net such that each place has exactly one incoming and one outgoing arc, wherein each transition represents either the rising (x^+) or falling (x^-) of a signal x which has signal levels high and low. V is the set of transitions, and E is the set of edges corresponding to the places of the underlying marked graph.

Preliminaries

This chapter provides the background material for the thesis.

2.1 Finite State Machines and Transition systems

Definition 1 (FSM). An FSM, M , is a five-tuple, $M = (I, O, S, s_0, \delta, \lambda)$, where I is a finite, nonempty set of inputs, O is a finite, nonempty set of outputs, S is a finite nonempty set of states, s_0 is the initially active state, $\delta: I \times S \rightarrow S$ is the next state function, and $\lambda: I \times S \rightarrow O$ (for a Mealy machine), or $\lambda: S \rightarrow O$ (for a Moore machine) is the output function.

An example of FSM is represented in Fig. 2.1.

Definition 2 (TS/LTS [26]). A labeled transition system (LTS, or simply TS) is defined as a 4-tuple (S, E, Δ, s_0) where:

- S is a non-empty set of states,
- E is a set of events,
- $\Delta \subseteq S \times E \times S$ is a transition relation,
- $s_0 \in S$ is an initial state.

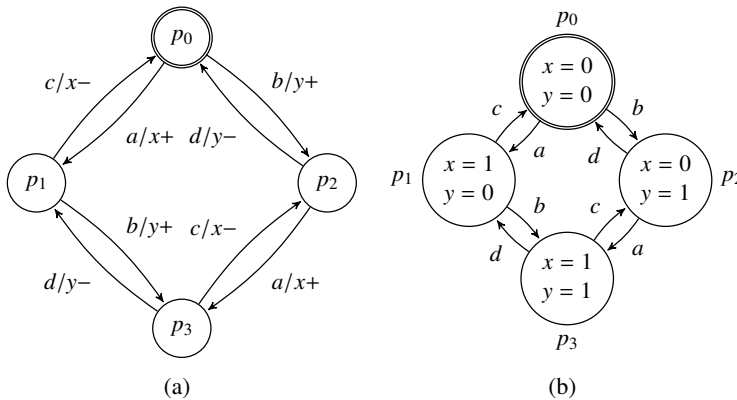


Fig. 2.1: Two different FSM representations of the same system.

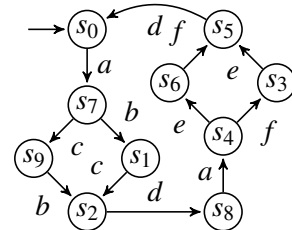


Fig. 2.2: Example of transition system.

Henceforth, $s \xrightarrow{e} s'$ will denote the fact that $(s, e, s') \in \mathcal{A}$. Every transition system is assumed to meet the following properties:

- Self-loops are not admitted, i.e., $s \xrightarrow{e} s' \implies s \neq s'$.
- Each event has at least one occurrence, i.e., $\forall e \in E : \exists s \xrightarrow{e} s'$.
- Every state $s \in S$ is reachable from the initial state, i.e., there is a sequence $s_0 \xrightarrow{e_1} s_1, s_1 \xrightarrow{e_2} s_2, \dots, s_{n-1} \xrightarrow{e_n} s_n$ such that $s_n = s$.
- It is deterministic, i.e., $s \xrightarrow{e} s', s \xrightarrow{e} s'' \implies s' = s''$.

An example of a transition system can be seen in Fig. 2.2

We denote the initial state of FSMs by a double circle (Fig. 2.1) and the initial state of transition systems by a pointing arrow (Fig. 2.2).

Definition 3 (Isomorphism). *Two transition systems $TS_1 = (S_1, E, T_1, s_{0,1})$ and $TS_2 = (S_2, E, T_2, s_{0,2})$ are said to be isomorphic (i.e., there is an isomorphism between TS_1 and TS_2) if there is a bijection $b_S : S_1 \rightarrow S_2$, such that:*

- $b_S(s_{0,1}) = s_{0,2}$,
- $\forall (s, e, s') \in T_1 : (b_S(s), e, b_S(s')) \in T_2$,
- $\forall (s, e, s') \in T_2 : (b_S^{-1}(s), e, b_S^{-1}(s')) \in T_1$.

Definition 4 (Bisimulation). *Given two transition systems $TS_1 = (S_1, E, T_1, s_{0,1})$ and $TS_2 = (S_2, E, T_2, s_{0,2})$, a binary relation $B \subseteq S_1 \times S_2$ is a bisimulation, denoted by $TS_1 \sim_B TS_2$, if $(s_{0,1}, s_{0,2}) \in B$ and if whenever $(p, q) \in B$:*

- $\forall (p, e, p') \in T_1 : \exists q' \in S_2$ such that $(q, e, q') \in T_2$ and $(p', q') \in B$,
- $\forall (q, e, q') \in T_2 : \exists p' \in S_1$ such that $(p, e, p') \in T_1$ and $(p', q') \in B$.

Two TSs are said to be bisimilar if there is a bisimulation between them.

The operation Ac removes from a TS all the states that are not reachable or accessible from the initial state and all the transitions attached to them.

Definition 5 (Synchronous product). *Given two transition systems $TS_1 = (S_1, E_1, T_1, s_{0,1})$ and $TS_2 = (S_2, E_2, T_2, s_{0,2})$, the synchronous product is defined as $TS_1 || TS_2 = Ac(S, E_1 \cup E_2, T, (s_{0,1}, s_{0,2}))$ where $S \subseteq S_1 \times S_2$, $(s_{0,1}, s_{0,2}) \in S$, $T \subseteq (S_1 \times S_2) \times E \times (S_1 \times S_2)$ is defined as follows:*

- if $a \in E_1 \cap E_2$, $(s_1, a, s'_1) \in T_1$ and $(s_2, a, s'_2) \in T_2$ then $((s_1, s_2), a, (s'_1, s'_2)) \in T$,
- if $a \in E_1$, $a \notin E_2$ and $(s_1, a, s'_1) \in T_1$ then $((s_1, s_2), a, (s'_1, s_2)) \in T$,
- if $a \notin E_1$, $a \in E_2$ and $(s_2, a, s'_2) \in T_2$ then $((s_1, s_2), a, (s_1, s'_2)) \in T$,
- nothing else belongs to T .

The synchronous product is associative, so we can define the product of a collection of n TSs: $TS_1 || TS_2 || \dots || TS_n = ((TS_1 || TS_2) \dots) || TS_n$; as an alternative, we can directly extend the previous definition to more than two TSs.

An example of a synchronous product is shown in Fig. 2.3.

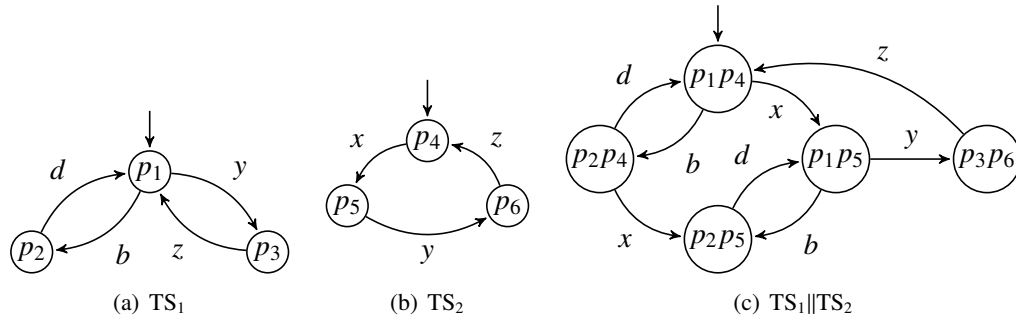


Fig. 2.3: Example of synchronous product.

2.2 Petri Nets and Signal Transition Graphs

This section introduces the relevant terminology on Petri nets. For a deeper insight into them, we refer to [29].

Definition 6 (Ordinary Petri net [29]). An ordinary Petri net is a 4-tuple, $N = (P, T, F, M_0)$ where:

- $P = \{p_1, p_2, \dots, p_m\}$ is a finite set of places,
- $T = \{t_1, t_2, \dots, t_n\}$ is a finite set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
- $M_0: P \rightarrow \{0, 1, 2, 3, \dots\}$ is an initial marking,
- $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

The previous definition corresponds to what is known as *ordinary* Petri net (no weights on the arcs) [29]. In this thesis, all Petri nets are assumed to be ordinary.

Definition 7 (Pre-set/post-set). For any $x \in P \cup T$, $\bullet x = \{y \mid (y, x) \in F\}$ is called the pre-set of x , and $x^\bullet = \{y \mid (x, y) \in F\}$, is called the post-set of x .

Definition 8 (Firing rule). Let $N = (P, T, F, M_0)$ be a Petri net. A transition $t \in T$ is enabled in marking M , represented as $M[t]$, if $M(p) > 0, \forall p \in \bullet t$. If t is enabled in M , then t can be fired

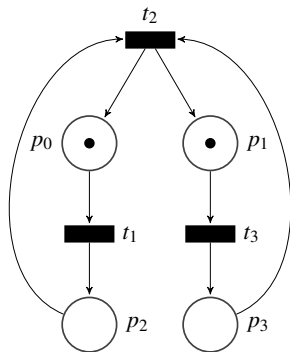


Fig. 2.4: Petri net example

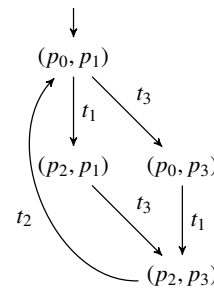


Fig. 2.5: Reachability graph of Petri net in Fig. 2.4.

leading to another marking M' , denoted as $M[t]M'$, such that:

$$M'(p) = \begin{cases} M(p) - 1 & \text{if } p \in \bullet t \setminus t^\bullet \\ M(p) + 1 & \text{if } p \in t^\bullet \setminus \bullet t \\ M(p) & \text{otherwise} \end{cases}$$

We call $[M]$ the set of reachable markings from M by firing sequences of enabled transitions.

Definition 9 (Bounded Petri net). A Petri net $N = (P, T, F, M_0)$ is said to be k -bounded or simply bounded if the number of tokens in each place does not exceed a finite number k for any marking reachable from M_0 , i.e., $M(p) \leq k$ for every place p and every marking $M \in [M_0]$.

Definition 10 (Safe Petri net). A Petri net $N = (P, T, F, M_0)$ is said to be safe if it is 1-bounded.

Definition 11 (Reachability graph [30, p. 20]). Given a safe Petri net $N = (P, T, F, M_0)$, the reachability graph of N is the transition system $RG(N) = ([M_0], T, \Delta, M_0)$ defined by $(M, t, M') \in \Delta$ if $M \in [M_0]$ and $M[t]M'$.

Definition 12 (State Machine, SM [29]). A state machine is a Petri net $N = (P, T, F, M_0)$ such that each transition $t \in T$ has exactly one incoming and one outgoing arc, i.e., $|\bullet t| = |t^\bullet| = 1$.

An SM with only one token in the initial marking cannot model concurrency, but it can model choice.

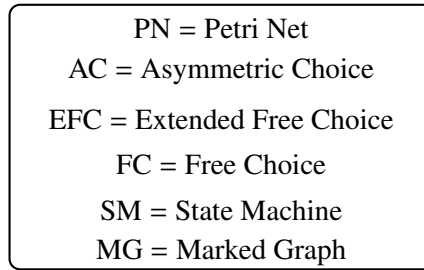
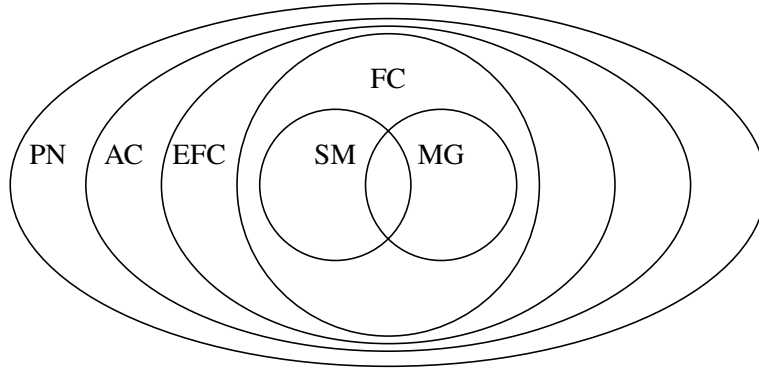


Fig. 2.6: PN hierarchy.

It has been observed in [30, p. 49] that a state machine $M = (P, T, F, M_0)$ can be interpreted as a transition system $TS = (P, T, \Delta, s_0)$, where the places correspond to the states, the transitions to the events, s_0 corresponds to the unique marked initial place, and $(p, t, p') \in \Delta$ iff $\bullet t = \{p\}$ and $t^\bullet = \{p'\}$ (in a SM by definition $|\bullet t| = |t^\bullet| = 1$). Therefore the reachability graph of M is isomorphic to the transition system TS , i.e., $RG(M)$ is isomorphic to TS .

Definition 13 (Marked Graph, MG (or Event Graph, EG) [29]). A *Marked Graph* is a Petri net $N = (P, T, F, M_0)$ such that each place $p \in P$ has exactly one incoming and one outgoing arc, i.e., $|\bullet p| = |p^\bullet| = 1$.

Definition 14 (Free-Choice Petri net, FCPN [29]). A *Free-Choice Petri net* is an ordinary Petri net $N = (P, T, F, M_0)$ such that every arc from a place is either a unique outgoing arc or a unique incoming arc to a transition, i.e.,

for all $p \in P$, $|p^\bullet| \leq 1$ or $\bullet(p^\bullet) = \{p\}$; equivalently,
for all $p_1, p_2 \in P$, $p_1^\bullet \cap p_2^\bullet \neq \emptyset \Rightarrow |p_1^\bullet| = |p_2^\bullet| = 1$.

Definition 15 (Extended Free-Choice Petri net, EFCPN [29]). An *Extended Free-Choice Petri net* is an ordinary Petri net $N = (P, T, F, M_0)$ such that given a transition in the post-set of two different places both places have the same post-set, i.e.,

for all $p_1, p_2 \in P$, $p_1^\bullet \cap p_2^\bullet \neq \emptyset \Rightarrow p_1^\bullet = p_2^\bullet$.

Definition 16 (Asymmetric-choice Petri net, ACPN [29]). An *Asymmetric-choice Petri net* is an ordinary Petri net $N = (P, T, F, M_0)$ such that for every pair of places $p_1, p_2 \in P$ with at least one common transition in their post-sets, one of the post-sets contains the other i.e.,

for all $p_1, p_2 \in P$, $p_1^\bullet \cap p_2^\bullet \neq \emptyset \Rightarrow p_1^\bullet \subseteq p_2^\bullet$ or $p_1^\bullet \supseteq p_2^\bullet$.

Definition 17 (Saturated Petri net). A PN is called *saturated net*, if no place can be added without changing the reachability space.

Definition 18 (Place-irredundant net). A Petri net $N = (P, T, F, M_0)$ is called *place-irredundant*, if for each $N' = (P', T', F', M'_0)$ such that $P' \subseteq P, T' \subseteq T, F' \subseteq F, M'_0 \subseteq M_0$ and $N' \neq N$, $RG(N') \not\sim_B RG(N)$, i.e. no place can be removed from N without losing the bisimilarity of the reachability graph.

Definition 19 (Place-minimal net). A Petri net is *place-minimal* if any other bisimilar PN contains a greater or equal number of places.

Definition 20 (Siphon and Trap). Let $N = (P, T, F, M_0)$ be a net. A set of places of a net N , $D \subseteq P$, is a *siphon (trap)*, if and only if $D \neq \emptyset \wedge \bullet D \subseteq D^\bullet$ ($D \neq \emptyset \wedge D^\bullet \subseteq \bullet D$). A siphon D (trap) is *minimal* if and only if there exists no siphon or trap D' such that $D' \subseteq D$.

Definition 21 (Strongly connected State Machine). Let $N = (P, T, F, M_0)$ be a State Machine. It is *strongly connected* if every place is reachable from every other place of the SM.

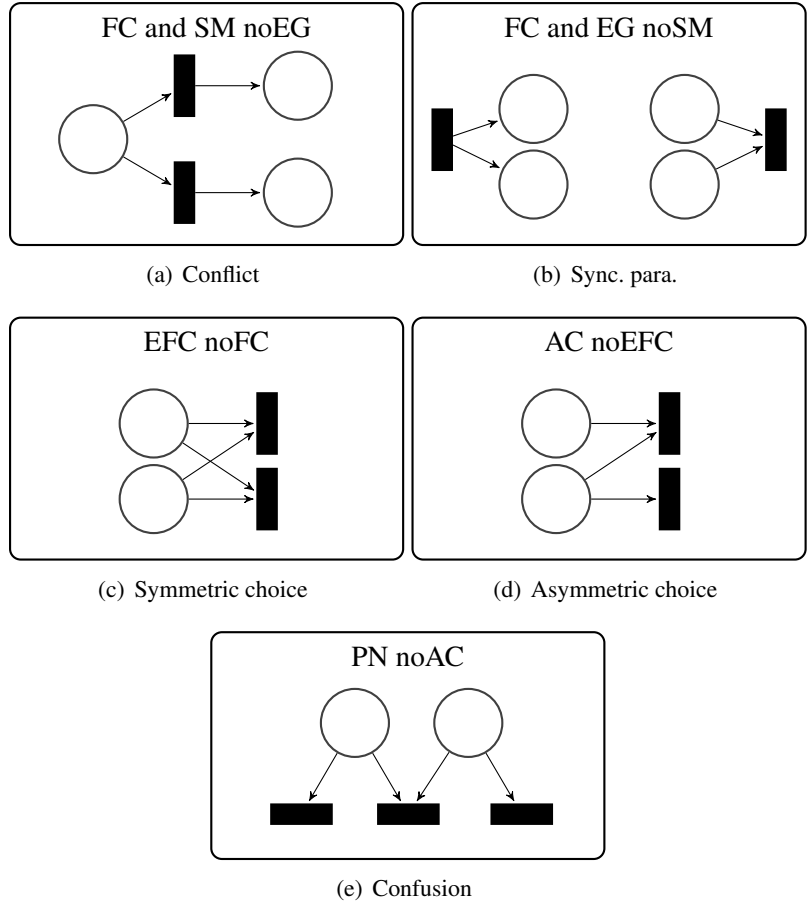


Fig. 2.7: Modelling power of different Petri Net classes [31].

Definition 22 (S-component). Let $N' = (P', T', F', M'_0)$ be a subnet of a net $N = (P, T, F, M_0)$. N' is an S-component of N if and only if $T' = \bullet P' \cap P' \bullet$ and N' is a strongly connected State Machine.

Definition 23 (Choice place and return from choice place). A place p is a choice place if $|p \bullet| > 1$. A place p' is a return from choice place if $|\bullet p'| > 1$. If both properties hold, the place is defined as a choice-return from choice place.

Theorem 1 (Commoner’s Theorem). Every proper siphon of a Free-choice system includes an initially marked trap if and only if the system is live.

Theorem 2 (Commoner’s Theorem for Asymmetric-choice systems). If every proper siphon of an asymmetric-choice system includes an initially marked trap, then the system is live.

Theorem 3 (S-Coverability Theorem). Well-formed nets are covered by S-components.

Definition 24 (Signal Transition Graph, STG). A Signal Transition Graph (STG) $G = (V, E)$ is an interpreted subset of marked graphs wherein each transition represents either the rising

($x+$ or x^+) or falling ($x-$ or x^-) of a signal x which has signal levels high and low. V is the set of transitions and E is the set of edges corresponding to places of the underlying marked graph.

2.3 Region theory and Petri net synthesis

In this section we survey the existing theory of regions [26] on which the synthesis tool Petriify [20] and a substantial part of the presented work is based.

2.3.1 Definitions and conversion flow overview

Definition 25 (Region). Given a $TS = (S, E, T, s_0)$ a region is defined as a set of states $r \subseteq S$ such that for each event $e \in E$ the following properties hold:

$$\text{enter}(e, r) \implies \neg \text{in}(e, r) \wedge \neg \text{out}(e, r) \wedge \neg \text{exit}(e, r),$$

$$\text{exit}(e, r) \implies \neg \text{in}(e, r) \wedge \neg \text{out}(e, r) \wedge \text{enter}(e, r),$$

$$\text{no_cross}(e, r) \implies \neg \text{enter}(e, r) \wedge \neg \text{exit}(e, r),$$

where:

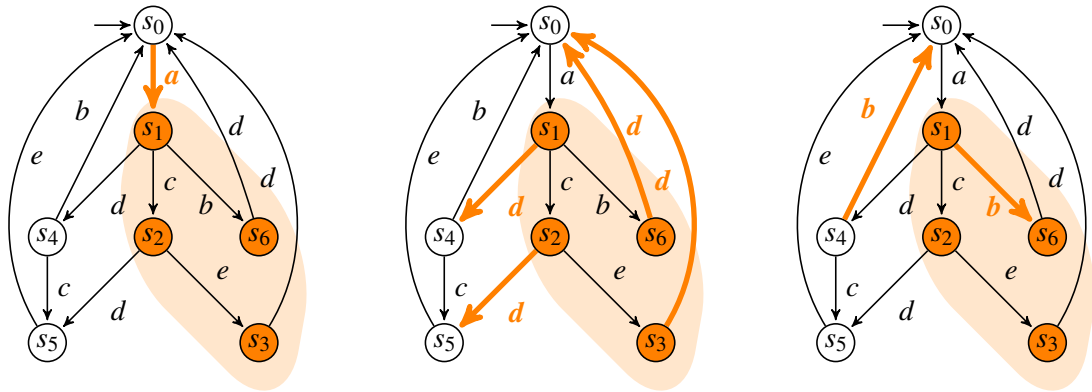
$$\text{in}(e, r) \equiv \exists (s, e, s') \in T : s, s' \in r,$$

$$\text{out}(e, r) \equiv \exists (s, e, s') \in T : s, s' \notin r,$$

$$\text{enter}(e, r) \equiv \exists (s, e, s') \in T : s \notin r \wedge s' \in r,$$

$$\text{exit}(e, r) \equiv \exists (s, e, s') \in T : s \in r \wedge s' \notin r,$$

$$\text{no_cross}(e, r) \equiv \text{in}(e, r) \vee \text{out}(e, r).$$



(a) *enter* property

(b) *exit* property

(c) *no_cross* property

Fig. 2.8: Example of *enter* property for event a , *exit* property for event d and *no_cross* property for event b respect to the highlighted region.

Fig. 2.8 shows an example with *enter*, *exit* and *no_cross* properties.

Table 2.1: Minimal regions of the TS in Fig. 2.2.

Region States of the TS	
r_1	$\{s_0, s_8\}$
r_2	$\{s_0, s_1, s_3, s_5, s_7\}$
r_3	$\{s_0, s_5, s_6, s_7, s_9\}$
r_4	$\{s_1, s_2, s_3, s_4, s_8\}$
r_5	$\{s_1, s_2, s_3, s_5\}$
r_6	$\{s_1, s_4, s_6, s_7\}$
r_7	$\{s_2, s_4, s_6, s_8, s_9\}$
r_8	$\{s_2, s_5, s_6, s_9\}$
r_9	$\{s_3, s_4, s_7, s_9\}$
r_{10}	$\{s_0, s_1, s_5, s_6, s_7\}$
r_{11}	$\{s_0, s_3, s_5, s_7, s_9\}$
r_{12}	$\{s_1, s_2, s_4, s_6, s_8\}$
r_{13}	$\{s_1, s_2, s_5, s_6\}$
r_{14}	$\{s_1, s_3, s_4, s_7\}$
r_{15}	$\{s_2, s_3, s_4, s_8, s_9\}$
r_{16}	$\{s_2, s_3, s_5, s_9\}$
r_{17}	$\{s_4, s_6, s_7, s_9\}$

Table 2.2: Pre-regions and ESs for each event of the TS in Fig. 2.2.

Event	Pre-regions	ES(event)
a	$\{r_1\}$	$\{s_0, s_8\}$
b	$\{r_3, r_9, r_{11}, r_{17}\}$	$\{s_7, s_9\}$
c	$\{r_2, r_6, r_{10}, r_{14}\}$	$\{s_1, s_7\}$
d	$\{r_5, r_8, r_{13}, r_{16}\}$	$\{s_2, s_5\}$
e	$\{r_4, r_9, r_{14}, r_{15}\}$	$\{s_3, s_4\}$
f	$\{r_6, r_7, r_{12}, r_{17}\}$	$\{s_4, s_6\}$

Definition 26 (Minimal region). A region r is called minimal if there is no other region r' strictly contained in r ($\nexists r' \mid r' \subset r$).

The minimal regions of the TS in Fig. 2.2 are shown in Table 2.1.

Definition 27 (Pre-region (Post-region)). A region r is a pre-region (post-region) of an event e if there is a transition labeled with e which exits from r (enters into r). The set of all pre-regions (post-regions) of the event e is denoted by ${}^\circ e$ (e°).

By definition if $r \in {}^\circ e$ ($r \in e^\circ$) all the transitions labeled with e are exiting from r (entering into r), furthermore, if the transition system is strongly connected, all the regions are also pre-regions of some event.

Definition 28 (Excitation set / Switching set). The excitation (switching) set of event e , $ES(e)$ ($SS(e)$), is the maximal set of states such that for every $s \in ES(e)$ ($s \in SS(e)$) there is a transition $t \in T$ such that $t = (s', e, s)$ ($t = (s, e, s')$).

The excitation sets of the TS in Fig. 2.2 are reported in Table 2.2.

Definition 29 (Excitation-closed transition system (ECTS)). A TS with the set of labels E and the pre-regions ${}^\circ e$ is an ECTS if the following conditions are satisfied:

- Excitation-closure: $\forall e \in E : \bigcap_{r \in {}^\circ e} r = ES(e)$
- Event effectiveness: $\forall e \in E : {}^\circ e \neq \emptyset$

The EC property also ensures that if two states s_1 and s_2 cannot be separated by any region, i.e., there is no minimal region r such that $s_1 \in r$ and $s_2 \notin r$, then s_1 and s_2 are bisimilar.

If the initial TS does not satisfy the excitation-closure (EC) or event effectiveness property, label splitting (Sec. 2.3.4) can be performed to obtain an ECTS.

2.3.2 Petri net synthesis

Definition 30 (Minimal Saturated Petri net). *The Petri net derived from all minimal regions is called Minimal Saturated PN.*

After having found from the ECTS the minimal pre-regions, a minimal saturated PN is obtained using the following steps:

- Each minimal pre-region becomes a place
- For each event e a transition labeled with e is created
- Tokens in the initial marking lie in places formed by regions that contain initial states of ECTS.
- There is a flow relation (r, e) from a place r to a transition e if r belongs to pre-regions of e ($r \in {}^\circ e$) and there is a flow (e, r) if r belongs to post-regions of e ($r \in e^\circ$).

The removal of the redundant regions guarantees to get a place-irredundant PN; then a quasi-place-minimal PN is obtained from an irredundant one by the greedy algorithm that merges some disjoint minimal pre-regions into a minimal set of non-minimal ones.

The basic synthesis method can be easily adapted in order to guarantee different classes of Petri nets as follows:

- Free-choice net: this property can be enforced by splitting labels until all choice regions become the only pre-regions of their post-events.
- Asymmetric-choice net: the method to enforce this property is the same as for the Free-choice nets.
- State-machine decomposable net: these nets are those that are decomposable into S-components, in which any transition of the subnet has only one predecessor place and one successor place. This approach is based on the fact that a partition of the set of states into regions corresponds to an S-component in the Petri net [32].

Next, the main steps of the synthesis are described more in-depth.

2.3.3 Minimal pre-regions generation

In the first phase of the algorithm the excitation set is the starting set to expand in order to find all regions for one event, i.e to find the subset of all states for which all the transitions labeled with

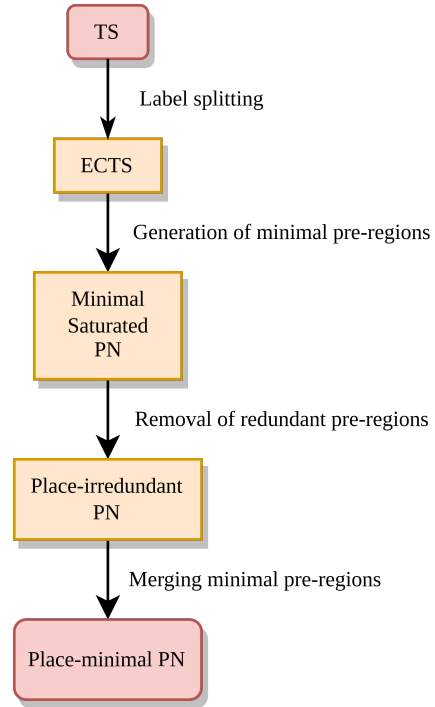


Fig. 2.9: Framework for Petri net synthesis

the same event will have the same property (*enter/exit/no_cross*). Starting from the excitation set of each event, a tree is created consisting of all possible expansions that *legalise* a property at each level, for an event chosen among those *in violation*, until the regions are obtained in the leaves.

For each set of states S that belongs to the expansion tree, one or more events can violate the aforementioned properties. Table 2.3 shows the combinations of different properties and which expansion could be made. N.B., the combination *in-out* does not appear since it does not represent a violation and corresponds to the *no_cross* property.

In order to ensure a decrease in the number of tree branches, if among all events there is one for which the only possible expansion is *no_cross*, it is chosen first for *legalization*. Since the search is monotonic, i.e each “father” node is a subset of the “child” one, once a region is found, the research on that branch is cut off. In fact, by continuing the expansion of a minimal region, it is not possible to find a minimal region again.

Furthermore, if a set of states candidate for becoming a region is explored for an event a and it also appears in the exploration tree of another event b , the expansion in that branch is not continued, since the region created from it is already known.

Once the expansion trees of all the events are completed and the regions on the leaves are found, the minimum pre-regions ($R = \{r \mid r \in \circ a\}$) are calculated, starting from the non-banal regions (the empty region and the one that consists of all states are ruled out).

Fig. 2.10 shows an example with the expansion tree for event b of TS in Fig. 2.2. For each level except the leaves, we show the events in violation and the type of violation. For leaves, instead, we show the name of the resultant region visible in Table 2.1. The label on each edge connecting two different levels of the tree explains how a violation was resolved, and in case of multiple events in violation, the involved event is specified. In this example, we can see two interesting facts:

- solving the violation for an event could solve further violations even if it is not always the case: when the violation involves both events e and f , the resolution of the violation for one event solves also the violation for the other event;
- the expansion tree produces minimal regions for this tree, therefore continuing the search would produce non-minimal regions, but it does not guarantee that the obtained regions are minimal with respect to all possible obtainable regions: regions $\{s_0, s_3, s_4, s_7, s_8, s_9\}$ and $\{s_0, s_4, s_6, s_7, s_8, s_9\}$ are minimal for the expansion tree of b but they are not minimal with respect to the region $r_1 = ES(a) = \{s_0, s_8\}$, furthermore it is possible to see that even without the aforementioned regions excitation-closure for event b is still achievable, indeed $r_3 \cap r_9 \cap r_{11} \cap r_{17} = ES(b)$.

Table 2.3: Possible expansions given a set of states with a combination of properties respect to a given event.

Combination of properties	Possible expansions
<i>exit-enter</i>	<i>no_cross</i>
<i>exit-in</i>	<i>no_cross</i>
<i>exit-out</i>	<i>no_cross</i> or <i>exit</i>
<i>enter-in</i>	<i>no_cross</i>
<i>enter-out</i>	<i>no_cross</i> or <i>enter</i>

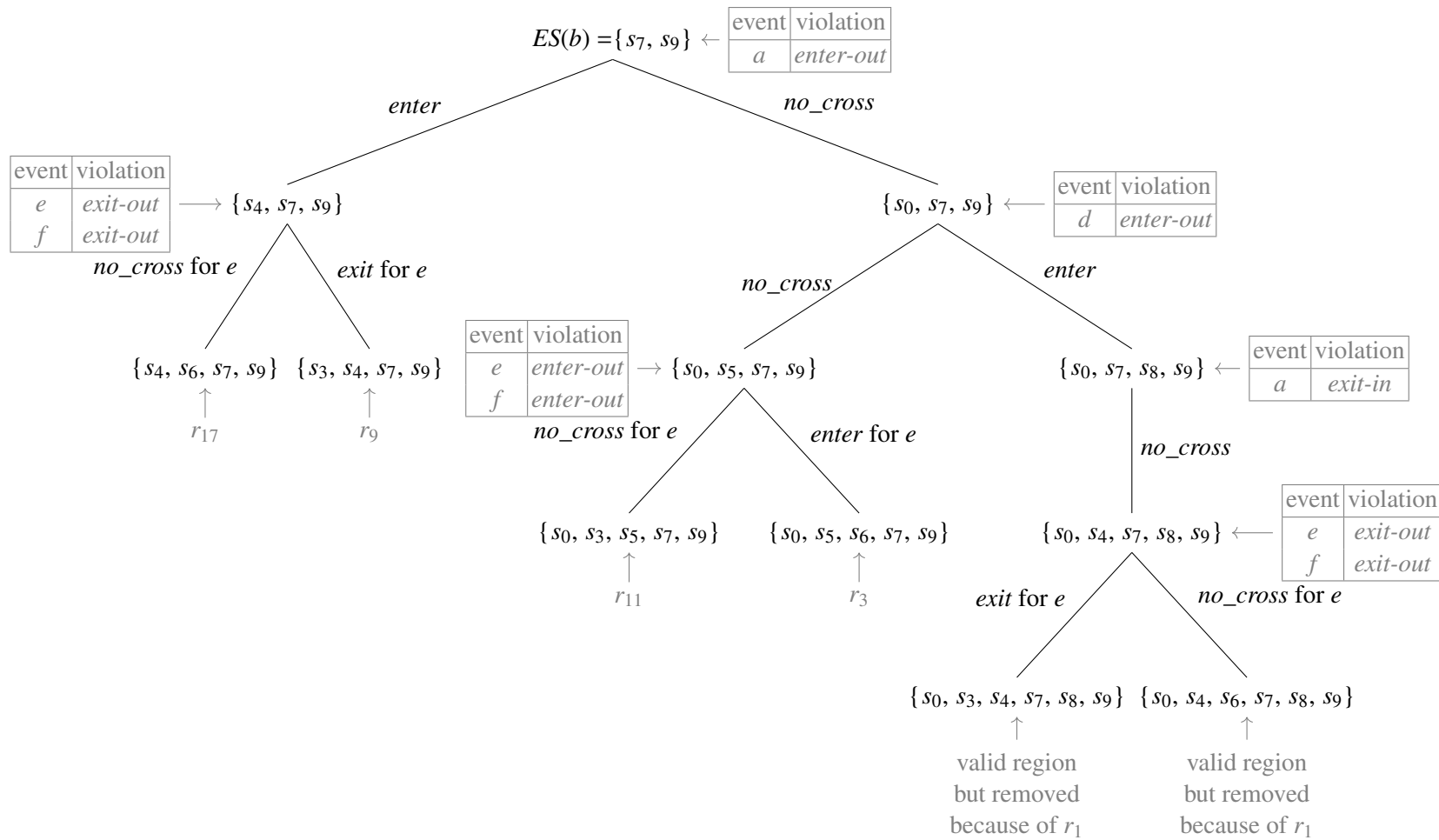


Fig. 2.10: Example of expansion tree for event b of TS in Fig. 2.2.

Definition 31 (Minimal pre-region). A region r is a pre-region for an event e if there exists an outgoing transition (of type exit) labeled with e from r . It is also minimal if there does not exist another region r' that is a sub-region of r .

2.3.4 Label Splitting

After computing the set of minimal pre-regions, for each event it is necessary to verify that the following condition of *excitation closure* holds:

$$\forall a : \bigcap_{r \in \circ a} r = ES(a) \wedge \circ a \neq \emptyset$$

In order to produce an excitation closed transition system (ECTS), if the excitation closure property does not hold, *label splitting* must be applied. The strategy is to split labels to turn a set of states into a region. A candidate region is chosen from sets of states that belong to the expansion tree for each event. The attention is focused on sets S' such that:

$$ES(a) \subseteq S' \subset \bigcap_{r \in \circ a} r$$

From each set of states of S' the one with fewer events that violate one of the region conditions is selected. If more states have the same number of “in violation” events, the smallest is selected. The latter set of states is forced to be a region by informally splitting those events that do not satisfy the region conditions.

Once events are split, the steps of region computation and *label splitting* are iterated until convergence, that is, until the *excitation closure* is satisfied, generating a split-morphic ECTS compared to the initial TS.

At every step, completely new regions can be produced, in most cases *label splitting* involves only the separation of the precalculated regions: if for instance there was a region $r_1 = \{s_1, s_2, s_3, s_4\}$ and a candidate region $s' = \{s_1, s_2\}$, as a result, regions $r_1 = \{s_1, s_2\}$ and $r'_1 = \{s_3, s_4\}$ would be generated.

The worst case is when *label splitting* is performed on all events, and so every region contains only one state. Therefore, the final Petri net cannot have fewer places than the original transition system.

An example of label splitting can be seen in Fig. 2.11: the initial TS has two regions $r_1 = \{s_0, s_1, s_2\}$ and $r_2 = \{s_3\}$. Label a satisfies the *no-cross* property, and so it is not an ECTS, because e.g., event effectiveness is not satisfied for the event a : $\circ a = \emptyset$. Also excitation-closure is not satisfied for the event b : $\bigcap_{r \in \circ b} r = r_1 \neq ES(b)$. After label splitting, label a is split into a and a' yielding the following smaller minimal regions: $r_0 = \{s_0\}$, $r_1 = \{s_1\}$, $r_2 = \{s_2\}$ and $r_3 = \{s_3\}$. After label splitting, both excitation-closure and event effectiveness are satisfied.

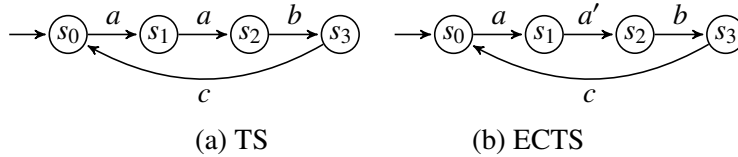


Fig. 2.11: TS before label splitting (a) and ECTS after label splitting (b).

N.B., during the PN synthesis process, if label splitting is not performed for each distinct ECTS event, a transition with the same event is created, therefore, the resultant minimal saturated PN also has the minimal number of events, and, furthermore, any derived PN has only one occurrence of each event. In case of label splitting, usually the splitting algorithm does not ensure the minimal number of resulting labels since an approximate approach is used, it is possible to perform the exhaustive search but it would be too complex. Consequently, the Petri net obtained may not have the fewest possible number of transitions, as this would equate to the number of unique labels produced post-label splitting. In this case, the PN would have at least one event with multiple occurrences, but each instance of that would have a unique label, e.g. $e/1$ and $e/2$. Fig. 2.12 presents a comparison between a Petri net obtained through optimal label splitting and one resulting from a suboptimal procedure. This example highlights the redundancy in Petri net's transitions, where the number of transitions directly correlates with the number of derived labels, leading to an unnecessary additional transition $c/1$ in Fig. 2.12(d).

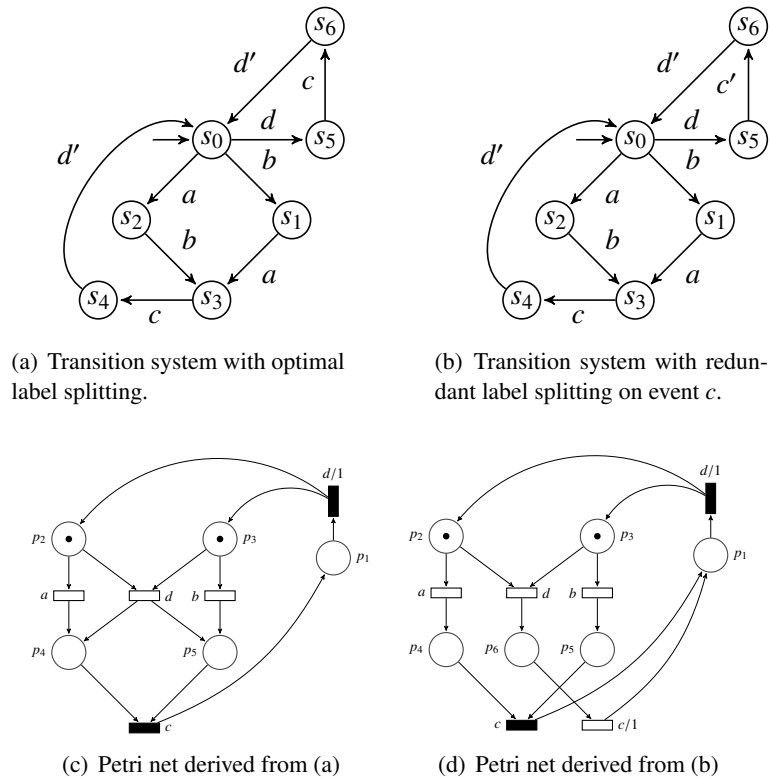


Fig. 2.12: Example showing the different Petri nets derived from optimal and sub-optimal label splitting.

2.3.5 Redundant pre-regions removal

In order to remove redundant regions, as a preliminary step, essential regions are found. Subsequently, the minimum set of non-essential and *irredundant* regions is found and the regions that are not part of the two cited groups are removed.

Definition 32 (Essential region). A region r is essential if there exists a state s and an event e such that $r \in \circ e$, $s \in r$ and $\forall r' \in \circ e \mid r' \neq r$, we have $s \notin r'$, i.e., if r is the only region that covers one or more states for an event, its removal leads to the violation of the excitation-closure property.

Definition 33. A set of regions R is called redundant if there is a region $r \in R$ such that $R - r$ still satisfies the excitation closure condition. Otherwise, R is called irredundant.

The search for the irredundant set of regions aims to find minimal covers of complete states that also satisfy the *excitation closure* property. Finding a minimum cost cover is equivalent to finding a minimum-cost solution of a Boolean equation describing the covering conditions, assigning to each region a cost, depending on the application. For instance, to minimize the number of places in the resultant Petri net, one may assign to each region a cost equal to the number of states belonging to it.

2.3.6 Minimal pre-regions merging

In order to generate a place-minimal Petri net, it is necessary to merge at least once two pre-regions. The algorithm consists in an exhaustive search of pairs of disjoint pre-regions, checking if merging these regions the *excitation-closure* property remains satisfied. When such a pair is found, a new region, formed by the union of those that belong to it, replaces the old ones. To find the optimal solution, all possible irredundant Petri nets must be computed as the starting point for merging. To save computing time, a suboptimal solution may be found. Merging minimal pre-regions allows merging places in the corresponding irredundant Petri net.

2.4 Binary Decision Diagram (BDD)

Given a function, for example $f = a * \bar{b} + c$, this function can be represented by a truth table (see Table 2.13). If we read the truth table, each row represents a set of assignments of *true* or *false* value to each variable, in our case a , b , and c . The same decisions can be represented graphically by a Binary Decision Tree (BDT) where each level of the tree represents one of the variables and the leaves represent the final assignment result. There are two output edges for each node: the first represents the false assignment of the variable, and the edge is usually dashed. The second represents the true assignment of the same variable (see Fig. 2.14).

By simplifying the Binary Decision Tree, a graph can be created: Binary Decision Diagram (BDD). Usually, Reduced-Ordered Binary Decision Diagrams (ROBDD) are used. In particular, a BDD is “Ordered” if the variables appear in the same order on all paths from the root and the BDD is “Reduced” if all isomorphic subgraphs are merged and nodes with isomorphic children are removed. The peculiarity of ROBDDs is their uniqueness, given a function and a variable order. N. B.: the order of the variables changes the resultant ROBDD, indeed, the BDDs in Figs. 2.15 and 2.16 are completely different even if both represent the same function.

In this chapter, preliminary information was provided including fundamental models used throughout the thesis, such as Petri nets and transition systems. Some key concepts regarding the

Fig. 2.13: Truth table for the function f .

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

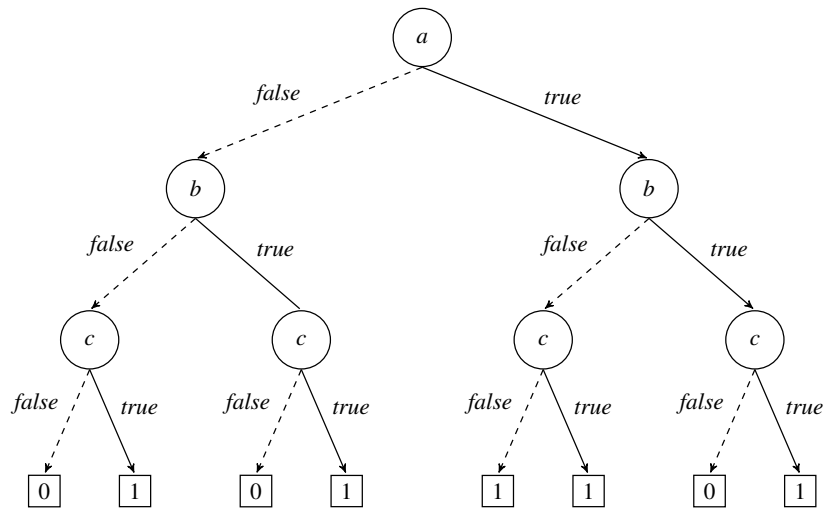


Fig. 2.14: Binary Decision Tree for the function f .

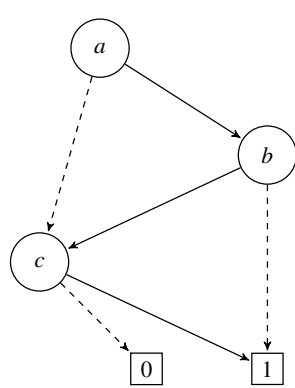


Fig. 2.15: Reduced Ordered Binary Decision Diagram for the function f .

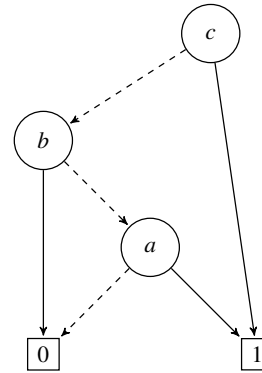


Fig. 2.16: Reduced Ordered Binary Decision Diagram for the function f created with a different variable order with respect to the ROBDD in Fig. 2.15.

theory of regions were provided since the first part of the thesis is based on the aforementioned theory. Finally, a reference to BDDs was made.

Decomposition based on regions theory

Decomposition into sets of synchronizing PNs

Exploiting concurrent models, a key question has been: how can we split a PN into different synchronizing PNs? The theory of regions [26], already used for decades in PN synthesis, allows us not only to answer this question, but also to perform the decomposition of transition systems into a set of desired subclass PNs (a recap on the theory of regions can be found in Section 2.3). The theory of regions allows us not only to create a versatile decomposition framework, but also to prove the existence of a bisimulation between the set of derived synchronized PNs and the initial LTS (Section 3.3.2) or PN (the latter, if we consider as a starting point the reachability graph of the desired Petri net). But before proceeding to the proposed method based on the theory of regions, let us first explore some significant techniques for decomposing Petri nets and transition systems that have been suggested in the past.

3.1 Evolution of the decomposition

Next, a chronological list with some meaningful works about Petri net decomposition is provided:

- 1974: The first attempt to decompose a net was due to M. Hack [33], in order to extend results about liveness of Free-choice Petri Nets, where the decomposition was performed extracting S-components (which actually can be seen as decomposition into a set of synchronizing State Machines);
- 1981: in [34] the usage of invariants for the Petri net decomposition was introduced;
- 1989: in [29] Murata consolidated the concept of live and safe Free-Choice Petri net as a combination of State Machines;
- 1994: in [35] Kemper proposed an algorithm for the Cover of S-Components for Extended Free-Choice Petri nets in $O(PT)$.

Moving towards more recent times, there were different attempts focused on transition system decomposition. In [36], a transition system is decomposed iteratively into an interconnection of n component transition systems with the objective of extracting a Petri net from them. This can be seen as a special case of the problem solved in this chapter, because in [36] the decomposition allows the extraction of a Petri net, but the decomposed set of transition systems

cannot be used as an intermediate model. Their approach is flexible in choosing how to split the original transition system, but it does not provide any minimization algorithm, so that the redundancy due to overlapping states in the component transition systems translates into redundant places of the final Petri net. Another method presented in [12] is based on the decomposition of transition systems into “slices”, where each TS is synthesized separately into a Petri net, and in case of Petri nets “hard” to understand the process can be recursively repeated on one or more “slices” creating a higher number of smaller PNs. With respect to the aforementioned methods, our approach yields by construction a set of PNs restricted to the desired subclass and applies minimization criteria to them. Instead, the results of [37] show how complex processes can be formally represented by process windows, where each window covers part of the overall process behavior. In our case, each PN could be interpreted as a window that represents a part of the entire process.

3.2 Decomposition based on theory of regions

In order to decompose a PN or a transition system into a set of synchronized PNs which generate the same behavior of the original one, the properties of Def. 34 have to be satisfied, considering LTS $TS = RG(PN_0)$, where PN_0 is the initial Petri net, or starting directly from TS .

Definition 34 (Excitation-closed set of Petri nets derived from an ECTS). *Given a set of PNs N derived from an ECTS TS , the set of all regions R of N , the set of labels E of TS , the sets of pre-regions $\bullet e$ of the TS for all $e \in E$, the set of events E_k and regions R_k of PN k :*

N is excitation-closed with respect to TS if the following conditions are satisfied:

- *Excitation closure:* $\forall e \in E : \bigcap_{r \in (\bullet e \cap R)} r = ES(e)$
- *Event effectiveness:* $\forall e \in E : \exists r \in R \mid r \in \bullet e$
- *Place-connectedness:* $\forall r \in R : (r \in R_k, e_i \in (\bullet r \cup r\bullet)) \implies e_i \in E_k$

The first property guarantees that for each event of the original PN there is a sufficient number of its pre-regions across the derived PNs, so that, given a marking in the decomposed set of PNs, a transition representing the event can fire if and only if a transition with the same event fires in the original PN. An insufficient number of pre-regions of an event would relax the constraint allowing the activation of events whose activation is not allowed in the original PN.

The second property guarantees that for each event of the original PN there is a region from which the event is enabled to fire.

The third property guarantees that, if a region represents a place in one of the derived PNs, this place will keep the connections of the region to all events for which the region is a pre-region or a post-region.

Next we will list the main steps to obtain a set of synchronized PNs:

1. Transformation of the PN into an LTS: reachability graph extraction
2. If LTS is not an ECTS *label splitting* is performed
3. Computation of all minimal regions

4. Generation of a set of PNs with desired constraints and *EC* property
5. Optional: removal of redundant PNs
6. Optional: merge between regions preserving *EC*

The first three steps of the flow can be easily found in the literature, and they are summarized in Sec. 2.3. Instead, the other steps are explained in this chapter.

To decompose an LTS/PN in a set of synchronized PNs, the excitation closure property is necessary and sufficient to guarantee an equivalent behavior of the decomposed PN. However, the decision on how to choose the regions for each PN can affect the size of the result. In the next section we will show how an additional constraint can be propagated to obtain a decomposition into a set of Petri nets of a certain class. The result can have a lot of redundancy, especially because the most efficient flows present approximated steps and pass through Minimal Saturated Petri nets, which describe the maximally redundant set of places derived from regions. For this reason, after decomposition, redundant PNs can be removed. In some cases, it is not possible to remove entire Petri nets but only redundant parts of them.

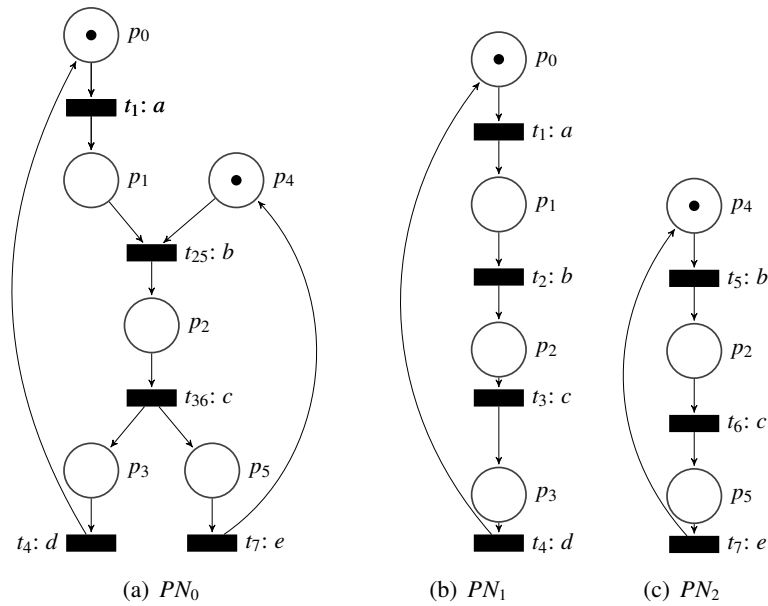


Fig. 3.1: Example of decomposition: PN_1 and PN_2 are derived from PN_0 .

Taking into account the PN in Fig. 3.1(a) as the original PN, the Petri nets in Figs. 3.1(b) and 3.1(c) can be considered as random choices of pairs of PNs that satisfy the excitation closure property. In this case, no PN and no place can be removed with further steps.

N.B. Since our decomposition algorithm is based on the theory of regions, as in the case of PN synthesis, the resulting PNs cannot have multiple occurrences of the same event, unless the event is identified with two different labels, for example a and $a/1$.

There is not only one way to generate a set of synchronizing PNs from an LTS, especially if we restrict the Petri nets to a specific subclass, since each approach has different pros and cons. In the next sections, the following approaches will be presented:

- Exact sequential search: explores all possible SMs and returns the smallest subset of them, not scalable at all.
- Approximate sequential search (e.g., of SMs and FCPNs): based on a heuristic search, scalable.
- Mixed strategy: a partial combination of the approximated sequential search followed by an exhaustive removal of redundant PNs, yielding a better result than the approximate sequential search.
- Simultaneous search: guarantees the minimality of the number of PNs, but its performance depends heavily on the subclass of PNs used in the decomposition.

Tests were performed especially on FCPNs and SMs.

3.3 Composition of PNs and equivalence to the original TS/PN

In this section, we prove a structural equivalence between the original TS and the synchronous composition of the reachability graphs of the PNs, by defining a bisimulation relation between them, under the assumptions of excitation-closure and place-connectedness of the PNs. This result is part of the contributions of our paper accepted at Digital System Design (DSD2023) [17].

Theorem 4. [17] *Given a set $\{PN_1, \dots, PN_n\}$ of PNs derived from the ECTS TS, there is a bisimulation B such that $TS \sim_B \parallel_{i=1, \dots, n} RG(PN_i)$ iff:*

1. *the set $\{PN_1, \dots, PN_n\}$ of PNs satisfies excitation closure over the events of TS;*
2. *each component satisfies place-connectedness.*

Proof. See Section 3.3.2.

If we start our flow from a transition system T , we can see that the set of derived PNs represents the same behavior of T .

In Fig. 3.3 we can observe the FCPNs obtaining decomposing TS in Fig. 3.2, with the respective reachability graphs in Fig. 3.4. Fig. 3.5 instead shows the synchronous product of the reachability graphs in Fig. 3.4, which is bisimilar to the original ECTS.

Notice that the proof presented in Section 3.3.2 is constructive, i.e. it builds the bisimulation relation by defining the structural mapping. One could take a behavioral approach showing a relation between languages using the theory in [27] (which implies the existence of a bisimulation for deterministic systems), but this would not yield the actual bisimulation.

3.3.1 Safe composition of unsafe PNs

Given a set of PNs from a decomposition, even if the composition of PNs is safe, single PNs considered without synchronizations could be unsafe, still keeping the bisimulation with the

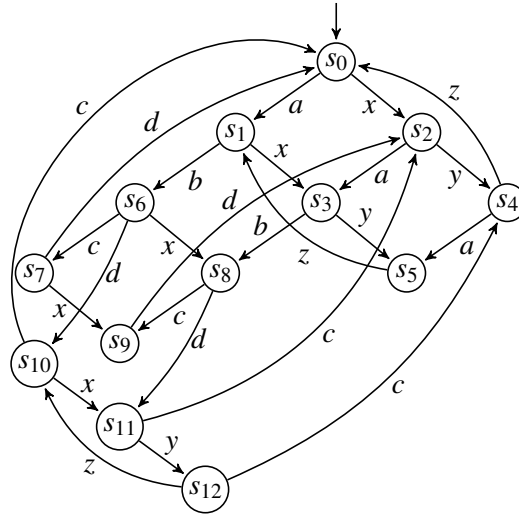


Fig. 3.2: Transition system example.

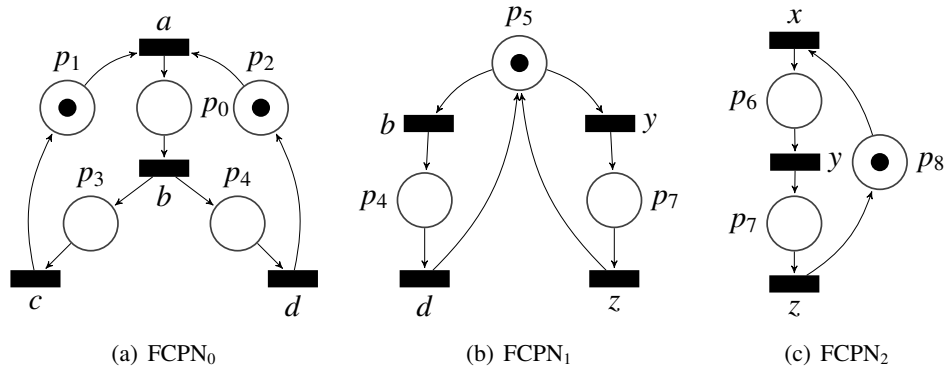


Fig. 3.3: Three FCPNs distilled from the TS in Fig. 3.2

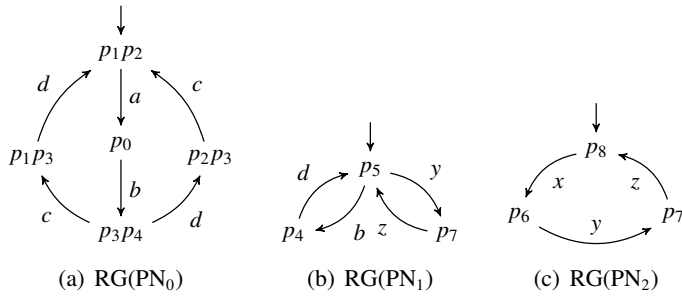


Fig. 3.4: Reachability graphs of the FCPNs in Fig. 3.3.

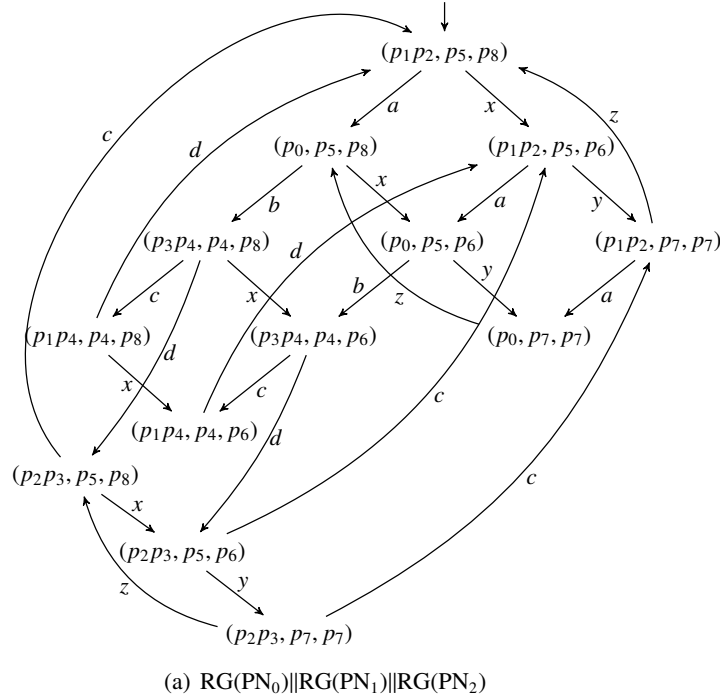


Fig. 3.5: Synchronous product of reachability graphs in Fig. 3.4.

original ECTS. Fortunately, in [38] the authors proved the existence of a safe PN starting from an ECTS. In our case, the decomposition into sets of PNs can be seen as a special case, for example, the decomposition into a set of synchronized SMs can be interpreted as the S-covering of a PN, which is proven to be safe. For other subclasses such as FCPNs or ACPNs, the existence of a safe SM decomposition implies also the existence of a decomposition by means of the aforementioned PN subclasses, where in the worst case all components would be SMs.

Let’s see an example of a safe composition of unsafe PNs. In Fig. 3.6 we have two unsafe Free-choice Petri nets. An example of a sequence that produces an unsafe marking is “*abce*” in the first FCPN and “*abec*” in the second. If we examine the two FCPNs, the second one has the same structure as the first, with the transitions containing events *c* and *e* exchanged between themselves with respect to the first FCPN. Both FCPNs are unsafe, but, with synchronization, the different event sequence between the two PNs creates a lock between them, disabling the firing of both aforementioned events and keeping unsafe markings unreachable. Indeed, Fig. 3.7 represents the composition of the reachability graphs of the PNs, which never reaches an unsafe marking.

In the next sections, we will see different ways to decompose an LTS into distinct PN subsets, in some cases like for SMs it is sufficient to avoid multiple initial places to keep safe a State Machine. For more complex PN subsets, a direct safeness check will be needed.

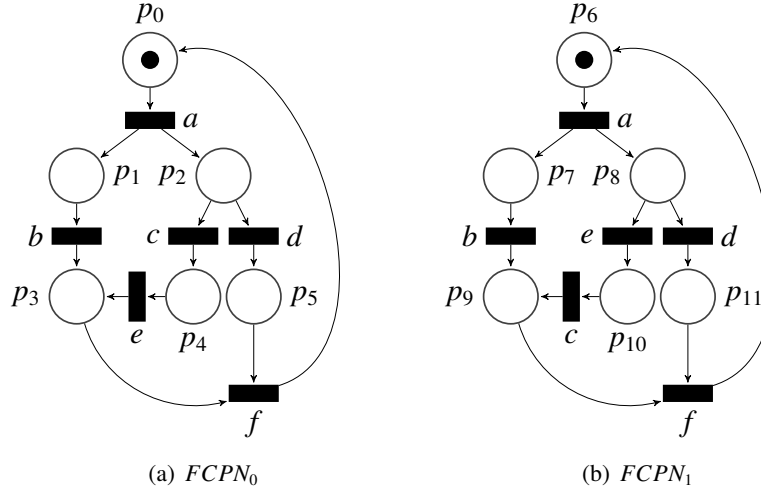


Fig. 3.6: Two unsafe Free-choice Petri nets with a safe synchronization.

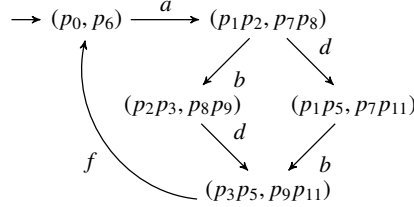


Fig. 3.7: Synchronous product of the reachability graphs of FCPNs in Fig. 3.6.

3.3.2 Proof of Theorem 4

Place connectedness is a property which consists in keeping all connections of a region (*exit/enter*) passing from regions to the place representation in PNs. This fundamental property, previously left implicit, enables us to use regions in the bisimulation, as it will be described soon. The lack of *place connectedness* means the loss of connections passing from a region r to a place p . As result p does not represent r anymore, but it represents an invalid region or a different region r' which has a *no cross* relation (neither *enter* nor *exit*) with the missing event. A region could be represented by different instances of places in two or more different PNs, but each of these places still preserves the same connections to the events of the represented region.

In this proof the following nomenclature will be used:

R is the total set of regions;

$R_i \subseteq R$ is the set of regions represented by places of PN_i ;

$R(s)$ is the set of regions that contain state s ;

$\overline{R_i}(s) = R(s) \cap R_i$ is the set of regions represented by places of PN_i that contain state s ;

$\overline{R}(s) = (\overline{R_1}(s), \dots, \overline{R_n}(s))$, this alias will be used for improved readability.

The equivalence between an ECTS and the derived set of PNs is proved by defining a bisimulation between the original TS, defined as $TS = (S, E, T, s_0)$, and the synchronous product of the reachability graphs of the derived Petri nets $RG(PN_1) \parallel RG(PN_2) \parallel \dots \parallel RG(PN_n)$, de-

noted by $\parallel_{i=1, \dots, n} RG(PN_i) = (S_{\parallel}, E, T_{\parallel}, s_{0, \parallel})$. Notice that each $RG(PN_i) = (R_i, E_i, T_i, R_i(s_0))$, with $T_i \subseteq R_i \times E_i \times R_i$, is defined on a subset E_i of events of TS , its states R_i correspond to regions or sets of regions derived from TS represented by the markings $[M]$ of PN_i .

The initial state of $RG(PN_i)$ is represented by $R_i(s_0)$: one or more regions containing the initial state s_0 of TS .

Excitation closure is another fundamental property. Indeed to prove the existence of a bisimulation we require that the regions contained in the set of PNs satisfy EC, where event-effectiveness guarantees that $\cup E_i = E$. The excitation closure property is crucial to prove the steps 1 and 3 of the proof.

Proof. We define the binary relation B as follows:

$$(s, \overline{R(s)}) \in B \iff s \in \bigcap_{i=1}^n r \mid r \in R_i(s) \quad (3.1)$$

where:

- $s \in S$;
- $\overline{R(s)}$ or $(R_1(s), \dots, R_n(s))$ is a tuple of markings where each marking $R_i(s)$ contains s .

Notice that writing $(s, \overline{R(s)}) \in B \iff \{s\} = \bigcap_{i=1}^n r \mid r \in R_i(s)$ would be wrong, because the intersection of regions could have two or more bisimilar (i.e., behaviourally equivalent) states, as in the TS $s_0 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{a} s_3 \xrightarrow{b} s_0$.

Now we prove that B is a bisimulation in three steps:

1. $(s_0, \overline{R(s_0)}) \in B$.
2. If $(s_j, \overline{R(s_j)}) \in B$ and $(s_j, e, s_k) \in T$, then there is $\overline{R(s_k)} \in S_{\parallel}$ such that $(\overline{R(s_j)}, e, \overline{R(s_k)}) \in T_{\parallel}$ and $(s_k, \overline{R(s_k)}) \in B$.
3. If $(s_j, \overline{R(s_j)}) \in B$ and $(\overline{R(s_j)}, e, \overline{R(s_k)}) \in T_{\parallel}$, then there is $s_k \in S$ such that $(s_j, e, s_k) \in T$ and $(s_k, \overline{R(s_k)}) \in B$.

Let us now proceed with the proofs.

1. Since TS has a unique initial state s_0 , each PN PN_i has at least one initial region $r \in R_i(s_0)$ such that $s_0 \in r$ because the regions of a PN cover all the states of initial TS (otherwise *excitation closure* would not be satisfied). Therefore, $s_0 \in \bigcap_{i=1}^n r \mid r \in R_i(s_0)$ and we have that $(s_0, \overline{R(s_0)}) \in B$.
2. Since $(s_j, e, s_k) \in T$ and $(s_j, \overline{R(s_j)}) \in B$, then $s_j \in \bigcap_{i=1}^n r \mid r \in R_i(s_j)$. Now we will prove that there is s_k such that $s_k \in \bigcap_{i=1}^n r \mid r \in R_i(s_k)$, so that we can have $(s_k, \overline{R(s_k)}) \in B$.
Notice that PNs may share places, but the property of place-connectedness guarantees that shared places have the same marking, i.e. if place p occurs in more than one PN_i and an event e is entering or exiting in one occurrence of p then e will be entering or exiting in all occurrences of p .
Since e is enabled in s_j , no region $R_i(s_j)$ can be a post-region of e . If one $r \in R_i(s_j) \in \overline{R(s_j)}$ would be a post-region, then $s_j \notin \bigcap_{i=1}^n r \mid r \in R_i(s_j)$. Therefore, the following cases can be distinguished for each $r \in R_i(s_j) \in \overline{R(s_j)}$:

- e is not an event of PN_i . Thus, $R_i(s_k) = R_i(s_j)$.
- e is an event of PN_i ,
 - and all regions $r \in R_i(s_j)$ are no-cross regions for e . Thus, $R_i(s_k) = R_i(s_j)$. Explanation: say that each of the n places of the current marking $R_i(s_j)$ represents a region. By place-connectedness each place is connected to all events of the corresponding region. So, if e is no-cross with respect to all regions, then no place of the PN will be connected to e , and firing e cannot remove tokens from any place of $R_i(s_j)$, and so $R_i(s_k) = R_i(s_j)$.
 - and all regions $r \in R_i(s_j)$ are pre-regions of e . Thus, $R_i(s_k) \neq R_i(s_j)$ and each region $r \in R_i(s_k)$ is a post-region of e . By place-connectedness.
 - and some regions of $R_i(s_j)$ are pre-regions of e . Thus, $R_i(s_k) \neq R_i(s_j)$ and each region $r \in R_i(s_k) \setminus R_i(s_j)$ is a post-region of e . By place-connectedness.

For the first and second case, PN_i will not change marking and TS will not change region when moving from s_j to s_k . Therefore, $s_k \in r \in R_i(s_j) = R_i(s_k)$.

For the third and fourth case, e will exit $r \in R_i(s_j)$ in at least one PN and will enter $r \in R_i(s_k)$ in TS , which means that $s_k \in r \in R_i(s_k)$. Therefore, $(\overline{R(s_j)}, e, \overline{R(s_k)}) \in T_{\parallel}$.

For all cases we have that $s_k \in r \in R_i(s_k)$ and therefore $s_k \in \bigcap_{i=1}^n r \mid r \in R_i(s_k)$.

3. Since $(s_j, \overline{R(s_j)}) \in B$, it holds that $s_j \in \bigcap_{i=1}^n r \mid r \in R_i(s_j)$. Given the existence of the transition $(\overline{R(s_j)}, e, \overline{R(s_k)})$, and knowing that the *excitation closure* property holds, we know that $s_j \in \bigcap_{i=1}^n r \mid r \in R_i(s_j) \subseteq ES(e)$. The latter inequality holds because we have 1) by PN construction, $\forall i, i = 1, \dots, n$, label e appears once in PN_i or it does not appear, and 2) being all states covered by at least one region of the set of PNs, $\forall i, i = 1, \dots, n$, if label e appears in PN_i then $r \in R_i(s_j)$ could be a no-cross region, otherwise $r \in R_i(s_j) \in (\bullet e \cap R)$, by which $\bigcup_{i=1}^n r \mid r \in R_i(s_j) \supseteq \bigcup_{r \in (\bullet e \cap R)} \{r\}$ and so by intersection of the regions seen as sets of states $\bigcap_{i=1}^n r \mid r \in R_i(s_j) \subseteq \bigcap_{r \in (\bullet e \cap R)} r = ES(e)$.

Therefore, there is s_k such that $(s_j, e, s_k) \in T$. We can also see that $s_k \in \bigcap_{i=1}^n r \mid r \in R_i(s_k)$, using the same reasoning as in step 2, since all the pre-regions $r \in R_i(s_j)$ of e in $\overline{R(s_j)}$ are exited by entering $r \in R_i(s_k)$, whereas the no-crossing regions remain the same. We can then conclude that $(s_k, \overline{R(s_k)}) \in B$.

3.4 What happens if excitation-closure is not satisfied?

Starting from Fig. 3.8 let us consider a decomposition which does not satisfy excitation-closure. Table 3.1 lists the regions obtained from the transition system in Fig. 3.8. from which we notice that for events a and b excitation closure is not satisfied and label splitting should be performed. In particular $r_3 = \{s_1, s_3, s_5, s_7\}$ is the only pre-region for event a , and $ES(a) = \{s_1, s_3, s_5\} \neq r_3$. The region $r_2 = \{s_1, s_2, s_5, s_6\}$ instead is the only pre-region for event b , and $ES(b) = \{s_1, s_2, s_5\} \neq r_2$. Continuing with the decomposition without having performed label splitting, we obtain the set of SMs in Fig. 3.9, which is one of the different possible decomposition results that can be obtained. The composition of the reachability graphs of the SMs in Fig. 3.8 is shown in Fig. 3.10. We can immediately notice an additional state with respect to the initial transition

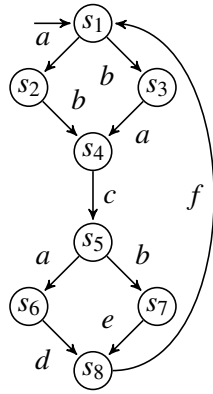


Table 3.1: Regions derived from TS in Fig. 3.8.

Region	States of the region
r_1	$\{s_1, s_2, s_3, s_4\}$
r_2	$\{s_1, s_2, s_5, s_6\}$
r_3	$\{s_1, s_3, s_5, s_7\}$
r_4	$\{s_2, s_4, s_6\}$
r_5	$\{s_3, s_4, s_7\}$
r_6	$\{s_5, s_6, s_7\}$
r_7	$\{s_8\}$

Fig. 3.8: Example of transition system with label splitting required for the decomposition based on regions theory.

system. This “error” state is reached exactly with two of the events involved in the violation of excitation-closure because without label splitting it is not possible to distinguish different instances of the aforementioned events: the marking (r_1, r_2, r_4) cannot be distinguished with (r_6, r_2, r_4) , allowing the firing of event b from (r_6, r_2, r_4) and the marking (r_1, r_5, r_3) cannot be distinguished with (r_6, r_5, r_3) , allowing the firing of a from (r_6, r_5, r_3) . In fact, the excitation closure property is **necessary** and **sufficient** in order to maintain the behavior of the original LTS.

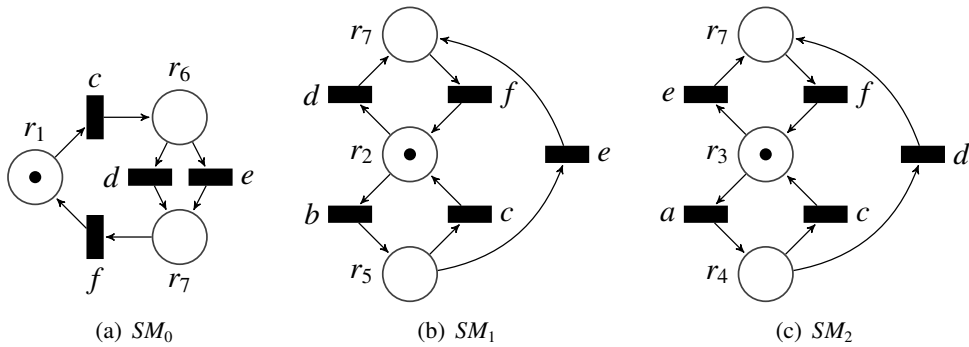


Fig. 3.9: SM decomposition derived from TS in Fig. 3.8.

3.5 What about the decomposition into sets of synchronizing Marked Graphs?

Previously we described an algorithm for the decomposition of a TS/PN in a set of synchronized SMs, so the following questions come next: “Is it possible to find a decomposition into a set of marked graphs?” and especially “Is it always possible to obtain such a decomposition?”. Remember that marked graphs are dual of state machines, where each place has exactly one entering edge and one exiting edge.

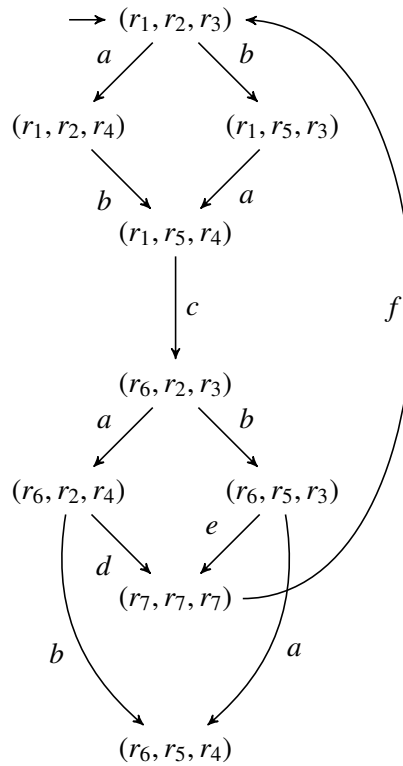


Fig. 3.10: Synchronous product of the reachability graphs of SMs in Fig. 3.9.

It has been observed in [39] that starting from a transition system it is not always possible to derive a marked graph with the same behavior, and sufficient conditions were provided for marked graph realizability. An example of a transition system that cannot be transformed into a marked graph is shown in Fig. 3.11.

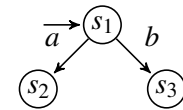


Fig. 3.11: Example of a TS which cannot be transformed into an MG.

If we want to decompose a transition system into a set of Marked Graphs, how should the MG structure be encoded? Since in Marked Graphs the choice structure is forbidden, given the set of regions derived from a transition system, only regions with exactly one entering and one outgoing event should be allowed (actually the constraint on only outgoing events would be sufficient), to force the derived PNs to be Marked Graphs.

Fig. 3.12 shows an ECTS containing a choice starting from s_4 (the choice on s_0 actually represents the start of a diamond with two concurrent flows, involving events a and b). Given the excitation sets of the events of the TS (Table 3.3), we can see that all minimal regions (Table 3.2) are necessary: $ES(a) = r_0$, $ES(b) = r_1$, $ES(c) = r_2 \cap r_3$ and $ES(d) = r_4$. Since both r_2 and r_3 have multiple outgoing events, the two regions cannot be used for MG decomposition. Since every region is necessary for the achievement of excitation-closure, the absence of r_2 and r_3 clearly shows the impossibility of executing a decomposition that has a bisimulation between reachability graphs of the derived MGs and the initial transition system.

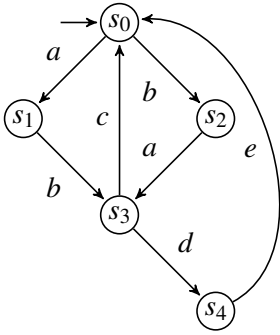


Fig. 3.12: Example ECTS

Table 3.2: Minimal regions derived from ECTS in Fig. 3.12.

Region	States of the region
r_0	$\{s_0, s_2\}$
r_1	$\{s_0, s_1\}$
r_2	$\{s_1, s_3\}$
r_3	$\{s_2, s_3\}$
r_4	$\{s_4\}$

Table 3.3: Excitation sets for each event of ECTS in Fig. 3.12.

Event	ES(event)
a	$\{s_0, s_2\}$
b	$\{s_0, s_1\}$
c	$\{s_3\}$
d	$\{s_4\}$

In this chapter, we introduced the decomposition of Transition Systems into synchronized Petri nets, performing the decomposition with the theory of regions. We highlighted the critical role of excitation closure, a vital condition for a correct decomposition, and demonstrated it, showing the consequences when it is not met. We also touched on the safety of the resulting model, even when unsafe PNs are part of the decomposition result. Additionally, we illustrated the challenges in decomposing a Transition System using Marked Graphs.

Transition System decomposition into sets of synchronizing SMs

Suppose that we want to decompose an LTS in a restricted set of Petri nets, for example, state machines. Each SM represents a flow of the original PN with only one token. In this chapter, we will present a possible application of the flow presented in the previous chapter and restricted to SMs. The generation of the regions is well known in PN synthesis; the subsequent steps, instead, represent our contribution, published at the 24th Conference on Digital System Design (DSD2021) [40].

For the generation of SMs only **minimal** regions are needed. A set of regions represents a State Machine if and only if the set covers all the states of the transition system and all the regions are disjoint, i.e., given an LTS with set of states S , a set of regions R represents a state machine iff:

1. $\forall r \in R \nexists r' \in R \mid r \cap r' \neq \emptyset$
2. $\forall s \in S \exists r \in R \mid s \in r$

Given a set of regions satisfying the previous properties, we get a state machine whose states correspond to the regions, with a transition from state r_i to state r_j under event e when r_i is a pre-region of e with post-region r_j . The initial state of the SM corresponds to the region including the initial state of the LTS,

The first step to decompose a transition system is to enumerate all the minimal regions of the original TS. Each collection of disjoint regions covering all the states of the TS represents a state machine, such that the regions are mapped to places of the SM, i.e., each such SM includes a subset of regions of the original TS and represents only the behavior related to the transitions entering into these regions or exiting from them (instead, internal and external events are missing).

Intuitively, the SM obtained as the synchronous product of two or more SMs models their interaction. For example, consider the SMs in Fig. 4.3, then the composition of the reachability graphs of the first two SMs (SM_4 and SM_5) yields the LTS in Fig. 4.4, which generates a subset of behaviors of the original TS (for example, the sequence “*acbdafed*”), but it can exhibit also new behaviors (e.g., sequences that start by firing the event *b*) because some constraints of the original TS are missing; indeed, these two SMs are not enough to satisfy the excitation-closure property, whereas event effectiveness is satisfied by them because all events are included in the

composition. In this example, by considering a single SM, even event effectiveness may fail, when some events are hidden because they are completely inside or outside some regions: e.g., considering only SM_4 , event effectiveness is not satisfied because the events b and f are missing (in this case sequences containing the aforementioned events cannot be produced, for example the previously cited sequence “ $acbdafd$ ”). The composition of SMs can exhibit these hidden behaviors by including new regions: for example, the composition of SM_4 with SM_5 includes two new regions r_{11} and r_{12} so that the events b and f show up in the composition.

In Sec. 3.3.2 we proved that, given a set of PNs (therefore also SMs), the excitation closure and event effectiveness of the union of their regions, in combination with place connectedness is a necessary and sufficient condition to guarantee that their synchronous product is equivalent to the original transition system. As an example, it can be verified that the composition of the four SMs in Fig. 4.3, which satisfy the excitation closure and event effectiveness properties, yields the original TS.

The previous example also shows that we do not need all the SMs to reconstruct the original TS, so the question is how many of them do we need and which is the “best” (in some sense) subset of SMs sufficient to represent the given TS. Therefore, we may set up a search to obtain a subset of SMs, which are excitation-closed and cover all events, to yield a composition equivalent to the original TS.

Since well-formed State Machines with only one token in the initial marking are guaranteed to be safe, the method presented next will produce only safe SMs.

4.1 Sequential SM search

A strategy to guarantee the complete coverage of all events is to add new SMs until all regions are used. However, the resulting collection of SMs may contain completely or partially redundant SMs (see Secs. 4.1.2 and 4.1.3), which can be removed exactly or greedily by verifying the excitation-closure property. Moreover, the size of the selected SMs can be reduced by removing redundant labels by merging regions.

The first step of the algorithm can be achieved by a greedy algorithm from the literature, which checks minimality while creating regions (Section 2.3.3).

Definition 35 (Independent Set). *Given an undirected graph $G = (V, E)$, an independent set is a subset of nodes $U \subseteq V$ such that no two nodes in U are adjacent.*

Definition 36 (Maximal Independent Set, MIS). *An independent set is maximal if no node can be added without violating independence.*

The second step of the decomposition algorithm is performed by reducing it to an instance of maximal independent set (MIS), and by calling an MIS solver on the graph whose vertices correspond to the minimal regions with edges connecting regions that intersect. Each *maximal independent set* of the aforementioned graph corresponds to a set of disjoint regions that define an SM. Algorithm 1 shows the detailed pseudocode.

A greedy algorithm is used for the computation of the third step: starting from the SM with the highest number of regions, one removes each SM whose removal does not invalidate the ECTS properties.

The last step of merging is reduced to an SAT instance, by encoding all the regions of each SM and also the events implied by the presence of one or more regions. By solving this SAT instance with an SAT solver, the number of labels can be minimized by merging the regions that occur multiple times in different SMs.

4.1.1 Generation of a set of SMs with excitation closure

Given a set of minimal regions of an excitation-closed TS, Algorithm 1 returns an excitation-closed set of SMs, by associating sets of non-overlapping regions to SMs as detailed below.

Initially, Algorithm 1 converts the minimal regions of the TS into a graph G , where intersecting regions define edges between the nodes of G (line 1). A copy G_0 of G is created (line 2), to save the original graph because G is modified during the execution of the algorithm. Then an empty set M is defined, whose elements are the outputs of the MIS procedure, i.e., each element of M is a maximal independent set of nodes of G ; similarly, the set F is defined to store the resulting state machines (line 3). At this point, as long as G is not empty, the search for maximal independent sets is performed on it by invoking the MIS procedure on G ($MIS(G)$, line 5), storing the results in M (line 6) and removing the vertices selected at each iteration (line 7). In this way, each vertex will be included in one MIS solution. Notice that the maximal independent sets computed after the first one are not maximal with respect to the original graph G_0 , because the MIS procedure is run on a subgraph of G_0 without the previously selected nodes. To ensure that we obtain maximal independent sets with respect to the original G_0 , we expand the independent sets to maximality in M , by invoking the MIS procedure on each independent set $m \in M$ constrained to obtain a maximal independent set $\tilde{m} \supseteq m$ in G_0 (from line 9). Then from the maximal independent sets we obtain the induced state machines to be stored in F (from line 12). The motivation of this step to enlarge the independent sets is to increase the number of

Algorithm 1: Excitation-closed SM set generation

Input: Set of minimal regions of an ECTS
Output: An excitation-closed set of SMs

- 1 Create the graph G where each node is a region and there is an edge between intersecting regions
- 2 $G_0 \leftarrow G$
- 3 $M \leftarrow \emptyset, F \leftarrow \emptyset$
- 4 **do**
- 5 Compute $m = MIS(G)$
- 6 $M \leftarrow M \cup \{m\}$
- 7 $G \leftarrow G_{\downarrow M}$
- 8 **while** $G \neq \emptyset$;
- 9 **for** $m \in M$ **do**
- 10 Compute $\tilde{m} = MIS(G_0)$ with the constraint $\tilde{m} \supseteq m$
- 11 Build state machine \tilde{sm} induced by set of regions \tilde{m}
- 12 $F \leftarrow F \cup \{\tilde{sm}\}$
- 13 **return** F

- $$M = \{\{r_1, r_6, r_{16}\}, \{r_2, r_7\}\}$$
- $$Nodes(G) = \{r_2, r_3, r_4, r_5, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{17}\} \setminus \{r_2, r_7\} = \{r_3, r_4, r_5, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{17}\}$$
3. $mis = \{r_3, r_4\}$
- $$M = \{\{r_1, r_6, r_{16}\}, \{r_2, r_7\}, \{r_3, r_4\}\}$$
- $$Nodes(G) = \{r_3, r_4, r_5, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{17}\} \setminus \{r_3, r_4\} = \{r_5, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{17}\}$$
4. $mis = \{r_8, r_{14}\}$
- $$M = \{\{r_1, r_6, r_{16}\}, \{r_2, r_7\}, \{r_3, r_4\}, \{r_8, r_{14}\}\}$$
- $$Nodes(G) = \{r_5, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{17}\} \setminus \{r_8, r_{14}\} = \{r_5, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{15}, r_{17}\}$$
5. $mis = \{r_{11}, r_{12}\}$
- $$M = \{\{r_1, r_6, r_{16}\}, \{r_2, r_7\}, \{r_3, r_4\}, \{r_8, r_{14}\}, \{r_{11}, r_{12}\}\}$$
- $$Nodes(G) = \{r_5, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{15}, r_{17}\} \setminus \{r_{11}, r_{12}\} = \{r_5, r_9, r_{10}, r_{13}, r_{15}, r_{17}\}$$
6. $mis = \{r_5, r_{17}\}$
- $$M = \{\{r_1, r_6, r_{16}\}, \{r_2, r_7\}, \{r_3, r_4\}, \{r_8, r_{14}\}, \{r_{11}, r_{12}\}, \{r_5, r_{17}\}\}$$
- $$Nodes(G) = \{r_5, r_9, r_{10}, r_{13}, r_{15}, r_{17}\} \setminus \{r_5, r_{17}\} = \{r_9, r_{10}, r_{13}, r_{15}\}$$
7. $mis = \{r_9, r_{13}\}$
- $$M = \{\{r_1, r_6, r_{16}\}, \{r_2, r_7\}, \{r_3, r_4\}, \{r_8, r_{14}\}, \{r_{11}, r_{12}\}, \{r_5, r_{17}\}, \{r_9, r_{13}\}\}$$
- $$Nodes(G) = \{r_9, r_{10}, r_{13}, r_{15}\} \setminus \{r_9, r_{13}\} = \{r_{10}, r_{15}\}$$
8. $mis = \{r_{10}, r_{15}\}$
- $$M = \{\{r_1, r_6, r_{16}\}, \{r_2, r_7\}, \{r_3, r_4\}, \{r_8, r_{14}\}, \{r_{11}, r_{12}\}, \{r_5, r_{17}\}, \{r_9, r_{13}\}, \{r_{10}, r_{15}\}\}$$
- $$Nodes(G) = \{r_{10}, r_{15}\} \setminus \{r_{10}, r_{15}\} = \emptyset$$

The last cycle of the procedure checks, for each element m of M , if there is a larger independent set $\tilde{m} \supseteq m$ in G_0 . The only independent sets which are extended are:

$SM_4 = \{r_1, r_8, r_{14}\}$ because $\{r_8, r_{14}\}$ is not a mis on G_0 .

$SM_6 = \{r_1, r_5, r_{17}\}$ because $\{r_5, r_{17}\}$ is not a mis on G_0 .

$SM_7 = \{r_1, r_9, r_{13}\}$ because $\{r_9, r_{13}\}$ is not a mis on G_0 .

The final set of SMs is represented by Fig. 4.2.

Sometimes a set of regions given from the MIS solver could actually represent two (or more) SMs, since no constraint on the complete connection of all regions has been added. That is not a problem, since we can split the disconnected sets of regions as separate SMs, forbidding the smallest ones. In this way sooner or later a solution with only one SM will be found, and it will be the biggest available SM. A solution with also smaller SMs could be valid, but heuristically without these SMs the procedure is more likely to find a solution with fewer components.

4.1.2 Removal of the redundant SMs

The set of SMs generated by Algorithm 1 may be redundant, i.e., it may contain a subset of SMs which still define an ECTS. We describe a greedy search algorithm to obtain an irredundant set of SMs: we order all the SMs by size and try to remove them one by one starting from the largest to the smallest, by checking that the union of the remaining regions satisfies *excitation-closure* and *event effectiveness*. If excitation closure and event effectiveness are preserved, then

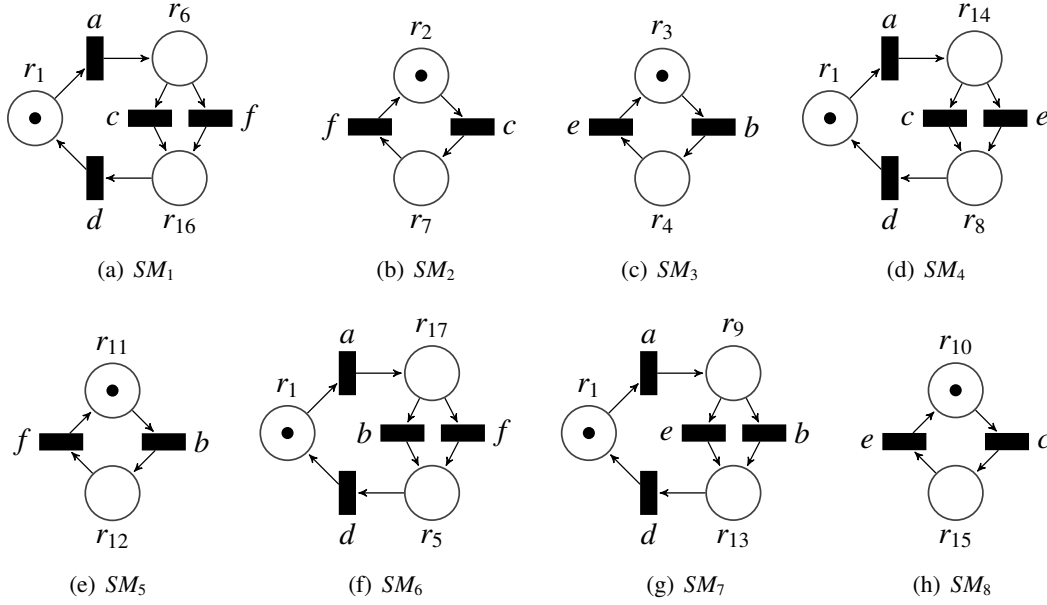


Fig. 4.2: All SMs created from TS in Fig. 2.2.

the given SM can be removed. This algorithm is not optimal, because the removal of an SM may prevent the removal of a set of smaller SMs whose sum of states is greater than the number of states of the removed SM. However, this approach guarantees good performance, having linear complexity in the number of SMs.

To check if the excitation closure property is still valid after the removal of an SM, we consider the excitation sets and the pre-regions (see Table 2.2) for each event of the original transition system. We notice that SM_2 (whose nodes are $\{r_2, r_7\}$) affects only the events c and f (see Fig. 4.2(b)). Indeed, in the graph of SM_2 there is an edge from r_2 to r_7 under c because r_2 is a pre-region of c since c exits from $\{s_1, s_7\} \subseteq r_2 = \{s_0, s_1, s_3, s_5, s_7\}$, and r_7 is a post-region of c since c enters into $\{s_2, s_9\} \subseteq r_7 = \{s_2, s_4, s_6, s_8, s_9\}$; similarly, there is an edge from r_7 to r_2 under f because r_7 is a pre-region of f since f exits from $\{s_4, s_6\} \subseteq r_7$, and r_2 is a post-region of f since f enters into $\{s_3, s_5\} \subseteq r_2$. After the removal of event c , the intersection of the pre-regions is: $r_6 \cap r_{10} \cap r_{14} = \{s_1, s_4, s_6, s_7\} \cap \{s_0, s_1, s_5, s_6, s_7\} \cap \{s_1, s_3, s_4, s_7\} = \{s_1, s_7\} = ES(c)$; after the removal of event f it is: $r_6 \cap r_{12} \cap r_{17} = \{s_1, s_4, s_6, s_7\} \cap \{s_1, s_2, s_4, s_6, s_8\} \cap \{s_4, s_6, s_7, s_9\} = \{s_4, s_6\} = ES(f)$. For the other events, the intersection of pre-regions is unchanged. Thus, SM_2 can be removed. Subsequently, following the same reasoning for the other events, SM_1 , SM_3 , and SM_7 can also be removed. The final result is shown in Fig. 4.3.

4.1.3 Merge between regions preserving the excitation closure

We will use the transition system in Fig. 4.5 as a running example to illustrate this subsection. By the procedure discussed so far, it can be decomposed as the synchronous product of two SMs shown in Fig. 4.6.

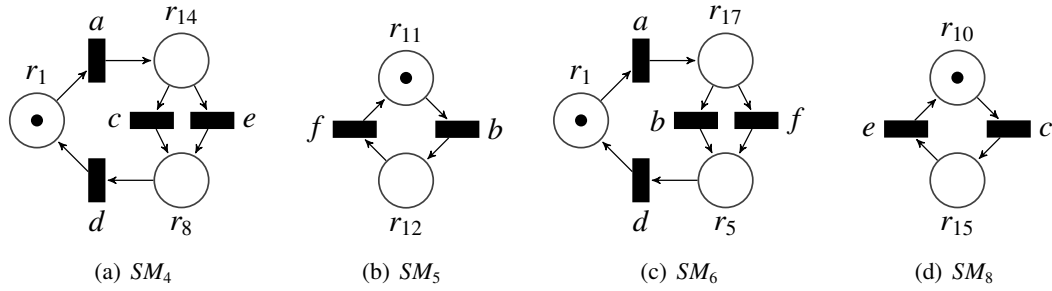


Fig. 4.3: Excitation-closed subset of state machines selected from the SMs in Fig. 4.2 as result of the greedy algorithm.

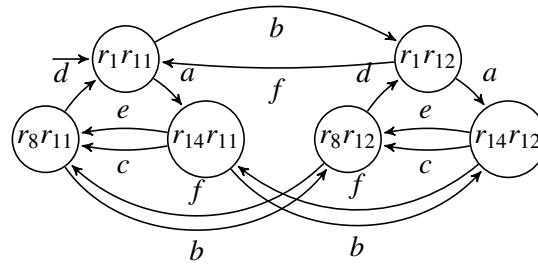


Fig. 4.4: LTS representing the composition $RG(SM_4)||RG(SM_5)$ of SMs in Fig. 4.3.

The third step of the procedure merges pairs of regions with the objective of minimizing the size of the sets of SMs: edges carrying labels are removed, and by consequence the two nodes connected to them are merged, decreasing their number. For example, in Fig. 4.6, both SMs contain an instance of label e connected by regions r_3 and r_4 . This means that an edge carrying label e can be removed in one of the SMs. The result of removing the edge with the label e in SM_b and merging the regions r_3^2 and r_4^2 replacing them with the region r_{34} is shown in Fig. 4.7.

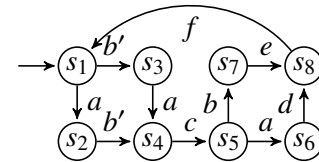


Fig. 4.5: ECTS.

Table 4.2: Minimal regions of the transition system in Fig. 4.5.

Region	States of the TS
r_1	$\{s_1, s_3, s_5\}$
r_2	$\{s_2, s_4, s_6\}$
r_3	$\{s_7\}$
r_4	$\{s_8\}$
r_5	$\{s_1, s_2\}$
r_6	$\{s_3, s_4\}$
r_7	$\{s_5, s_6\}$

Table 4.3: Pre-regions for each event of the transition system in Fig. 4.5

Event	Pre-regions
a	$\{r_1\}$
b	$\{r_1, r_7\}$
b'	$\{r_5\}$
c	$\{r_2, r_6\}$
d	$\{r_2, r_7\}$
e	$\{r_3\}$
f	$\{r_4\}$

In general, one can remove all instances of a region except one, because removing all of them would change the set of regions used for checking the excitation-closure property, whereas keeping at least one guarantees the preservation of the property.

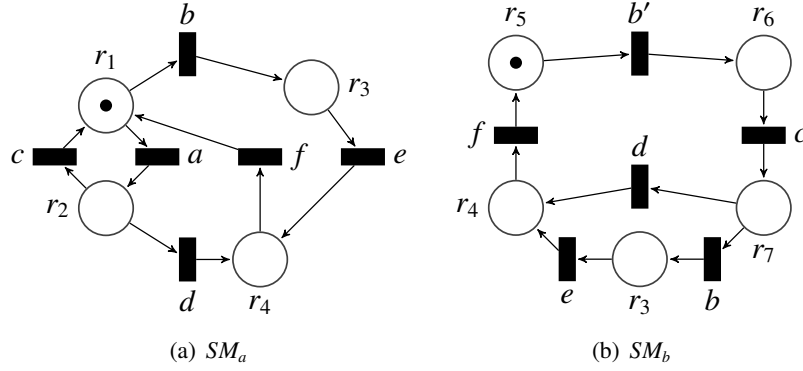


Fig. 4.6: SMs obtained with the MIS solver from the TS of Fig. 4.5.

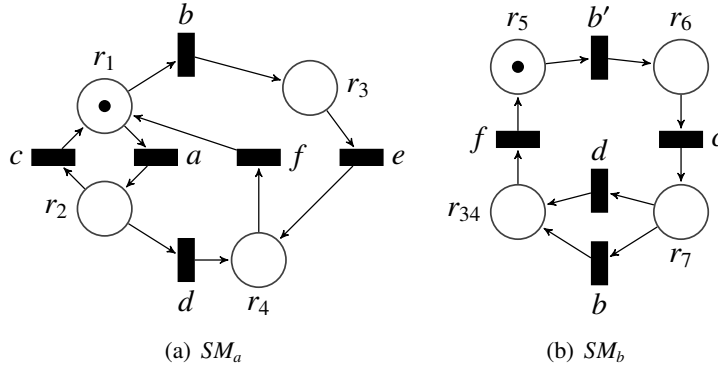


Fig. 4.7: SMs of Fig. 4.6 after the removal of label e in SM_b .

We formulated the merging problem as solving an instance of SAT. We now describe the encoding of the problem. We introduce next three types of SAT clauses required to represent the problem.

1. A set of SAT clauses states that we cannot remove all instances of a given region r from all the SMs where it appears. If we define r_i^k to be true if region r_i appears in SM_k , the constraint that each region must appear in at least one SM is modeled by the inequality 4.1 which is then encoded with SAT clauses:

$$\forall_i \exists_k r_i^k = 1. \quad (4.1)$$

So the SAT model will contain a clause for each region r_i to represent all instances of a given r_i in all SMs. In the running example the clauses will be:

$$r_1^1 \wedge r_2^1 \wedge (r_3^1 \vee r_3^2) \wedge (r_4^1 \vee r_4^2) \wedge r_5^2 \wedge r_6^2 \wedge r_7^2.$$

2. Another set of SAT clauses states, for each SM, that if a label on a given edge is removed then also the two regions connected by the edge are removed, i.e., if label l is on the edge connecting regions r_1 and r_2 , the clause template is $(r_1 \vee r_2) \rightarrow l$, i.e., $(\neg r_1 \wedge \neg r_2) \vee l$, i.e.,

$(\neg r_1 \vee l) \wedge (\neg r_2 \vee l)$. In the running example, the clauses for SM_a are the following (l replaced by the actual labels):

$$\begin{aligned} & (\neg r_1^1 \vee a) \wedge (\neg r_2^1 \vee a) \wedge (\neg r_1^1 \vee b) \wedge (\neg r_3^1 \vee b) \wedge (\neg r_1^1 \vee c) \wedge (\neg r_2^1 \vee c) \wedge \\ & (\neg r_2^1 \vee d) \wedge (\neg r_4^1 \vee d) \wedge (\neg r_3^1 \vee e) \wedge (\neg r_4^1 \vee e) \wedge (\neg r_1^1 \vee f) \wedge (\neg r_4^1 \vee f); \end{aligned}$$

the clauses for SM_b are (l replaced by the actual labels):

$$\begin{aligned} & (\neg r_5^2 \vee b) \wedge (\neg r_6^2 \vee b) \wedge (\neg r_6^2 \vee c) \wedge (\neg r_7^2 \vee c) \wedge (\neg r_4^2 \vee d) \wedge \\ & (\neg r_7^2 \vee d) \wedge (\neg r_3^2 \vee e) \wedge (\neg r_4^2 \vee e) \wedge (\neg r_4^2 \vee f) \wedge (\neg r_5^2 \vee f). \end{aligned}$$

3. Finally, we must express the optimization objective: keep the minimum number of labels needed to satisfy the excitation-closure property. This is expressed by Eq. 4.2, where l_j^k is true if there is an instance of label j in SM_k :

$$\min\left(\sum_{\forall k \forall j} l_j^k\right). \quad (4.2)$$

Setting x as the number of labels there are in all SMs (in the running example, $x = \sum l_j^k = 6 + 6 = 12$), this constraint is rewritten as:

$$\sum_{\forall k \forall j} l_j^k \leq x. \quad (4.3)$$

Then Eq. 4.3 is converted into a set of SAT clauses using the library PBLib [19]. The first assignment of x yields a trivially true SAT instance because it corresponds to the initial situation, as stated by Eq. 4.4.

$$\sum_{\forall k \forall j} l_j^k = x. \quad (4.4)$$

Therefore, a solution of Eq. 4.2 can be found by solving a sequence of SAT instances whose clauses are the ones previously defined (clauses to represent regions, clauses encoding the relation between regions and labels, and clauses from the conversion of Eq. 4.3), and where x decreases from the initial largest value down, until when an UNSAT model is reached. The solution of the last satisfiable SAT instance encountered represents the best decomposition of the initial transition system. As a matter of fact, the linear search on x is sped up by transforming it into a logarithmic binary search on x (in the running example, we solve for $x = 12$, $x = 6$, $x = 9$ until we converge for $x = 11$).

In the end, according to the SAT solution, the SMs are restructured by removing arcs and nodes to be deleted, adding merged nodes, and redirecting arcs as appropriate. In the running example, in SM_b we merge the nodes r_3, r_4 into node r_{34} , remove the edge labeled e between the deleted nodes r_3 and r_4 , and redirect to r_{34} the edges that point to r_3 or r_4 .

In this chapter, a specific application of the decomposition algorithm based on the theory of regions was proposed, producing a set of Synchronizing SMs from a transition system. It was shown that by encoding the problem into Maximal Independent Set, each independent set

represents a State Machine. Continuing the search until all regions are used automatically ensures the excitation closure property, even if there is a high probability of having completely or partially redundant components. Indeed, the proposed flow natively includes two minimization algorithms: a greedy algorithm removing completely redundant components and an algorithm based on the merging of adjacent regions for the minimization of the components left after the previous minimization. This final step was proposed by encoding the regions into clauses, allowing for a SAT solver-based solution.

Transition System decomposition into sets of synchronizing FCPNs

The main limitation of the restriction to SMs is their sequential nature (SMs model only causality and choice), i.e., do not capture concurrency, and concurrent events must be split into different components. This chapter mitigates such restriction by allowing concurrent events to be in the same component, under the constraint that the component must be free-choice. With the new approach, the number of components is reduced, trading-off number vs. complexity of components. The step from SMs to FCPNs requires the introduction of a more complex set of constraints on the decomposition process, as explained in detail in the following sections.

In this chapter, two different approaches for the decomposition into a set of synchronized FCPNs will be presented, one based on a sequential search similar to the approach used for SMs, and another novel approach which allows finding directly k FCPNs. The first method was presented at the 31st International Workshop on Logic & Synthesis (IWLS2022) and published at the 25th Euromicro Conference on Digital System Design (DSD2022) [41], the second one at the 26th Euromicro Conference on Digital System Design (DSD2023) [17], together with our software that performs the decomposition.

5.1 Overview

We start by an overview of the method proposed in this chapter and by a simple example to illustrate the main ideas.

Fig. 5.1 shows a TS and a PN net with equivalent behavior. A concurrent system often represents the cooperation of different subsystems that interact through common events. It is interesting to identify and distill the components of the system in a way that they can be visualized and analyzed individually, thus hiding the other components.

This work is an attempt to perform a distillation without prior knowledge of the components, with two goals: (1) extract subsystems with nice structural properties, i.e., easy to visualize and analyze, and (2) guarantee that their composition reproduces the original behavior of the system.

Fig. 5.2 shows an SM-decomposition as proposed in [40]. Due to the inherent concurrency of the system, the decomposition requires four SMs to fully capture the behavior.

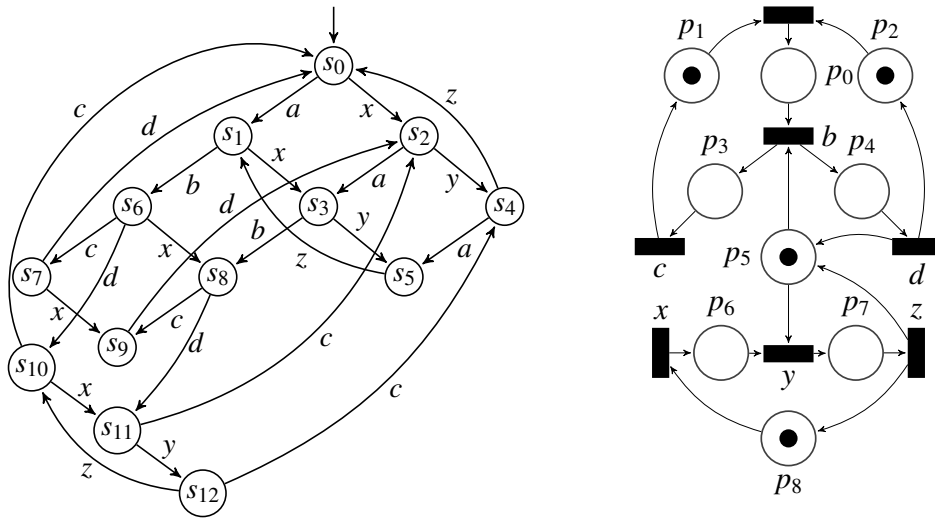


Fig. 5.1: Transition system of Fig. 3.2 and a bisimilar Petri net.

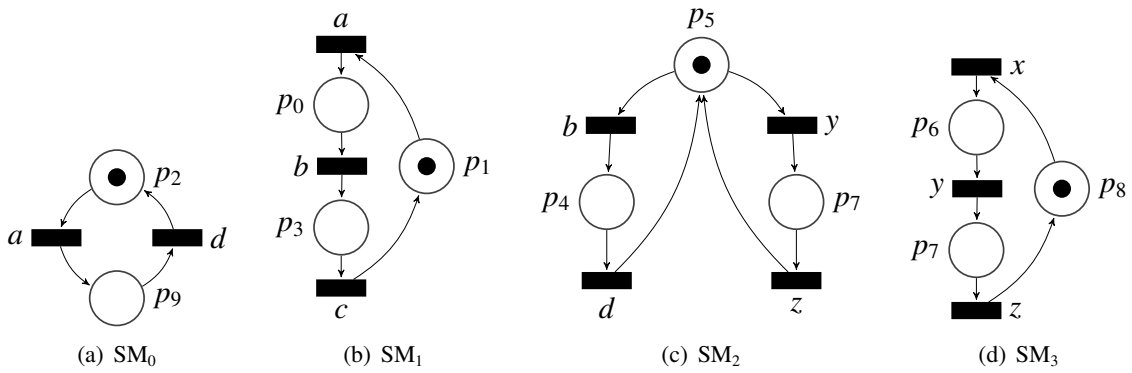


Fig. 5.2: Four SMs distilled from the TS in Fig. 5.1.

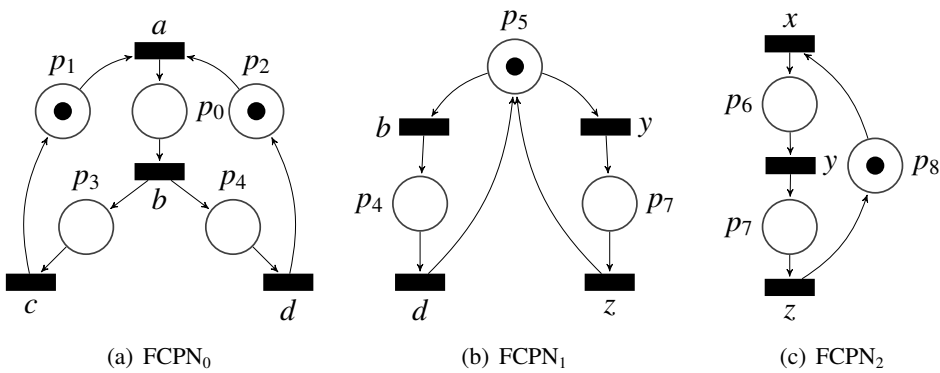


Fig. 5.3: Three FCPNs distilled from the TS in Fig. 5.1.

In this case study, incorporation of concurrency in components is allowed by extending their structural properties. Fig. 5.3 shows three components extracted from the TS in Fig. 5.1. It can be seen that the system models the interaction of two components with alphabets $\{a, b, c, d\}$ and $\{x, y, z\}$, respectively, shown on the left (a) and right (c) of the picture. In the middle, there is an arbitration process (b) that creates a mutual exclusion between $\{b, d\}$ and $\{y, z\}$.

This example illustrates the purpose of this work: extract hidden knowledge from complex systems. In this case, we enforce the components to have structural properties that are formally captured by the concept of FCPN, which combines causality, concurrency, and choice with simple structural rules.

The example also illustrates the main contribution of this work with respect to the SM decomposition proposed in [40]. The use of FCPNs allows the number of components to be reduced while preserving nice structural properties.

5.2 Sequential FCPN search

In this case, as a starting point we could take the flow presented for the state machines, which already satisfies also the constraints for the FCPNs, but we could do even better: since the FCPNs are less restrictive than SMs allowing concurrency inside them, one could improve the previous flow by adopting some changes. First, the properties that are satisfied when a set of regions \widehat{P} represents a Free-choice Petri net are shown:

- *Place connectedness.* If a region $r \in \widehat{P}$, then all incoming and outgoing events are also present in the FCPN.
- *Event connectedness.* If an event e is present, at least one pre-region and one post-region of e must belong to \widehat{P} .
- *Free-choiceness.* if r is a choice present in the FCPN, then r must be the only selected pre-region of its post-events. Formally, if $r \in \widehat{P}$, $|r^\bullet| > 1$, $e \in r^\bullet$, $r' \in {}^\bullet e$, and $r \neq r'$, then $r' \notin \widehat{P}$.

In order to create a restricted set of FCPNs also the following constraints have to be taken into account during the creation of a new FCPN:

- *Maximization function:* given the set of minimal regions, during the creation of a new FCPN, maximize the usage of regions which does not take part of the previously created FCPNs;
- (Optional) *Minimization function:* minimize the number of the total used regions in the new FCPN, keeping constant the maximum number of new used regions.

The purpose of the first functions is to create as few FCPNs as possible: the maximization function forces the creation of largest possible FCPNs; in this way the number of sufficient regions in order to achieve a set of synchronized FCPNs can be reached reducing the number of used FCPNs. Instead, the second function is used to minimize the number of redundant regions during the creation of the new FCPN, keeping constant the result obtained by the previous constraint.

Looking into the implementation of this kind of decomposition, differently from the SMSs, in this case, in order to find a set of synchronized FCPNs a SAT solver is used. Each of the previously described properties was encoded into a set of CNF clauses, and then, by performing a binary search in the range $[0, n]$, instances of SAT problems were solved trying to find a new FCPN with $i \in [0, n]$ new regions, where the maximum number of unused regions is n . Afterward, keeping the obtained number of FCPNs fixed, single FCPN minimization can be performed (merging algorithm).

5.3 Decomposition into k FCPNs simultaneously

In this section, we present a novel method for the decomposition of LTSs into sets of synchronized FCPNs. This method has many similarities to the previous one, but the approach is different since the search is performed simultaneously on k FCPNs. This allows one to encode the excitation-closure property using the set of minimal regions obtained from the LTS. To encode the excitation-closure, for each event, all possible sets of minimal regions that satisfy EC are found. The complexity of this step is $O(2^n)$, where n is the number of pre-regions of the event. Fortunately, the maximum number of pre-regions for an event hardly exceeds a dozen. Once all possible sets of pre-regions that satisfy EC for each event were found, using the CUDD package [42] these sets are encoded into binary decision diagrams (BDDs). Each BDD represents a set of choices of regions to satisfy excitation closure for the given event. It is important to note that each BDD is composed of pre-regions for the given event and not all regions of an LTS. The path from the root to a leaf represents a set of choices of the pre-regions of the given event, which can bring to the satisfaction of excitation closure or not. Taking all paths that satisfy EC, a set of DNF clauses which represents the EC satisfaction can be created. In our case, we need them in CNF form to be fed to a SAT solver, therefore, the complement is taken: all paths that do not satisfy EC. In detail: a path root-leaf can be represented as a clause where each literal is a region which could be *true* or *false*, i.e., according to whether the region is taken or not, for example, $(r_1 \wedge r_2 \wedge \bar{r}_3)$ represents a set of three pre-regions for an event where r_1 and r_2 are taken and r_3 not. This path could bring to an unsatisfied excitation-closure, as many other paths. For the satisfaction of the EC property, it is sufficient to avoid each of these unsatisfiable paths. Given the set of clauses C where each clause represents a set of region assignments that bring to unsatisfied excitation-closure, a set of clauses $\bigwedge_{c_i \in C} \bar{c}_i$ is created to represent that each clause $c_i \in C$ has to be avoided. Being each clause c_i a conjunction of literals, \bar{c}_i becomes a disjunction of literals, bringing the entire set of clauses into CNF form. N.B.: the aforementioned method works with any number of clauses, but we can do better: sometimes there is a special case with only one set of regions satisfying excitation closure, in this case not all possible unsatisfiable paths should be visited, since a set of CNF clauses representing the usage of all involved regions can be directly created. For example, if for a given event e two pre-regions r_1 and r_2 are sufficient to ensure EC, it is possible to create a set of two clauses: $(r_1) \wedge (r_2)$.

Combining the new EC encoding procedure with the previously presented decomposition flow we obtain the following set of constraints which has to be satisfied:

1. *Excitation-closure*;
2. *Place connectedness*: if a region r is used in an FCPN, then all events with incoming or outgoing edges with respect to r must appear in it;
3. *Event connectedness*: structural connection between events and regions; if an event is taken, at least one pre-region and one post-region of this event should be in the FCPN, this constraint guarantees structurally simple FCPNs;
4. *Free-choiceness*: prohibits symmetric choice, asymmetric choice and conflict structures;
5. (Optional) *Event usage*: each event has to be taken at least once: this constraint can be derived from the combination of constraints 1 and 2 but the usage of clauses with only one literal can speed up the computation.

Having the excitation closure encoded into a set of clauses also all remaining constraints for a set of k FCPNs can be encoded, and with $k = 1$ a linear search can start, increasing k until a SAT result is found. Also in this case, once a SAT result is found, a minimization function can be used, minimizing the number of used regions, and keeping the result satisfiable with the fixed number of FCPNs.

As has been seen for SMs, also in case of FCPNs, a satisfiable set of regions could actually represent two (or more) FCPNs, even if a MIS solver is not used anymore. It means that given the solution of the simultaneous method, the disconnected sets of regions can still be divided as separate FCPNs and reach a suboptimal result. To ensure that the result achieved is optimal, it is possible to sign the set of regions representing more than one FCPN as a forbidden result, continuing the search for the optimal solution. This method is very simple, but it is not applied really often and does not produce a drastic performance drop.

5.3.1 Example of k FCPN simultaneous decomposition

Let us see the simultaneous FCPN decomposition performed on the ECTS in Fig. 5.4. The minimal regions are shown in Table 5.1 and excitation sets with pre-regions in Table 5.2. Let us see how the initial SAT clauses for the creation of the FCPNs are encoded.

First of all, the search of minimal sets of pre-regions that satisfy EC for each event is performed. This part is always exhaustive, but fortunately the maximum number of pre-regions for an event is always small. The sets of regions that satisfy EC are shown in Table 5.4. For each set, a Binary Decision Diagram is created (it is possible to see the case related to event f with the truth table in Table 5.3, the derived Binary Decision Tree in Fig. 5.5 and the BDD in Fig. 5.6). Since the SAT clauses should be in CNF form, the cases where an event has only one set of pre-regions satisfying EC were directly transformed into a set of CNF clauses, only in case of f there was the need to use conjunction of all paths unsatisfying EC (Table 5.4). In case of event f there are two sets of regions satisfying EC: $\{r_1, r_7\}$ and $\{r_7, r_8\}$. The paths leading to an unsatisfiable result are the following: $\{\bar{r}_1, \bar{r}_7, \bar{r}_8\}$, $\{\bar{r}_1, \bar{r}_7, r_8\}$, $\{\bar{r}_1, r_7, \bar{r}_8\}$, $\{r_1, \bar{r}_7, \bar{r}_8\}$, $\{r_1, \bar{r}_7, r_8\}$ (see Fig. 5.5 or Fig. 5.6). Each set should

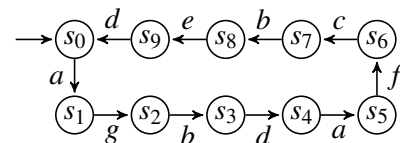


Fig. 5.4: Current example ECTS.

Table 5.1: Minimal regions of TS in Fig. 5.4.

Region	States
r_0	$\{s_0, s_4\}$
r_1	$\{s_3, s_4, s_5, s_8\}$
r_2	$\{s_7, s_8\}$
r_3	$\{s_2, s_7\}$
r_4	$\{s_6\}$
r_5	$\{s_2, s_3, s_9\}$
r_6	$\{s_3, s_8, s_9\}$
r_7	$\{s_1, s_5\}$
r_8	$\{s_2, s_3, s_4, s_5\}$
r_9	$\{s_0, s_1, s_9\}$

Table 5.2: Excitation sets, pre-regions and post-regions for each event of TS in Fig. 5.4.

Event	ES(event)	Pre-regions	Post-regions
a	$\{s_0, s_4\}$	$\{r_0\}$	$\{r_7\}$
b	$\{s_2, s_7\}$	$\{r_3\}$	$\{r_1, r_6\}$
c	$\{s_6\}$	$\{r_4\}$	$\{r_2, r_3\}$
d	$\{s_3, s_9\}$	$\{r_5, r_6\}$	$\{r_0\}$
e	$\{s_8\}$	$\{r_1, r_2\}$	$\{r_5, r_9\}$
f	$\{s_5\}$	$\{r_1, r_7, r_8\}$	$\{r_4\}$
g	$\{s_1\}$	$\{r_7, r_9\}$	$\{r_3, r_5, r_8\}$

Table 5.3: Truth table for EC of event f of TS in Fig. 5.4.

r_1	r_7	r_8	$EC(f)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

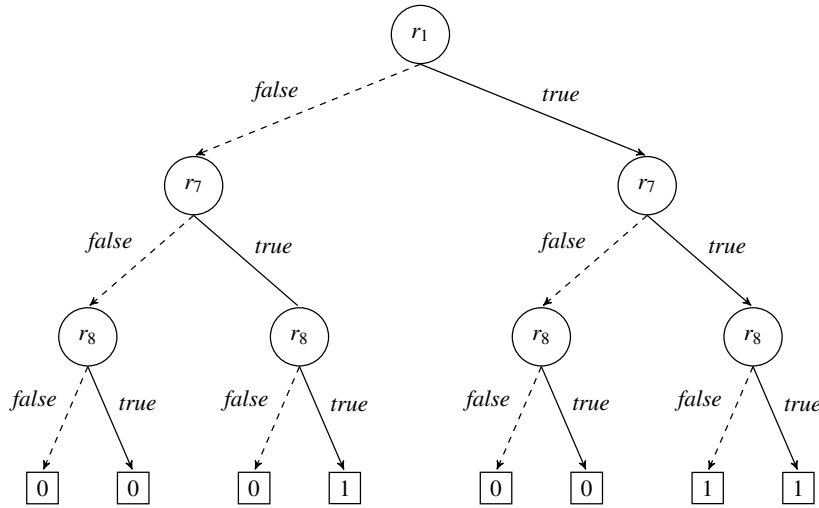


Fig. 5.5: Binary Decision Tree for EC of event f of TS in Fig. 5.4 representing the choice of the regions in the following order: r_1, r_7, r_8 .

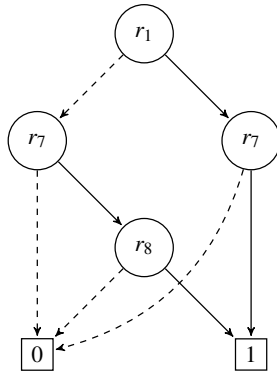


Fig. 5.6: Binary Decision Diagram for EC of event f of TS in Fig. 5.4, following the same variable order of BDT in Fig. 5.5.

Table 5.4: Sets of clauses to satisfy excitation-closure for each event of TS in Fig. 5.4.

Event	Sets of regions	Set of clauses (not simplified)
a	$\{r_0\}$	(r_0)
b	$\{r_3\}$	(r_3)
c	$\{r_4\}$	(r_4)
d	$\{r_5, r_6\}$	$(r_5) \wedge (r_6)$
e	$\{r_1, r_2\}$	$(r_1) \wedge (r_2)$
f	$\{r_1, r_7\}, \{r_7, r_8\}$	$(r_1 \vee r_7 \vee r_8) \wedge (r_1 \vee r_7 \vee \bar{r}_8) \wedge (r_1 \vee \bar{r}_7 \vee r_8) \wedge (\bar{r}_1 \vee r_7 \vee r_8) \wedge (\bar{r}_1 \vee r_7 \vee \bar{r}_8)$
g	$\{r_7, r_9\}$	$(r_7) \wedge (r_9)$

be considered as a conjunction, and there is a disjunction between different sets. Since we want to avoid all these paths, we complement each set by creating a conjunction of clauses: $(r_1 \vee r_7 \vee r_8) \wedge (r_1 \vee r_7 \vee \bar{r}_8) \wedge (r_1 \vee \bar{r}_7 \vee r_8) \wedge (\bar{r}_1 \vee r_7 \vee r_8) \wedge (\bar{r}_1 \vee r_7 \vee \bar{r}_8)$. Since each clause represents an unsatisfiable path, the resultant set of clauses is not minimized, indeed the previous set of clauses can be minimized becoming: $r_7 \wedge (r_1 \vee r_8)$.

Targeting only one FCPN, the following are the clauses encoded for the SAT solver:

- Excitation closure:

$$r_0 \wedge r_3 \wedge r_4 \wedge r_5 \wedge r_6 \wedge r_1 \wedge r_2 \wedge (r_7 \vee r_8) \wedge (r_7 \vee \bar{r}_8) \wedge (r_1 \vee \bar{r}_7 \vee r_8) \wedge r_7 \wedge r_9$$

minimizing, these clauses become

$$r_0 \wedge r_1 \wedge r_2 \wedge r_3 \wedge r_4 \wedge r_5 \wedge r_6 \wedge r_7 \wedge r_9$$

- Place connectedness:

- clauses related to events for which the regions are post-regions

$$(\bar{r}_0 \vee d) \wedge (\bar{r}_1 \vee b) \wedge (\bar{r}_2 \vee c) \wedge (\bar{r}_3 \vee c) \wedge (\bar{r}_3 \vee g) \wedge (\bar{r}_4 \vee f) \wedge (\bar{r}_5 \vee e) \wedge (\bar{r}_5 \vee g) \wedge (\bar{r}_6 \vee b) \wedge (\bar{r}_7 \vee a) \wedge (\bar{r}_8 \vee g) \wedge (\bar{r}_9 \vee e)$$

- clauses related to events for which the regions are pre-regions

$$(\bar{r}_0 \vee a) \wedge (\bar{r}_1 \vee e) \wedge (\bar{r}_1 \vee f) \wedge (\bar{r}_2 \vee e) \wedge (\bar{r}_3 \vee b) \wedge (\bar{r}_4 \vee c) \wedge (\bar{r}_5 \vee d) \wedge (\bar{r}_6 \vee d) \wedge (\bar{r}_7 \vee f) \wedge (\bar{r}_7 \vee g) \wedge (\bar{r}_8 \vee f) \wedge (\bar{r}_9 \vee g)$$

- Event connectedness:

- clauses related to pre-regions of the events:

$$(\bar{a} \vee r_0) \wedge (\bar{b} \vee r_3) \wedge (\bar{c} \vee r_4) \wedge (\bar{d} \vee r_5 \vee r_6) \wedge (\bar{e} \vee r_1 \vee r_2) \wedge (\bar{f} \vee r_7 \vee r_8 \vee r_1) \wedge (\bar{g} \vee r_7 \vee r_9)$$

- clauses related to post-regions of the events:

$$(\bar{a} \vee r_7) \wedge (\bar{b} \vee r_1 \vee r_6) \wedge (\bar{c} \vee r_2 \vee r_3) \wedge (\bar{d} \vee r_0) \wedge (\bar{e} \vee r_5 \vee r_9) \wedge (\bar{f} \vee r_4) \wedge (\bar{g} \vee r_3 \vee r_5 \vee r_8)$$

- FCPN structure: no clauses were produced;

- Event usage:

$$a \wedge b \wedge c \wedge d \wedge e \wedge f \wedge g$$

After the first cycle with the given clauses, an unsatisfiable result was produced. Increasing the total number of FCPNs to two, we introduce the bindings between symbolic regions/events and actual ones. The bindings reduce the total number of clause: for each symbolic region sr_i there is an implication to the different instances of the region r_i^k where i represents the actual region index and k represents the actual FCPN index (0 or 1) since there are two FCPNs. In case

of events, there is the same type of constraint between symbolic events and actual events of the different FCPNs. The constraints for regions are the following:

$$sr_i \implies (r_i^0 \vee r_i^1) \text{ with } i \in [0 - 9]$$

In case of events the constraints are:

$$s(ev) \implies ((ev)^0 \vee (ev)^1) \text{ with } ev \in \{a, b, c, d, e, f, g\} \text{ and } i \in [0, 1]$$

Next are represented the CNF clauses for the second iteration searching a result with two FCPNs:

- Symbolic regions: converting the symbolic region constraints into CNF form the result is:

$$\overline{sr_i} \vee r_i^0 \vee r_i^1 \text{ with } i \in [0 - 9]$$

- Symbolic events: converting the symbolic event constraints into CNF form the result is:

$$\overline{s(ev)} \vee (ev)^0 \vee (ev)^1 \text{ with } ev \in \{a, b, c, d, e, f, g\}$$

- Excitation closure: this constraint remains the same as in the previous cycle but it is applied to symbolic regions because it remains unchanged and represents a property that should be satisfied globally and not on a single FCPN:

$$sr_0 \wedge sr_1 \wedge sr_2 \wedge sr_3 \wedge sr_4 \wedge sr_5 \wedge sr_6 \wedge sr_7 \wedge sr_8 \wedge sr_9$$

- Place connectedness: in this case we have doubled the number of clauses since the entire set has to be proposed for each FCPN:

- clauses related to events for which the regions are post-regions

$$\begin{aligned} & (\overline{r_0^i} \vee d^i) \wedge (\overline{r_1^i} \vee b^i) \wedge (\overline{r_2^i} \vee c^i) \wedge (\overline{r_3^i} \vee c^i) \wedge (\overline{r_3^i} \vee g^i) \wedge (\overline{r_4^i} \vee f^i) \wedge \\ & (\overline{r_5^i} \vee e^i) \wedge (\overline{r_5^i} \vee g^i) \wedge (\overline{r_6^i} \vee b^i) \wedge (\overline{r_7^i} \vee a^i) \wedge (\overline{r_8^i} \vee g^i) \wedge (\overline{r_9^i} \vee e^i) \\ & \text{where } i \in [0, 1] \end{aligned}$$

- clauses related to events for which the regions are pre-regions

$$\begin{aligned} & (\overline{r_0^i} \vee a^i) \wedge (\overline{r_1^i} \vee e^i) \wedge (\overline{r_1^i} \vee f^i) \wedge (\overline{r_2^i} \vee e^i) \wedge (\overline{r_3^i} \vee b^i) \wedge (\overline{r_4^i} \vee c^i) \wedge \\ & (\overline{r_5^i} \vee d^i) \wedge (\overline{r_6^i} \vee d^i) \wedge (\overline{r_7^i} \vee f^i) \wedge (\overline{r_7^i} \vee g^i) \wedge (\overline{r_8^i} \vee f^i) \wedge (\overline{r_9^i} \vee g^i) \\ & \text{where } i \in [0, 1] \end{aligned}$$

- Event connectedness: as in case of place connectedness the constraint is local to each FCPN therefore it should be repeated for each FCPN:

- clauses related to pre-regions of the events:

$$\begin{aligned} & (\overline{a^i} \vee r_0^i) \wedge (\overline{b^i} \vee r_3^i) \wedge (\overline{c^i} \vee r_4^i) \wedge (\overline{d^i} \vee r_5^i \vee r_6^i) \wedge \\ & (\overline{e^i} \vee r_1^i \vee r_2^i) \wedge (\overline{f^i} \vee r_7^i \vee r_8^i \vee r_1^i) \wedge (\overline{g^i} \vee r_7^i \vee r_9^i) \\ & \text{where } i \in [0, 1] \end{aligned}$$

- clauses related to post-regions of the events:

$$\begin{aligned} & (\overline{a^i} \vee r_7^i) \wedge (\overline{b^i} \vee r_1^i \vee r_6^i) \wedge (\overline{c^i} \vee r_2^i \vee r_3^i) \wedge (\overline{d^i} \vee r_0^i) \wedge \\ & (\overline{e^i} \vee r_5^i \vee r_9^i) \wedge (\overline{f^i} \vee r_4^i) \wedge (\overline{g^i} \vee r_3^i \vee r_5^i \vee r_8^i) \\ & \text{where } i \in [0, 1] \end{aligned}$$

- FCPN structure: no clauses were produced;

- Event usage: in this case the constraint is passed to symbolic events since the constraint is global, indeed we don't have repetitions for each FCPN:

$$sa \wedge sb \wedge sc \wedge sd \wedge se \wedge sf \wedge sg$$

This time the result is satisfiable, returning the FCPNs in Fig. 5.7.

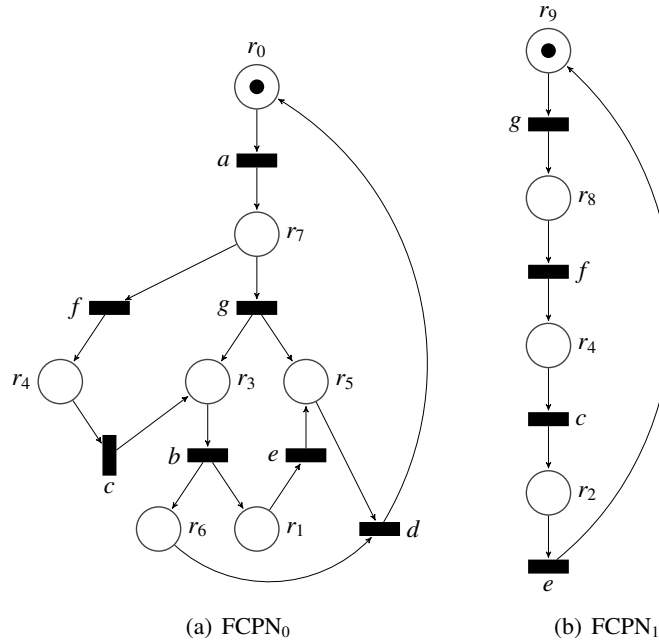


Fig. 5.7: Free-choice Petri nets derived from LTS in Fig. 5.4.

5.4 Additional constraint: safeness for each FCPN

Even if in most cases the decomposition obtains only safe FCPNs, in rare cases unsafe FCPNs may appear. Actually, it is not a problem since the result of the synchronization between FCPNs is guaranteed to be safe, but taking the FCPNs separately this property is not guaranteed (differently from SMs). It means that to satisfy the safeness of each component, additional checks have to be performed.

Different solutions were tested for both sequential and simultaneous approaches (see Section 6.2.2). From these two main approaches arise.

The first is very simple, consists of the simultaneous FCPN search, in the case of at least one unsafe FCPN, SM search is performed. This method is trivial, but it has the best performance, sacrificing the size of the derived model.

The other approach is a little more complex and aims to minimize the number of derived components. This method is based on the sequential FCPN search, adding a counter for each region. If the region is used in an unsafe FCPN, we check whether the region is a post-region of

a fork transition. In the positive case, the counter of the region is increased, and the search for new FCPNs is performed, searching for the maximal number of unused regions, minimizing the sum of the new counters. All performed methods are explained in Section 6.2.2, also showing the experimental results.

5.5 Decomposition optimization

Once a set of synchronized FCPNs is obtained, in the case of sequential FCPN search, as in the case of SMs, we can improve our result trying to minimize the number of the obtained FCPNs; indeed, the greedy algorithm is performed in the same way as in the case of State Machines. In the case of optimal simultaneous FCPN search, there is no need for greedy FCPN removal, since we know that there is no solution with fewer FCPNs: imagine that there exists a solution with n FCPNs where $n < k$, which means that the SAT solver had to return a solution with n . If instead we are not sure about the minimality of the solution because a set of regions probably represents two or more FCPNs, the greedy algorithm could be performed.

In addition to minimization of the number of FCPNs, as we have seen with SMs, we can further reduce the size of each FCPN merging adjacent regions. In this case, optimization is useful independently of the number of FCPN chosen by the decomposition algorithm. In case of FCPNs, regions merge is slightly different from the merge performed on SMs: a set of constraints was added in order to maintain the correct behavior and FCPN property. The following SAT constraints are needed to encode the merge of regions:

1. Preservation of **excitation closure**: all occurrences of each used region can be removed except one occurrence (which preserves excitation closure).
2. Effective merge: removing a label the connected regions are removed.
3. Structural constraint: the merging regions have to be **disjoint** or the intersection of the merging regions has to be a region.
4. Free-choice structure preserving constraint: giving a couple of merging regions r_1 and r_2 where the removing event is e and r_1 is a pre-region for e , r_2 is a post-region for e , if r_1 is a pre-region for an event e' where $e \neq e'$ and r_2 is a pre-region for an event e'' where $e'' \neq e$ and there exists a pre-region of e'' called r_3 i.e. $r_3 \neq r_2$ then the merge between r_1 and r_2 cannot be done in order to preserve free-choice structure.
5. Minimization function: number of events.

The constraints 1, 2 and 5 are the same constraints seen for SMs, let's see in detail the others. Constraint 3 is implicit in the case of SMs because an SM by construction is composed of a set of disjoint regions. In case of FCPNs we could have non-disjoint regions; therefore, this kind of constraint has to be specified explicitly in order to guarantee that the merging regions will create still a valid region: in case of a set of non-disjoint regions sometimes the merge creates a region.

Theorem 5 ([41]). *If r_1 and r_2 are non-disjoint regions, then $r_1 \cup r_2$ is a region iff $r_1 \cap r_2$ is a region.*

Proof. Suppose that r_1 and r_2 are non-disjoint regions and $r_1 \cap r_2$ is also a region. Then $r_1 \setminus (r_1 \cap r_2) = r_1 \setminus r_2$ is a region, since r_1 and $r_1 \cap r_2$ are regions, and $(r_1 \cap r_2) \subseteq r_1$ (by Prop. 1.50, p. 43, [30]). Then $r_1 \setminus r_2$ and r_2 are two disjoint regions and therefore $(r_1 \setminus r_2) \cup r_2 = r_1 \cup r_2$ is a region, since the union of two disjoint regions is also a region (by Prop. 1.64, p. 50, [30]).

Suppose that r_1 and r_2 are non-disjoint regions and $r_1 \cup r_2$ is a region, then $(r_1 \cup r_2) \setminus r_1$ and $(r_1 \cup r_2) \setminus r_2$ are regions, as differences of regions. So $((r_1 \cup r_2) \setminus r_1) \cup ((r_1 \cup r_2) \setminus r_2)$ is a region as a union of disjoint regions. Then $r_1 \cap r_2 = (r_1 \cup r_2) \setminus (((r_1 \cup r_2) \setminus r_1) \cup ((r_1 \cup r_2) \setminus r_2))$ is a region as a difference of regions.

Theorem 5 represents a contribution of our *DSD2022* publication [41], which allows performance improvement during constraint creation. This theorem also shows that using only minimal regions, the result of the merge of two non-disjoint minimal regions may be an invalid region, but after some iterations, having created non-minimal regions, merges between non-disjoint regions become possible.

Even if it is not the case of FCPNs, for generic PNs we can see that Constraint 3 is important also for a second purpose: safeness of the entire composition, considering a composition of unsafe PNs. If we are performing the decomposition allowing unsafe PNs, after the last minimization step the unsafeness can be propagated from single components to the PN composition by merging non-disjoint regions. In Fig. 5.8(a) we can see an unsafe PN (the sequence “ ab ” produces an unsafe marking), but considering it a part of a decomposition which prevents the unsafe sequence by always clearing the place p_2 before the firing of b and other unsafe combinations we can still keep safe the synchronization between PNs. We know that p_1 and p_2 are not disjoint, since both are post-regions for event a which means that there exists at least one state reachable from a which is contained in both p_1 and p_2 and therefore the two regions are not disjoint. Suppose that the merge algorithm removes event b by merging exactly p_1 and p_2 , producing the result in Fig. 5.8(b). In this case, it is impossible to prevent unsafe markings if the firing of event a is allowed since it directly produces an unsafe marking.

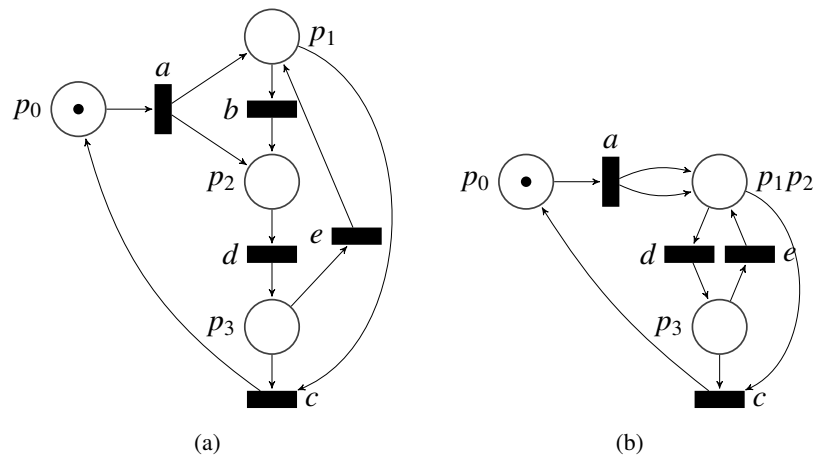


Fig. 5.8: Example with merge of two not disjoint regions.

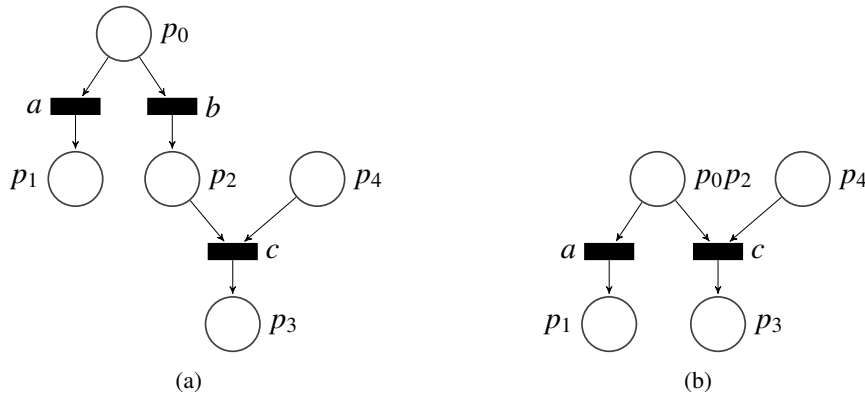


Fig. 5.9: Example of the loss of free-choice property after the removal of event b .

Constraint 4 is fundamental in order to maintain the free-choice property: merging a couple of regions, the PN structure is modified and could contain forbidden structures for the chosen subclass of PNs e.g. a free-choice Petri net which becomes an asymmetric-choice PN (Fig. 5.9).

Theorem 6. *The example in Fig. 5.9 is the simplest case of violation of the free-choice structure. It can be extended to more complex cases with the same structure but more intricate patterns of places/transitions, and it represents the only way to loose the free-choice property by merging.*

Proof. The merge operation does not create new structures but by removing labels it becomes closer to the already existing structure, which means that given an already existing structure in order to create an asymmetric choice, a choice and a join must overlap as a result of a merge. Initially, the choice and the join structures cannot have in common a transition because it would mean that we already have an asymmetric choice. If the two structures do not have anything in common, the merge on one of them does not affect the structure of the other; therefore, in this case the merge is safe. But what happens if the two structures have one or more common places? Considering only the simplest structures, we cannot have three common places (overlapping the two structures, we can have at most two common places). Let's see the other possible cases considering a choice containing places p_1 , p_2 and p_3 and a join containing places p_4 , p_5 and p_6 (Fig. 5.10).

Let us see the cases with initially two common places:

- $p_1 = p_4$ or $p_1 = p_5$ and $p_2 = p_6$ or $p_3 = p_6$: this case is not possible because it represents already an asymmetric choice;
- p_2 and p_3 match p_4 and p_5 : this case is possible and the removal of each event among a , b and c is safe.

Lastly, there are cases with only one common place:

- $p_1 = p_6$: each merge is safe;
- p_1 matches with p_4 or p_5 : this case is not possible because it already represents an asymmetric choice;
- p_6 matches with p_2 or p_3 : each possible merge on a , b or c is safe;

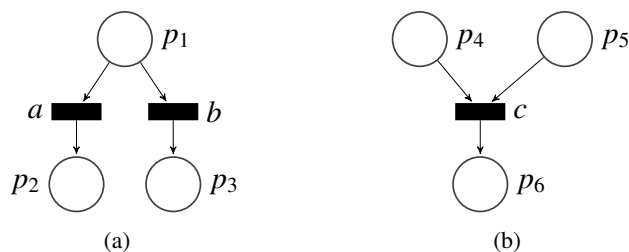


Fig. 5.10: Choice (a) and join (b) used in the proof.

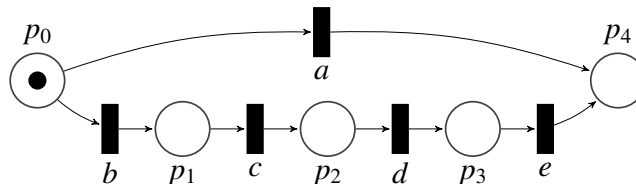


Fig. 5.11: Removing events b , c , d and e as a collateral effect also event a is removed.

- p_2 or p_3 is equal to p_4 or p_5 : merge on c is safe and the same for the event a or b if it is not connected neither to p_4 nor p_5 , otherwise, if the merging transition which does not correspond to c is connected to p_4 or p_5 we have the forbidden case which is represented also in Fig. 5.9.

When the SAT computation is finished, not all merges are performed directly, so an additional check is done in order to avoid the unexpected removal of events: suppose that the SAT solver got as result a sequence of merges where each merge is valid, but the final result is the removal of an entire choice branch, if the choice on the other side does not contain any place but only one or more events, these events would be removed merging the entire set of places (Fig. 5.11). In this case, given a sequence of merging regions, at least one merge is not performed in order to maintain both branches.

5.6 Decomposition into a set of synchronizing ACPNs

The decomposition of a TS/PN into a set of synchronizing Asymmetric-Choice Petri Nets could be seen as a special case of the flow described previously for FCPNs. In this case the flow is the same, except for adding the following property which has to be satisfied so that a set of regions define an ACPN instead of a FCPN:

- Given two events e_1 and e_2 , if $r \in {}^\circ e_1$ and $r \in {}^\circ e_2$ if there is a region $r' \neq r$ being $r' \in {}^\circ e_1$ or $r' \in {}^\circ e_2$ then $|r'^\bullet| = 1$.

This example shows how versatile the decomposition flow is, allowing for plenty of other different constraints for the final model.

In this chapter, we introduced a technique based on the theory of regions to break down a transition system into a set of Synchronizing FCPNs. This process is illustrated by translating

the decomposition constraints into a Boolean Satisfiability problem. The search, based on a SAT solver, continues until the excitation closure property is met, even if there is a significant likelihood of encountering fully or partially redundant components. As with SMs (Sec. 4), the suggested method inherently integrates two minimization techniques: one is a greedy algorithm that eliminates entirely redundant components, and the other focuses on merging neighboring regions to reduce the remaining components after the initial minimization. Furthermore, an enhanced FCPN decomposition version is presented, which incorporates the encoding of excitation-closure into the SAT constraints using BDDs. This refined methodology improves both the average component count and the decomposition duration compared to its predecessor. Furthermore, the possibility of extending the presented approach to Asymmetric-Choice Petri nets is presented.

Experimental results

6.1 SMs

The procedure described in Sec. 3 was implemented and experiments were performed on an Intel Core running at 2.80GHz with 16 GB of RAM. The software is written in C++ and uses *PBLib* [19] for SAT resolution. The resolution of the MIS problem is performed by the *NetworkX* library [18, 43]. For the tests, two sets of benchmarks were used, both from the world of asynchronous controllers: the first set (the same as in [44]), with smaller transition systems is listed in the first rows of Table 6.1 and denoted as “Miscellaneous benchmarks”; the second one containing large transition systems is listed in the second part of Table 6.1, denoted as “Parametrized controllers”. “Parametrized controllers” represents benchmarks from [45, 46] including indeed parametrized controllers *art_m_n*, sequencer *seq_40* and parametrized controllers *pparb_m_n*. Unlike the small and miscellaneous benchmarks, the large set mainly contains controllers with m pipelines (*art_m_n*, *pparb_m_n*), a suitable type of input for our algorithm, being highly concurrent. However, there is also a case with a completely sequential circuit (*seq_40*) in order to show also the worst case where each region contains only one state.

The software used for the synthesis of Petri nets is Petrify¹ [20]. Even if the core of this software did not change for many years, it is still a reference point for PN synthesis using the theory of regions. Genet [47] is the only alternative used nowadays (still based on the theory of regions).

Table 6.1 shows the absolute and relative runtimes of the flow steps: region generation, decomposition into SMs, irredundancy, and place merging. The generation of minimal regions is the dominating operation, taking more than 45% of the overall time spent; it is exponential in the number of events and with the increase of the input dimensions it becomes the bottleneck shadowing the remaining computations (see “Parametrized controllers”). However, it is still possible to decompose quite large transition systems with about 10^6 states and $3 \cdot 10^6$ transitions.

Table 6.2 compares the impact of each optimization step with respect to places and transitions in the examples of “Small-sized set”. The table represents the percentage of places and transitions removed with respect to the execution of the same decomposition without performing minimization steps, which are “Greedy SM removal” and “Places merge”. After the opti-

¹ Version 5.2, May 2019.

Table 6.1: TS statistics and CPU time for each decomposition step including the time spent to generate the regions.

Input	States	Transitions	Events	Regions	Time region generation [s]	Time decomp. [s]	Time Greedy [s]	Time Merge [s]	Total time [s]	Time region gen. [%]	Time decomp. [%]	Time Greedy [%]	Time Merge [%]
alloc-outbound	17	18	14	15	0.00	0.25	0.00	0.06	0.31	0.36	80.36	0.07	19.21
clock	10	10	4	11	0.01	0.20	0.00	0.03	0.24	3.02	85.71	0.13	11.14
dff	20	24	7	20	0.29	0.20	0.00	0.77	1.27	23.28	15.50	0.08	61.14
“espinalt”	27	31	20	23	0.00	0.21	0.00	0.49	0.70	0.37	29.54	0.07	70.02
“fair_arb”	13	20	8	11	0.02	0.20	0.00	0.03	0.25	8.80	80.41	0.04	10.74
future	36	44	16	19	0.03	0.21	0.00	0.11	0.35	9.40	60.35	0.20	30.05
intel_div3	8	8	4	8	0.00	0.23	0.00	0.01	0.24	0.75	94.73	0.04	4.48
intel_edge	28	36	6	27	1.60	0.20	0.00	1.30	3.11	51.58	6.41	0.14	41.86
isend	53	66	15	128	57.67	0.31	0.32	1.04	59.33	97.21	0.51	0.53	1.75
lin_edac93	20	28	8	10	0.00	0.19	0.00	0.01	0.21	1.16	93.38	0.10	5.36
“master-read”	8932	36	26	33	6.71	0.53	0.12	1.03	8.39	80.00	6.28	1.45	12.28
“pe-rcv-ifc”	46	62	16	7	8.80	0.19	0.00	1.21	10.21	86.20	1.90	0.01	11.90
pulse	12	12	6	33	0.00	0.19	0.00	0.01	0.19	0.36	96.62	0.05	2.96
rcv-setup	14	17	10	11	0.00	0.19	0.00	0.04	0.23	1.41	81.36	0.09	17.14
vme_read	255	668	26	44	0.53	0.20	0.01	15.17	15.91	3.36	1.23	0.04	95.37
vme_write	821	2907	30	51	2.78	0.24	0.03	30.03	33.08	8.39	0.73	0.10	90.77
“art_3_05”	4000	11300	30	34	3.57	0.26	0.03	0.04	3.90	91.54	6.67	0.77	1.03
“art_3_10”	32000	93200	60	64	154.96	2.02	0.07	1.18	158.23	97.93	1.28	0.04	0.75
“art_3_15”	108000	317700	90	94	2240.17	10.20	0.77	4.11	2255.25	99.33	0.45	0.03	0.18
“art_3_20”	256000	756800	120	124	11915.93	30.30	1.97	0.65	11948.85	99.72	0.25	0.02	0.01
“art_4_03”	10368	37152	24	30	8.10	0.51	0.09	0.03	8.73	92.78	5.84	1.03	0.34
“art_4_09”	839808	3242592	72	78	4293.87	57.98	4.95	2.07	4358.87	98.51	1.33	0.11	0.05
seq_40	164	164	164	164	0.04	0.23	0.00	1.47	1.75	2.54	13.19	0.01	84.26
“pparb_2_3”	1088	3392	22	42	1.87	0.21	0.11	0.09	2.28	82.02	9.21	4.83	3.95
“pparb_2_6”	69632	321536	34	77	886.54	25.51	30.27	8.32	950.65	93.26	2.68	3.18	0.88
AVERAGE										45.33	31.04	0.53	22.70

mization steps, more than 25% of places and transitions are removed. In case of miscellaneous benchmark set 30% is exceeded: the average percentage is decreased because of the “*art_m_n*” benchmarks, which do not need further minimization after the initial decomposition.

Table 6.3 compares the states and transitions of transition systems vs. the places/transition-/crossing arcs of the Petri nets derived by Petrify (columns under PN), and vs. our product of state machines for the first benchmark set. The number of crossing arcs is reported by the *dot* algorithm of *graphviz* [48] and can be considered as a metric of structural simplicity of the model (i.e., fewer crossings implies a simpler structure). Our results from synchronized state machines have similar sizes compared to those from Petri nets, but they have fewer crossings, which is a significant advantage in supporting a visual representation for “large systems”. Therefore, the plots, in a two-dimensional graphical representation of synchronizing SMs, are substantially more *readable* than the ones of Petri nets: see the inputs *intel_edge* and *pe-rcv-ifc* witnessing that peaks of edge crossings are avoided. The example *master-read* instead is an impressive case of how our decomposition tames the state explosion of the original transition system derived from a highly concurrent environment, since from 8932 states we go down to 8 SMs with an average number of 5 states each.

We also implemented an exact search of all SMs derived from the original TS (Table 6.4), to gauge our heuristics, when it is possible to find a nearly exact solution. The solution is nearly

Table 6.2: Impact of each optimization step in terms of places (P) and transitions (T)

Input	Size			Removed places Greedy [%]	Removed places Merge [%]	Total removed places [%]	Size			Removed transitions Greedy [%]	Removed transitions Merge [%]	Total removed transitions [%]	Final size [P+T]
	Size after decomp. [P]	after Greedy algorithm [P]	Final size [P]				Size after decomp. [T]	after Greedy algorithm [T]	Final size [T]				
alloc-outbound	21	21	16	0.00	23.81	23.81	25	25	21	0.00	16.00	16.00	37
clock	18	14	11	22.22	16.67	38.89	22	18	15	18.18	13.64	31.82	26
dff	50	38	26	24.00	24.00	48.00	72	55	41	23.61	19.45	43.06	67
espinalt	44	44	26	0.00	40.91	40.91	50	50	33	0.00	34.00	34.00	59
fair_arb	12	12	12	0.00	0.00	0.00	18	18	18	0.00	0.00	0.00	30
future	41	29	21	29.27	19.51	48.78	43	30	22	30.23	18.61	48.84	43
intel_div3	12	12	10	0.00	16.67	16.67	13	13	11	0.00	15.38	15.38	21
intel_edge	91	51	26	43.96	27.47	71.43	152	88	64	42.11	15.78	57.89	90
isend	958	165	75	82.78	9.39	92.17	1413	249	166	82.38	5.87	88.25	241
lin_edac93	13	13	13	0.00	0.00	0.00	14	14	14	0.00	0.00	0.00	27
master-read	48	48	38	0.00	20.83	20.83	48	48	38	0.00	20.83	20.83	76
pe-rcv-ifc	47	47	35	0.00	25.53	25.53	66	66	57	0.00	13.64	13.64	92
pulse	7	7	7	0.00	0.00	0.00	10	10	10	0.00	0.00	0.00	17
rcv-setup	18	18	12	0.00	33.33	33.33	22	22	14	0.00	36.36	36.36	26
vme_read	116	102	53	12.07	42.24	54.31	136	121	69	11.03	38.23	49.26	122
vme_write	142	142	67	0.00	52.82	52.82	164	164	81	0.00	50.61	50.61	148
art_3_05	34	34	34	0.00	0.00	0.00	34	34	34	0.00	0.00	0.00	68
art_3_10	64	64	64	0.00	0.00	0.00	64	64	64	0.00	0.00	0.00	128
art_3_15	94	94	94	0.00	0.00	0.00	94	94	94	0.00	0.00	0.00	188
art_3_20	124	124	124	0.00	0.00	0.00	124	124	124	0.00	0.00	0.00	248
art_4_03	30	30	30	0.00	0.00	0.00	30	30	30	0.00	0.00	0.00	60
art_4_09	78	78	78	0.00	0.00	0.00	78	78	78	0.00	0.00	0.00	156
seq_40	164	164	164	0.00	0.00	0.00	164	164	164	0.00	0.00	0.00	328
pparb_2_3	57	40	40	29.82	29.82	59.65	68	46	46	32.35	32.35	64.71	86
pparb_2_6	82	74	69	29.82	29.82	59.65	91	81	76	32.35	32.35	64.71	86
AVERAGE				10.96	16.51	27.47				10.89	14.52	25.41	

exact because the exact computation is performed only by the MIS solver, searching all possible SMs from the given set of region, but the next optimization steps, i.e., removal of redundant SMs and merging algorithm, are still kept approximate.

We compare the times taken by the exact and heuristic SM generation steps: the exponential behaviour of the exact algorithm makes it hardly affordable for about 15 regions and runs out of 16 GB of memory for more than 20 regions (Fig. 6.1). Instead, the approximate algorithms presented in Sec. 3 can handle very large transition systems. Even though the result is not guaranteed to be a minimum one, the irredundancy proce-

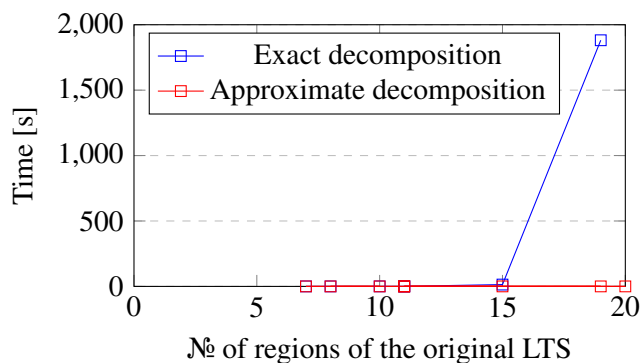
**Fig. 6.1:** Trend of the exact algorithm for the decomposition of a TS compared to the approximate version.

Table 6.3: Number of places (P), transitions (T) and arc crossings (C) of the original transition systems vs. derived Petri nets vs. product of SMs and SM details.

Input	Size comparison									SM details						
	TS		PN			PN*2			Synchronizing SMs			Number of SMs	Avg. places per SM	Avg. alphabet per SM	Places largest SM	Alphabet largest SM
	States	T	P	T	C	P	T	C	P	T	C					
alloc-outbound	21	18	14	14	3	17	18	0	17	21	0	2	8.50	10.50	10	11
clock	10	10	8	5	4	10	10	0	11	15	0	3	3.67	5.00	4	4
dff	20	24	13	14	21	20	20	0	25	41	0	3	8.33	13.33	13	7
espinalt	27	31	22	20	5	27	25	1	29	32	0	3	9.33	11.00	11	13
fair_arb	13	20	11	10	4	11	10	4	12	18	0	2	6.00	9.00	6	6
future	36	44	18	16	1	30	28	0	21	22	0	3	7.00	7.33	13	14
intel_div3	8	8	7	5	2	8	8	0	10	11	0	2	5.00	5.50	6	4
intel_edge	28	36	11	15	22	21	30	56	35	68	1	4	8.50	16.75	13	6
isend	53	66	25	27	106	54	43	5	80	138	4	13	6.31	11.85	12	11
lin_edac93	20	28	10	8	1	14	12	0	13	14	0	3	4.33	4.67	5	6
master-read	8932	36226	33	26	0	33	26	0	38	38	0	8	4.75	4.75	10	10
pe-rcv-ifc	46	62	23	20	96	43	37	13	39	57	2	2	19.00	28.50	21	13
pulse	12	12	7	6	2	12	12	0	7	10	0	2	3.50	5.00	3	6
rcv-setup	14	17	10	10	5	14	14	4	12	14	0	2	6.00	7.00	9	10
vme_read	255	668	38	29	18	41	32	2	50	67	1	9	6.11	7.67	12	13
vme_write	821	2907	46	33	31	49	36	6	57	74	1	11	6.18	7.36	9	11

Table 6.4: CPU time and results of the exact decomposition algorithm.

Input	Decomposition Greedy Merge			States after decomposition	States after greedy	States after merge	Trans. after decomposition	Trans. after greedy	Trans. after merge	Number of regions TS
	[s]	[s]	[s]							
alloc-outbound	14.01	0.0009	0.06	42	21	17	50	25	21	15
clock	0.55	0.0003	0.02	18	14	11	22	18	15	11
fair_arb	0.58	0.0007	0.03	24	12	12	36	18	18	11
future	1881.00	0.0012	0.10	41	29	22	43	30	23	19
intel_div3	0.21	0.0001	0.01	12	12	10	13	13	11	8
lin_edac93	0.33	0.0002	0.01	13	13	13	14	14	14	10
pulse	0.20	0.0000	0.01	7	7	7	10	10	10	7
rcv-setup	0.36	0.0002	0.04	18	18	12	22	22	14	11

ture guarantees a form of minimality, yielding a compact representation that avoids state explosion and exhibits concurrency explicitly.

Another experiment consists in the execution of the exact algorithm for both phases: search of the new SMs and the removal of redundant ones. As previously reported, the execution of the exact algorithm for this task followed by an approximate removal of SMs requires a lot of effort without bringing interesting results. The removal of redundant SMs with an exact algorithm too, provides a lower bound of the decomposition (in terms of number of final SMs), since both steps are performed with an exact algorithm; moreover, it hits the scalability threshold of the exact algorithm, since the computation of many benchmarks does not finish. Therefore, the fully exact computation can be performed only on very tiny benchmarks where the number of SM combinations is very restricted. Notice that for each available result of the completely exact flow, the same number of SMs was found also by the completely approximate one, and by the combination of exact SM search and approximate SM removal.

6.1.1 Creation of a new mixed strategy

In this section are presented just published results at the International Journal of Applied Mathematics and Computer Science (AMCS) [49]. Two experiments were performed in addition to those reported in the previous section.

The first experiment consists of the execution of the approximate SM search followed by an exact algorithm for SM removal (column “Exact algorithm” on Table 6.5). For some benchmarks we got better results compared to a completely approximate approach (*intel_edge*, *vme_write*), but in other cases (*isend*) the computation did not finish due to the high number of SMs available for the removal algorithm (more than 50). Indeed, the complexity of exact removal of redundant SMs is $O(2^n)$, where n is the number of SMs.

The second experiment explored a mixed strategy and represents the main algorithmic improvement compared to the previous version presented in [40]. This approach is based on the number of derived SMs after the approximate search, given that between the two computation steps we know the exact number of derived SMs. The mixed strategy works as follows: let n be the initial number of SMs found with the approximate search; then, when n is “small” we apply the exact removal algorithm whose computational times are affordable; otherwise we apply the approximate removal algorithm. In our experiments, we have chosen $n = 20$. The column “Mixed strategy” in Table 6.5 represents the result of this combination between the exact and approximate algorithm to remove redundant SMs. On average, this combination obtains slightly better results than the previously proposed completely approximate solution (column “Greedy algorithm”), but without significant improvements.

We did not report the computational times for the mixed strategy, as the results do not deviate significantly from the previous approach because we ran the exact algorithm only on small n s, and the computational overhead of SM removal with the approximate algorithm or the controlled exact algorithm represents less than 1% with respect to the overall computational time. Moreover, “Large-sized benchmarks” were not added since from the number of components point-of-view the approximate approach always achieved the optimal result of 5 SMs, without further possible optimization.

Table 6.5: Number of final SMs derived using an approximate algorithm for the search of new SMs and different approaches for the removal of redundant SMs, i.e. greedy, exact and a mixed approach.

	Greedy algorithm (approximate)	Exact algorithm	Mixed strategy
alloc-outbound	2	2	2
clock	3	3	3
dff	3	3	3
espinalt	3	3	3
fair_arb	2	2	2
future	3	3	3
intel_div3	2	2	2
intel_edge	4	3	3
isend	13	-	13
lin_edac93	3	3	3
master-read	8	8	8
pe-rcv-ifc	2	2	2
pulse	2	2	2
rcv-setup	2	2	2
vme_read	9	9	9
vme_write	11	10	10
AVERAGE	4.50		4.38

Table 6.6: Comparison between sequential SM search using previously created heuristics (sequential version) and the new approach (simultaneous version) directly encoding excitation-closure property.

	Number of derived SMs		Decomposition time [s]	
	sequential version	simultaneous version	sequential version	simultaneous version
alloc-outbound	2	2	0.12	0.03
art_3_05	5	5	0.36	0.04
art_3_10	5	5	2.26	8.29
art_3_15	5	5	10.51	45.22
art_3_20	5	5	31.30	146.27
art_4_03	7	7	0.63	1.62
art_4_09	7	7	62.98	456.23
clock	3	3	0.12	0.02
dff	3	3	0.25	0.18
espinalt	3	3	0.21	0.13
fair_arb	2	2	0.12	0.02
future	3	3	0.13	0.04
intel_div3	2	2	0.13	0.02
intel_edge	3	3	0.39	0.38
isend	7	5	0.68	0.40
lin_edac93	3	3	0.11	0.02
master-read	8	8	1.04	1.70
pe-rcv-ifc	2	2	0.35	0.31
pparb_2_3	12	10	0.56	0.28
pparb_2_6	18	17	41.24	47.02
pulse	2	2	0.12	0.02
rcv-setup	2	2	0.17	0.02
seq_40	1	1	0.75	1.42
vme_read	8	8	0.34	0.44
vme_write	10	10	0.75	1.00
AVERAGE	5.12	4.92	6.22	28.44

6.1.2 Simultaneous SM search

We implemented the method presented in Section 5.3 changing the structure constraint from FCPN to SM. In order to ensure only safe SMs, an additional constraint was added: limit to only one initially marked place, since multiple marked places could bring to unsafe markings. Table 6.6 compares the results with the sequential SM search. We did not present the variant of this method for SMs, mainly because it represents exactly the same approach with the variation of only the structural constraint. As for FCPNs, the new method achieves the minimal number of components. Even having this property, in most cases the number of components remains unchanged with respect to the results of the sequential method (only benchmarks in bold have a reduced number of components), showing how the sequential approach is close to the optimal result. The average decomposition time spent is about five times higher with respect to the sequential approach; therefore, the new method cannot directly compete with the previous one, but in the few cases with a reduced number of components, the decomposition time is also reduced with respect to the sequential FCPN search. It means that it is still possible to run both methods in parallel, and probably the first method to finish will also present the better result.

6.1.3 SMs without guarantee the safeness of single components

Although the new method based on excitation-closure encoding did not achieve good results, it is witnessed that the EC property is sufficient to ensure the existence of a safe decomposition, even if the single components are not constrained to be safe. Thanks to a completely different way to search SMs by means of a SAT solver, we may obtain SMs containing multiple initial places (removing the newly added constraint), while keeping the synchronization of the State Machines always safe. We are sure that there exists a safe solution from the combination of [38] and [29]. In [38] is proven that there exists a safe PN derived from any ECTS. In [29] instead, it is shown that each safe and live FCPN can be covered by strongly connected SMs. In our case, we can think of the decomposition as the coverability problem applied to the safe PN guaranteed to exist [38], which is a superset of FCPN, or it is directly a single SM, in which case the result consists of only one component.

Even when trying to find unsafe SMs, the result of the decomposition was still almost always safe. It was possible to find only two benchmarks that contained unsafe SMs. One of these is “*pparb_2_3*”: it was possible to further reduce the number of components, allowing unbounded ones, passing from 10 to 7 SMs. The other benchmark comes from the same set of parametrized controllers “*pparb_m_n*”: “*pparb_2_6*”, passing from 17 safe SMs to 15, containing a combination of safe and unsafe components.

6.2 FCPNs

For the FCPN decomposition we used the same setup as for the SM decomposition, with the same set of benchmarks.

6.2.1 FCPNs without guarantee the safeness of single components

Since for FCPNs, limiting the initial markings is not a solution to guarantee safeness, many tests were made on potentially unsafe FCPNs while keeping the synchronization safe. After that, additional constraints were added to guarantee safeness of the single components, as explained in the next section.

Table 6.7 shows the comparative results between the two main methods to extract FCPNs: sequential vs. simultaneous decomposition and also the optimal version of the simultaneous approach. Differently from SMs, in case of FCPNs the simultaneous approach achieved good results: it was possible to dramatically decrease the average number of components while keeping the decomposition time at the same level as before, even improving it a bit. Having also performed the optimal decomposition, it is possible to see how the simultaneous version is close to the optimal result (1.96 FCPNs on average vs 1.92 FCPNs in the optimal case) and furthermore we can see that the average time to perform the optimal simultaneous decomposition requires more than twice the time required by the simultaneous version only to ensure the optimality (31.88 s vs 73.07 s). All the aforementioned statements show how simultaneous FCPN decomposition achieves the best trade-off between minimality of the number of derived components

Table 6.7: Comparison between sequential and simultaneous FCPN search.

Input	Maximum number of pre-regions for an event	Number of derived FCPNs			Decomposition time [s]		
		sequential version	simultaneous version	simultaneous version (optimal)	sequential version	simultaneous version	simultaneous version (optimal)
alloc-outbound	2	2	2	2	0.03	0.03	0.03
art_3_05	3	2	1	1	0.64	0.46	0.44
art_3_10	3	2	1	1	10.11	8.50	9.87
art_3_15	3	2	1	1	45.42	43.94	45.23
art_3_20	3	2	1	1	151.09	143.00	1142.90
art_4_03	3	2	1	1	1.60	1.40	1.40
art_4_09	3	2	1	1	480.48	539.80	534.60
clock	3	2	2	2	0.01	0.02	0.01
dff	3	3	3	3	0.23	0.27	0.28
espinalt	2	2	2	2	0.07	0.06	0.06
fair_arb	2	2	2	2	0.01	0.02	0.02
future	2	2	2	2	0.01	0.02	0.02
intel_div3	2	2	2	2	0.01	0.02	0.02
intel_edge	4	3	3	3	0.41	0.47	0.39
isend	8	6	4	4	40.97	1.73	1.56
lin_edac93	2	3	2	2	0.01	0.01	0.01
master-read	3	2	1	1	1.85	1.42	1.53
pe-rcv-ifc	2	3	2	2	43.06	0.32	0.36
pparb_2_3	4	3	2	2	0.40	0.20	0.20
pparb_2_6	4	3	4	3	49.78	52.78	85.23
pulse	2	2	2	2	0.01	0.01	0.01
rcv-setup	2	1	1	1	0.02	0.01	0.01
seq_40	1	1	1	1	1.30	1.27	1.26
vme_read	5	3	3	3	0.45	0.46	0.49
vme_write	5	3	3	3	0.88	0.90	0.93
AVERAGE		2.40	1.96	1.92	33.15	31.88	73.07

and the decomposition time. All of this is possible thanks to a reduced number of pre-regions for each event. As we can see in the second column (“Maximum number of pre-regions for an event”) of Table 6.7, it never exceeds a dozen, therefore even if the excitation-closure encoding is exponential, it never causes problems.

Furthermore, from Table 6.7 different statistics can be gathered: looking at the number of FCPNs produced by the optimal simultaneous version, we notice that many cases generate only one FCPN. This result means that the classic PN creation flow based on the theory of regions could also directly create a PN restricted to the Free-choice subclass. There may also be the opposite situation: *isend* creates a set of four FCPNs. This result is caused by a very complex structure, indeed, the PN created directly without the FCPN constraints has 106 crossings, unlikely to be an FCPN having only 25 places (Table 6.3).

The decomposition times for FCPNs have still the same order of magnitude of SM decomposition; often the times are smaller because of the usage of a SAT solver during FCPN creation and the merge procedure. There are still some cases which have a higher order of magnitude than others because of a high number of regions extracted from the initial transition system. But it should be noted that FCPNs are not guaranteed to be safe, meanwhile SMs are.

The time spent for the generation of regions is still the same as the one of SMs, therefore it was not reported in the new table and still represents the bottleneck of the decomposition algorithm.

Due to a bug in the initial versions of code, it was possible to observe a very interesting result. After the creation of the regions only minimal regions are kept, otherwise the computational time grows exponentially. But it was possible to see the power of non-minimal regions: keeping some of them, it was possible to find a solution with only one FCPN directly with the sequential search for one of the presented benchmarks: “*master-read*”, a benchmark which otherwise would produce a solution with two FCPNs. A wider search space would help to improve the decomposition result, but it would dramatically slow down the computation, especially with the simultaneous version because the usage of non-minimal regions would highly increase the maximum number of pre-regions.

6.2.2 Safe FCPNs

Even if the methods shown in the previous section do not guarantee the safeness of the derived FCPNs, actually only three benchmarks were able to produce unsafe FCPNs, which are: “*vme_write*”, “*pparb_2_3*” and “*pparb_2_6*”. Only these benchmarks will be shown to compare the different methods to ensure safeness of the single components.

Different methods were tested to ensure FCPN safeness. Some results can be seen in Table 6.8. This table shows the number of regions of the few cases that produce unsafe FCPNs and the comparison between different methods to create a set of safe FCPNs. The presented heuristics are the following:

Table 6.8: Comparison of different techniques to ensure safe FCPNs or a combination of safe SMs and FCPNs.

Input	Trivial safe search			Simultaneous FCPN search with safeness checks		Sequential safe FCPN search with addition of SMs						decomp. time [s]
	# comp. before greedy alg.	# comp. after greedy alg.	decomp. time [s]	# FCPNs	decomp. time [s]	Before greedy algorithm			After greedy algorithm			
						# comp.	# FCPNs	# SMs	# comp.	# FCPNs	# SMs	
pparb_2_3	-	-	-	6	1257.92	17	2	15	12	1	11	17.13
pparb_2_6	-	-	-	-	-	49	1	48	19	0	19	1591.00
vme_write	7	6	480.31	4	6773.85	5	1	4	5	1	4	1.93

Input	Simultaneous FCPN search followed by sequential SM search						Unsafe simultaneous FCPN search eventually followed by sequential SM search		
	Before greedy algorithm			After greedy algorithm			decomp. time [s]	# comp. (SMs)	decomposition time [s]
	# comp.	# FCPNs	# SMs	# comp.	# FCPNs	# SMs			
pparb_2_3	20	4	16	11	0	11	10.00	10	0.62
pparb_2_6	34	6	28	18	3	15	649.92	16	81.81
vme_write	18	6	12	11	0	11	59.99	10	1.70

- “Trivial safe search”: sequential FCPN search, getting rid of unsafe FCPNs when they are found;
- “Simultaneous FCPN search with safeness checks”: method based on the encoding of excitation closure searching simultaneously k FCPNs, in case of unsafe FCPNs these are marked as forbidden and the search continues until when a set of safe FCPNs which satisfies excitation-closure is found;
- “Sequential safe FCPN search with addition of SMs”: sequential search of FCPNs checking if every newly found FCPN is safe; then safe FCPNs are saved, otherwise after having found an unsafe FCPN a safe SM is searched, and afterwards FCPN search resumes;
- “Simultaneous FCPN search followed by sequential SM search”: combination of the best approaches for FCPNs and SMs to guarantee a set of safe FCPNs, initially k FCPNs are found simultaneously; unsafe components are removed, and the sequential SM search continues;
- “Unsafe simultaneous FCPN search eventually followed by sequential SM search”: after the end of the computation with the simultaneous FCPN search a check on safeness of the derived components is done; if all components are safe the computation stops, otherwise the sequential SM search starts without using information from the previous computation.

In the table, the columns are ordered with the complexity of approaches increasing from right to left, except for the last approach.

Comparing the new methods to achieve a set of safe FCPNs, we can immediately notice how the trivial approach, which discards all unsafe results and continues the search, yields unacceptable results and in most cases cannot find a solution (columns “Trivial safe search” and “Simultaneous FCPN search with safeness checks”). It should also be considered that each of the presented benchmarks has at least 40 regions. “Sequential safe FCPN search with addition of SMs” presents better results from a performance point of view, but it is still too slow and produces too many components. Unexpectedly also “Simultaneous FCPN search followed by sequential SM search” does not achieve good results, even if it represents the best of both the FCPN and SM heuristics. The reason is that reducing the number of components but still being far from the optimal result can have a negative impact on the final number of components. As an example, we observe the initial number of components of “*pparb_2_3*” with “Sequential safe FCPN search with addition of SMs” and “Simultaneous FCPN search followed by sequential SM search”. In the first case we have fewer components with respect to the second one, but after having performed the greedy FCPN removal the situation is reversed. It can be seen also that in all approaches involving FCPNs and SMs, FCPNs were almost totally removed by the greedy algorithm (because of their larger size), leaving mostly SMs in the final result.

Next, a further investigation of the trivial approach was done. Table 6.9 shows the safe and unsafe FCPNs found until reaching a timeout of one or two hours. From the table, it can be seen that by increasing the timeout, no safe FCPN were found, continuing to find only unsafe ones. This means that the optimization objective, which consists in maximization of the usage of regions still not used in accepted safe FCPNs, is not used at all since the number of found FCPNs remains unchanged, resulting in a “blind” search. In order to improve the search, an

Table 6.9: Trivial safe search performed with timeouts.

Input	Trivial safe search with 1 hour timeout			Trivial safe search with 2 hours timeout		
	Unsafe FCPNs found	Safe FCPNs found	Timeout reached?	Unsafe FCPNs found	Safe FCPNs found	Timeout reached?
pparb_2_3	1716	0	yes	2408	0	yes
pparb_2_6	3310	0	yes	4856	0	yes
vme_write	1124	7	no	1124	7	no

additional heuristic was added. The heuristic consists in the introduction of counters: each time a region is used in an unsafe FCPN, the counter is increased. In this way, once a SAT solution with the maximum number of unused regions is found, the search continues, also minimizing the sum of the new counters. The idea is to avoid using always the same regions, which usually occur in unsafe FCPNs. Table 6.10 shows the results obtained trying two different variants: one which resets the counters every time a safe FCPN is found and another without resetting the counters. The table shows a huge improvement in the case of “*pparb_m_n*” benchmarks, but on the other side “*vme_write*” had a performance drop. During the execution of “*vme_write*” a good performance improvement was noticed searching first safe FCPNs, but after some iterations the search got stuck, showing that the new heuristic ceased to be productive. Consequently, at the end of the computation, we have more unsafe FCPNs with respect to the basic trivial safe search.

Table 6.10: Results achieved on sequential safe search with the usage of counters.

Input	Trivial safe search with counters and counter reset				Trivial safe search with counters			
	Unsafe FCPNs found	Safe FCPNs found	FCPNs after greedy decomp.	Decomp. time [s]	Unsafe FCPNs found	Safe FCPNs found	FCPNs after greedy decomp.	Decomp. time [s]
pparb_2_3	84	9	7	21.98	16	7	7	6.22
pparb_2_6	363	19	12	3976.76	33	11	11	276.41
vme_write	1287	7	7	1260.65	3571	6	6	9964.57

Since the new approach did not turn out to be very precise, finding a lot of unsafe FCPNs in the example “*vme_write*”, a new version of the heuristics was proposed. Given that counting all regions used in an unsafe FCPN counts also regions probably not involved in the reachability of unsafe markings, there was the need to count only the regions that are more involved. The idea is to count only the post-regions of fork transitions. In particular, we considered the fork transitions of the newly found unsafe FCPNs and not the entire set of regions, since two concurrent flows could start, but the flows would split into different FCPNs without creating boundedness problems. Table 6.11 shows the results achieved with this last change, resulting in an acceptable approach for all available benchmarks if counter reset is not performed.

From the different approaches, it turns out that the fastest way to ensure a set of safe FCPNs is to use the simultaneous FCPN decomposition, checking at the end the safeness of each FCPN. In the case of at least one unsafe FCPN, it’s faster to perform the previously presented sequential SM decomposition, with SMs also being FCPNs. This approach suffers from the minimization point of view, since at the end we search for a set of SMs. If minimization is important, the se-

Table 6.11: Results achieved on sequential safe search with the usage of improved counters.

Input	Trivial safe search with improved counters and counter reset				Trivial safe search with improved counters			
	Unsafe FCPNs found	Safe FCPNs found	FCPNs after greedy decomp.	Decomp. time [s]	Unsafe FCPNs found	Safe FCPNs found	FCPNs after greedy decomp.	Decomp. time [s]
pparb_2_3	141	6	6	58.28	41	5	5	19.67
pparb_2_6	568	17	9	16908.00	99	10	10	1092.79
vme_write	214	12	8	38.99	40	7	7	4.60

quential FCPN search with the addition of the improved counters seems to be the best solution, reporting acceptable times for each benchmark with a reduced number of components.

How much does the safeness of single components cost? All previously presented tables represent the results in the worst-case scenario where the simultaneous FCPN search finds at least one unsafe FCPN. Actually, these cases are very rare, indeed, in Table 6.12 it is possible to see the actual impact of the safeness check, and eventually the SM search, on the entire set of benchmarks. This table shows two approaches to guarantee a set of safe FCPNs: one oriented on the performance, actually searching SMs, and the second approach (size oriented) trying to

Table 6.12: Comparison between the best approaches to decompose an LTS into a set of synchronizing FCPNs with and without guarantee on the safeness of the components.

	Unsafe FCPN search		Safe FCPN search (performance oriented)		Safe FCPN search (size oriented)	
	# components	decomp. time [s]	# components	decomp. time [s]	# components	decomp. time [s]
alloc-outbound	2	0.03	2	0.03	2	0.03
art_3_05	1	0.46	1	0.46	1	0.46
art_3_10	1	8.50	1	8.50	1	8.50
art_3_15	1	43.94	1	43.94	1	43.94
art_3_20	1	143.00	1	143.00	1	143.00
art_4_03	1	1.40	1	1.40	1	1.40
art_4_09	1	539.80	1	539.80	1	539.80
clock	2	0.02	2	0.02	2	0.02
dff	3	0.27	3	0.27	3	0.27
espinalt	2	0.06	2	0.06	2	0.06
fair_arb	2	0.02	2	0.02	2	0.02
future	2	0.02	2	0.02	2	0.02
intel_div3	2	0.02	2	0.02	2	0.02
intel_edge	3	0.47	3	0.47	3	0.47
isend	4	1.73	4	1.73	4	1.73
lin_edac93	2	0.01	2	0.01	2	0.01
master-read	1	1.42	1	1.42	1	1.42
pe-rcv-ifc	2	0.32	2	0.32	2	0.32
pparb_2_3	2	0.20	10	0.62	5	19.67
pparb_2_6	4	52.78	16	81.81	10	1092.79
pulse	2	0.01	2	0.01	2	0.01
rcv-setup	1	0.01	1	0.01	1	0.01
seq_40	1	1.27	1	1.27	1	1.27
vme_read	3	0.46	3	0.46	3	0.46
vme_write	3	0.90	10	1.70	7	4.60
AVERAGE	1.96	31.88	3.04	33.09	2.48	74.41

find only FCPNs at the expense of the decomposition time. Since searching for safe FCPNs represents the addition of a new constraint, there is a drawback, which can be the number of derived components or the decomposition time. Keeping the decomposition time more or less the same as in the case of unsafe FCPNs, the average number of derived components is increased from 1.96 to 3.04. Searching a solution with a reduced number of components, the decomposition time is more than doubled, scoring an average of 2.48 components. As expected, we observe an inverse relationship between the number of components and the decomposition time.

6.2.3 Reset of the learned clauses

During the search of new FCPNs sometimes it happens that a solution represents a set of FCPNs, therefore, the smallest nets are forbidden in order to allow finding the best result with a single FCPN. Since the forbidden results could represent FCPNs necessary to achieve excitation-closure, once a single FCPN is found the clauses related to smaller FCPNs are forbidden, again allowing us to find a solution with one of these. Table 6.13 shows the experiment that runs the sequential FCPN search without resetting the learned clauses. This experiment was done to see if there is an improvement in the decomposition time, since an extended set of clauses should

Table 6.13: Comparison between standard sequential FCPN search and a variation of the same algorithm without resetting the set of learned clauses related to the FCPNs forbidden because part of a solution with multiple FCPNs.

	Number of derived FCPNs		Decomposition time [s]	
	(standard version)	(without clause reset)	(standard version)	(without clause reset)
alloc-outbound	2	2	0.03	0.03
art_3_05	2	2	0.64	0.53
art_3_10	2	2	10.11	8.80
art_3_15	2	2	45.42	49.24
art_3_20	2	2	151.09	144.36
art_4_03	2	2	1.60	1.52
art_4_09	2	2	480.48	467.64
clock	2	2	0.01	0.01
dff	3	3	0.23	0.23
espinalt	2	2	0.07	0.07
fair_arb	2	2	0.01	0.01
future	2	2	0.01	0.04
intel_div3	2	2	0.01	0.01
intel_edge	3	3	0.41	0.39
isend	6	6	40.97	39.47
lin_edac93	3	3	0.01	0.01
master-read	2	2	1.85	1.62
pe-rcv-ifc	3	3	43.06	17.43
pparb_2_3	3	3	0.40	0.37
pparb_2_6	4	3	55.83	46.00
pulse	2	2	0.01	0.00
rcv-setup	1	2	0.02	0.02
seq_40	1	1	1.30	1.32
vme_read	3	5	0.45	1.57
vme_write	3	3	0.88	0.66
AVERAGE	2.44	2.52	33.40	31.25

converge faster to a solution. Moreover, the following limit case may occur: the inability to achieve EC because all possible FCPNs that extend the usage of the unused regions are forbidden. Only when the deadlock situation was reached, the learned clauses were reset. Contrary to expectations, the decomposition time remains similar to the previous version, but we had three cases with a different number of final FCPNs: “*pparb_2_6*” reduced the number of FCPNs from 4 to 3, “*rcv-setup*” and “*vme_read*” instead has increased the number of components, from 1 to 2 and from 3 to 5, respectively. This test has shown that the new variation of the standard sequential FCPN search does not exhibit significant improvements, when avoiding resetting the learned clauses.

Multiple Synchronized FSMs

MSFSM model

Developing my research in the field of Petri net decomposition, I became interested in the MSFSM model, proposed by Pavlos Mattheakis [23, 50]. This model aims to bridge the Petri net world with the one of Interacting FSMs, avoiding in the meantime the usual state explosion of FSMs, by means of a polynomial time algorithm with two constraints on the initial model: the PN has to be Free-Choice or Asymmetric-Choice and the labels have to respect the syntax of the STG. This is a good prerequisite for a deeper analysis of this model, especially having to deal with FCPNs and ACPNs, explored in the first part of the thesis. Additionally, the MSFSM model allows for analysis centered on the FSM content rather than on FSM synchronizations, as it expresses synchronizations using synchronization primitives. Next, the notions of Interacting FSMs, MSFSMs and synchronization primitives (Wait State and Transition Barrier) will be shown.

Definition 37 (Interacting FSMs). *A set of Interacting FSMs $\{M_1, M_2, \dots, M_n\}$ is a system in which each FSM communicates with other FSMs, by exchanging inputs and outputs.*

Definition 38 (Multiple Synchronized Finite State Machine (MSFSM) Set [23]). *An MS-FSM set, MS , is a five-tuple $(I, O, M, \Delta, \Lambda)$, where I is a finite, nonempty set of global inputs, O is a finite, nonempty set of global outputs, M is a finite nonempty set of N FSMs M_i , with state sets S_i and corresponding local output sets, $\lambda_i : I \times S_i \rightarrow O_i$ (if M_i is a Mealy machine), or $\lambda_i : S_i \rightarrow O_i$ (if M_i is a Moore machine), Δ is a set of next state functions, one per FSM i , where $\Delta_i : I \times O_1 \times O_2 \times \dots \times S_i \times \dots \times O_N \rightarrow S_i$, and $\Lambda : I \times O_1 \times O_2 \times \dots \times S_i \times \dots \times O_N \rightarrow O$ is the global output generation function.*

The FSMs change their state or produce the output function (according to whether they are Mealy or Moore machines) relying on global inputs and state-dependent outputs of other FSMs of the set. Local outputs of the FSMs are combined to produce the global output of the system. The initial state of the MSFSM set corresponds to the set of initial states of its machines.

Fig. 7.1 shows an example of a synchronous architecture based on MSFSM. Signals In and Out represent inputs and outputs of single FSMs. We can notice that sometimes the output of an FSM can represent the input of another. PIs and POs are primary inputs and primary outputs, which actually represent the interface between the controller and the environment.

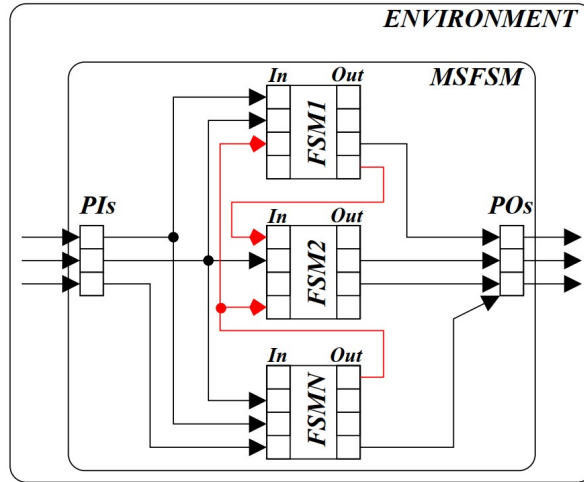


Fig. 7.1: Example of synchronous MSFSM architecture [23].

Definition 39 (MSFSM Wait State [23]). In an MSFSM set, MS , a Wait State W is a state of a machine M , which belongs to MS , where the next state function for state W , $\Delta_i(W)$, depends on a combinational function f of the global inputs I , and on a product of local outputs of a subset $J \subseteq \{1, \dots, N\}$ of the FSMs of MS , i.e. is of the form: $\Delta_i(W) = f(I) \cdot \prod_{k \in J} O_k$.

A Wait State can also be represented as a tuple (s_i, f, w_{δ_i}) where s_i represents the current state, f is the combinational function of the global input I and w_{δ_i} is the set of awaited states. This representation omits the state reachable from s_i by f since it is used essentially to represent the relations between the waiting and the awaited states.

Fig. 7.2 illustrates an example of Wait State. It can be seen that the first FSM waits the second one. In particular, t_1 cannot be activated until the second FSM has reached the state s_4 . It is important to notice that since the Wait State definition is based on the outputs generated by reaching a certain place, it means that the output is high when the FSM is currently in the awaited state, otherwise it is “low” not allowing the activation of the waiting transition. Looking to the example: if FSM 2 reaches s_5 without having activated t_1 when FSM 2 was in s_4 , then FSM 1 is not able to activate t_1 anymore since the output of s_4 is “low”. Furthermore from the WS definition we notice that each awaited output has its own index j representing one of the different FSMs, which means that it is possible to wait for more than one state but that each state should belong to a different FSM. Of course a transition

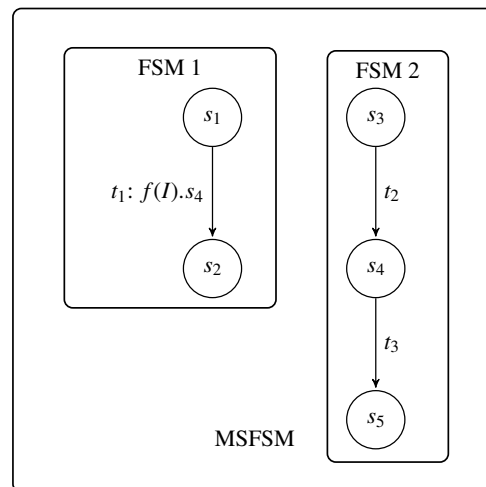


Fig. 7.2: Wait State example.

cannot wait for a state of the same FSM, since it would represent a transition which is never activated.

Definition 40 (MSFSM Transition Barrier [23]). *In an MSFSM set, MS , a Transition Barrier T , is a set of Wait State transitions of different FSMs of MS , with identical combinational function $f(I)$, and an equivalent output product in the respective Δ_i 's, i.e. each transition of the synchronization barrier T and corresponding wait state local output product, $\prod_{j \in N} O_j$, includes all other wait states of T .*

Fig. 7.3 illustrates an example of *Transition Barrier* that represents a set of *Wait States* that are activated simultaneously. It can be seen that the two FSMs are mutually dependent, since t_1 is activated by s_3 and t_2 is activated by s_1 . Since states s_1 and s_3 share the same combinational function $f(I)$, one FSM waits for the other, and vice versa, furthermore they have to be activated simultaneously, e.g. we pass from a “global state” (s_1, s_3) to (s_2, s_4) without the possibility of having combinations derived from activation of $f(I)$ in only one of the FSMs, in our case (s_1, s_4) and (s_2, s_3) .

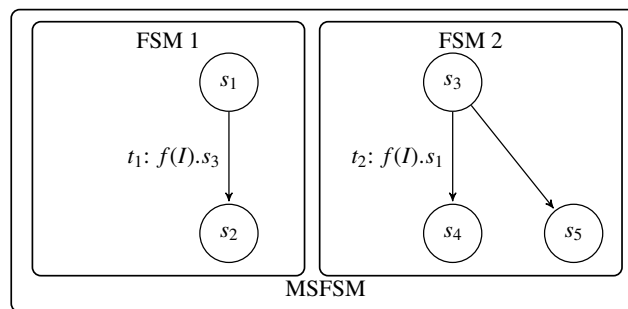


Fig. 7.3: Transition Barrier example.

The MSFSM model separates the synchronization between the FSMs from their functionality, and it could be usefully adopted as a control model for the synthesis of Petri Nets and verification of concurrent systems.

7.1 Other models of concurrency

Before presenting the MSFSM-based decomposition flows, we will provide a brief overview of comparable models.

7.1.1 Communicating Sequential Processes (CSP)

In 1978, one of the earliest models introduced for modeling concurrent systems was “Communicating Sequential Processes,” commonly referred to by its abbreviation CSP [51]. CSP had many extensions over time, among them:

- Timed CSP or TCSP [52]: integration of the real-time concept in CSP;

- CSP# [53]: introduction of shared variables in CSP;
- Probabilistic CSP or PCSP [54]: models systems with probabilistic behaviours;
- CSP-CASL [55]: combination of Common Algebraic Specification Language (CASL) with CSP;
- CSPm [56]: a machine readable version of CSP compatible with tools such as FDR [57];
- Circus [58]: based on Z^1 notation and allows the description of states and behaviours of the systems;
- CSP-OZ [60]: combination of CSP with Object- Z^2 allowing the representation of both state-based and event-based behaviours.

In each of the listed variants of CSP the synchronizations between concurrent processes are obtained through parallel composition.

7.1.2 Synchronous Languages and their representations

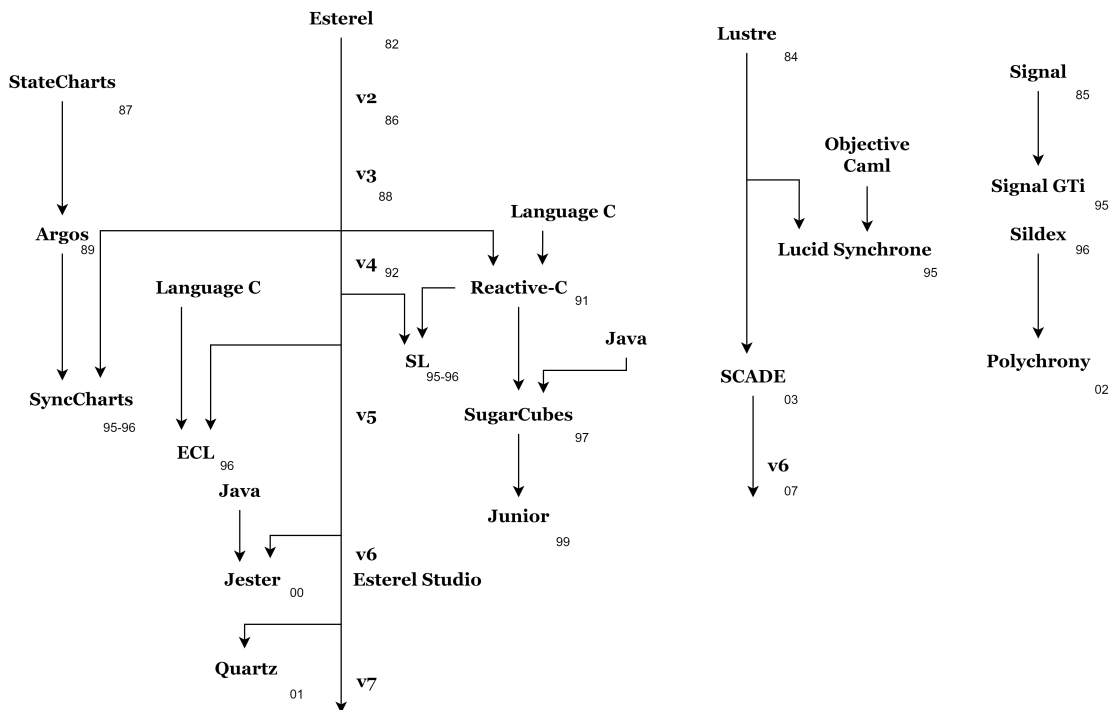


Fig. 7.4: Evolution of the synchronous approach.

¹ The Z notation (often simply referred to as “Z”) is a formal specification language used for describing and modeling computing systems. In the Z notation there are two languages [59]:

- **Mathematical Language:** The mathematical language is used to describe various aspects of a design: objects and the relationships between them by using propositional logic, predicate logic, sets, relation, and functions.
- **Schema Language:** The schema language is used to structure and compose descriptions: collecting pieces of information, encapsulating them, and naming them for reuse.

² Object-Z is an extension of the Z formal specification language that introduces object-oriented features [61].

During the years different synchronous languages were developed. Fig. 7.4 represents the dependencies between the most important synchronous languages, their evolution, and representation models. This figure is an expansion of the representation presented in [62]. From the different languages mentioned in the figure we can select two with a visual representation similar to MSFSMs: SyncChart [63] and Argos [64], a graphical synchronous language for reactive systems representation, where a system is reactive if there is a continuous interaction between the system and the surrounding environment [65]. Also SCADE (Safety Critical Application Development Environment) [66, 67] exhibits a similar visual representation, even if it is a software development environment allowing different levels of precision during the modeling process. Next, these models will be compared with MSFSMs.

SyncChart is a formalism that is used to specify and design reactive systems. This model is an extension of StateCharts [68], which is a visual formalism introduced by David Harel in the 1980s to design reactive systems. SyncCharts have been introduced as a graphical form of the Esterel language [69] that inherits its mathematical semantics [70].

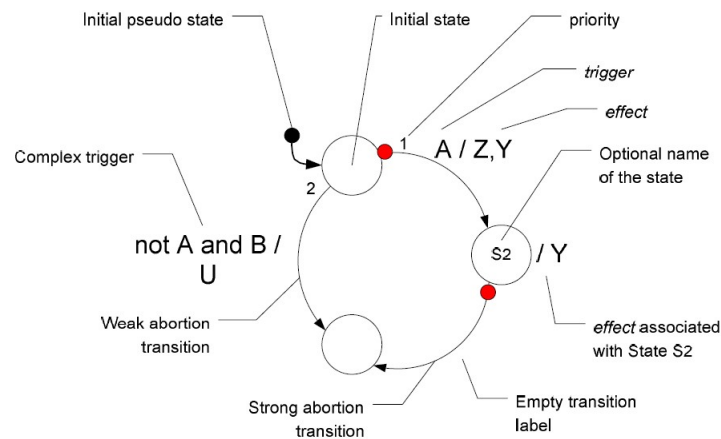


Fig. 7.5: SyncChart FSM notations [71].

Fig. 7.5 represents the description of the different parts of a basic SyncChart. Here, we can notice that each state has an effect associated with it; the same happens also in the case of MSFSMs but the effect represents the condition of reaching the state, without an explicit representation. This effect is not represented explicitly since it is valid for each state, and we will see that it is used for the extraction of synchronization primitives. A substantial difference from MSFSMs is the presence of two types of transitions in SyncCharts: weak abortion transitions and strong abortion transitions. It is possible to combine these two features of the model thanks to model preemption: strong abortion transitions produce as output the effect associated to the arrival state; in case of weak abortion transitions also the effect of the state from which the transition starts is activated.

Fig. 7.6 represents an example of hierarchy in the SyncChart model, where the transition containing the green triangle allows to exit from a SyncChart and go to a state in the parent

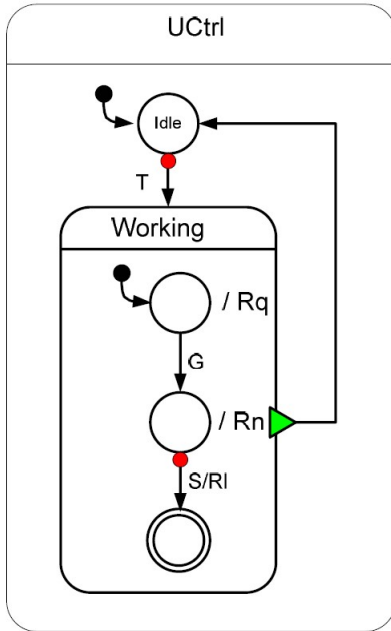


Fig. 7.6: Hierarchy representation example with SyncCharts [71].

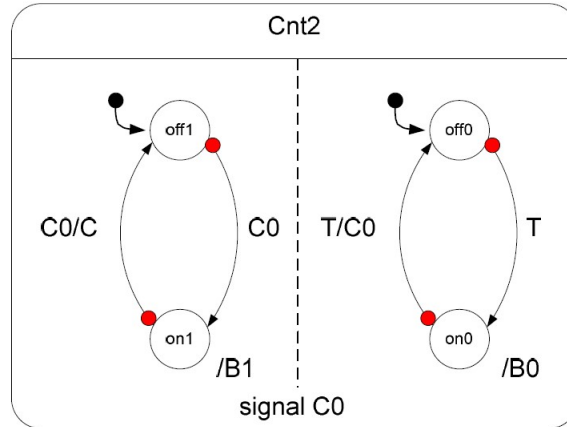


Fig. 7.7: Concurrency representation example with SyncCharts (2-bit binary counter) [71].

SyncChart. The only form of hierarchy supported by MSFSMs is like the one, e.g., in Fig. 7.3, which shows an MSFSM model containing different FSMs.

but actually this is the only type of hierarchy which can be represented by MSFSMs.

In Fig. 7.7 we can observe the representation of concurrency with SyncCharts, where two concurrent FSMs are divided by a dashed line. Here, we can see the main difference between the two models and understand why SyncCharts cannot be seen as an extension of MSFSMs with a different representation. Taking into account two concurrent FSMs, SyncCharts synchronize the two with *parallel composition*. In the case of MSFSMs, the synchronization constraint is much tighter: the synchronization is done on the transition with the same $f(I)$ input function (it is not sufficient to contain the same signal events as in the synchronizations of Fig. 7.7, where the synchronizing $f(I)$ function is always different since C_0 on the right FSM is always an output, where the left FSM was designed in cascade with the right FSM). Here we can see a first limitation of the MSFSM model: the concurrent FSMs are supposed to be synchronized by common input signals, which is not the case of the current example.

7.1.3 Why MSFSMs?

MSFSMs at first sight could be seen as a simplified version of SyncCharts, but it is not so. SyncCharts are a very versatile model, able to represent *preemption*, *hierarchy* and *concurrency*. Excluding the aspects of preemption and hierarchy, which are beyond the modeling capabilities of MSFSMs, the representation of concurrency is similar to that of MSFSMs. However, the method employed for synchronizing the FSMs differs significantly.

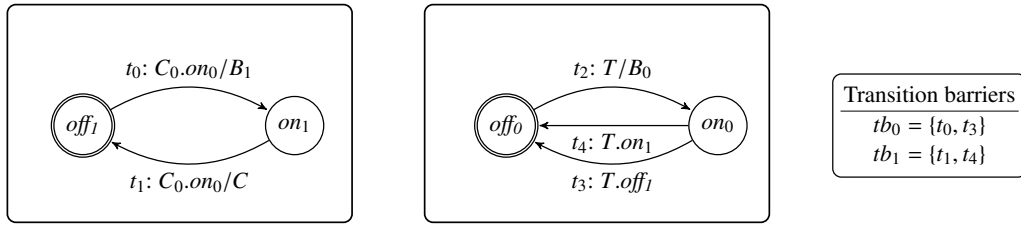


Fig. 7.8: 2-bit binary counter MSFSM.

If we would represent the same example of Fig. 7.7 with an MSFSM, we would get the result in Fig. 7.8. We notice immediately an additional transition in the right FSM caused by a different synchronization mechanism based on the Wait State and Transition Barrier synchronization primitives. To ease understanding, we show next the evolution of the MSFSM in Fig. 7.8:

1. We start from $\{off_1, off_0\}$.
2. Only transition t_2 can fire.
3. We arrive in $\{off_1, on_0\}$.
4. Only the transition barrier $tb_0 = \{t_0, t_3\}$ can be activated.
5. We arrive in $\{on_1, off_0\}$.
6. Only the transition t_2 can be activated.
7. We arrive in $\{on_1, on_0\}$.
8. Only the transition barrier $tb_1 = \{t_1, t_4\}$ can be activated.
9. We arrive to the initial situation: $\{off_1, off_0\}$.

This is the main difference between MSFSMs and SyncCharts. Moreover, all previous models, including Communicating Sequential Processes with their variants and the models of Synchronous Languages, use synchronizations based on parallel composition: the most intuitive way to synchronize FSMs and processes. MSFSMs use a less intuitive way to synchronize FSMs, inspired from Free-Choice Petri nets, introducing additional constraints, but at the same time admitting a polynomial flow for circuit synthesis. Since Petri nets are the core model of my thesis, this combination of positive and negative aspects was a sufficient motivation for me to analyze more in depth the MSFSM model.

7.2 Creation of synchronous circuits with MSFSMs

In [23] a flow for the implementation of synchronous circuits was presented, starting from a PN specification. This flow is based on the creation of an MSFSM model, but the model itself is not enough, since the design of a synchronous circuit requires the addition of a *clk* signal, so that both PN and MSFSM models are synchronized with the clock. Next, a transformation flow from PNs to MSFSMs will be presented, and later the integration of the clock signal will be shown.

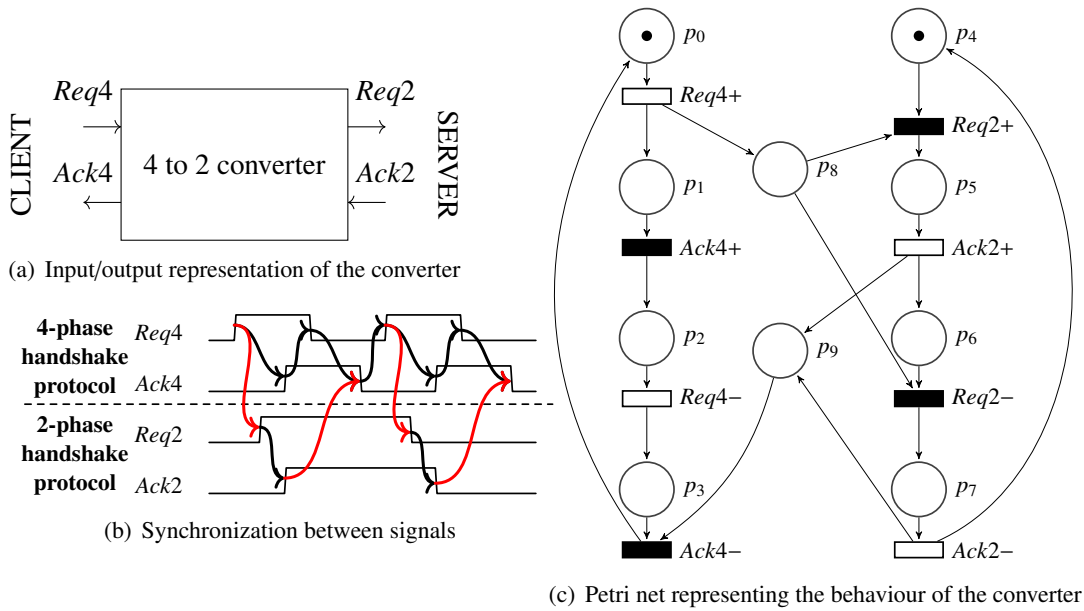


Fig. 7.9: 4-phases to 2-phases handshake converter.

7.2.1 PN to MSFSM transformation flow

In this section, the conversion flow from Petri nets to MSFSMs will be discussed. In particular, two subclasses of Petri nets are considered: Free-Choice nets and its extension to Asymmetric-Choice, whereas for arbitrary Petri nets the synthesis algorithms would suffer from high complexity. In both cases, only safe Petri nets are considered.

To explain this transformation flow, the 4-phase to 2-phase handshake protocol converter in Fig. 7.9 will be analyzed. 4-phase handshake converter, as indicated by its name, is based on four key steps:

1. $Req4+$: client requests the data transfer;
2. $Ack4+$: server has made available the data and acknowledges receipt of the initial request;
3. $Req4-$: client confirms the completion of the data transfer;
4. $Ack2-$: server then informs client of its readiness for the next data transfer.

2-phase handshake protocol involves two primary phases, with an optional reset phase:

1. $Req2+$ (or $Req2-$): client requests the data changing $Req2$ value;
2. $Ack2+$ (or $Ack2-$): server makes available the data and acknowledges client about it, changing $Ack2$ value.

The benefit of the 2-phase protocol lies in its ability to initiate a new request without needing to reset both the Req (Request) and Ack (Acknowledgement) signals to a low level. In contrast, the 4-phase protocol requires a return to the initial state before starting a new transfer. As a result, the 2-phase protocol is faster but less robust.

The controller illustrated in Fig. 7.9 facilitates communication between these two protocols by adapting the 4-phase sequence to align with the 2-phase protocol.

In this conversion, the input signals are represented as white PN transitions, and the output signals are represented as black ones (Fig. 7.9(c)). We can notice how *Req* is an input signal for the sender and output signal for the receiver, vice versa for *Ack* signal (Fig. 7.9(a)). Analyzing Fig. 7.9(b), we can notice black and red arrows. Black arrows represent the communication between the signals *Req* and *Ack* of each handshake protocol. Observing the PN in Fig. 7.9(c) we can notice the same signal sequences of the signal diagram: the sequence of signals for the 4-phase handshake protocol on the left part of the PN (a closed loop between transitions *Req4+*, *Ack4+*, *Req4-* and *Ack4-*) and the sequence for the 2-phase handshake protocol on the right part (a closed loop between transitions *Req2+*, *Ack2+*, *Req2-* and *Ack2-*). The red arrows in Fig. 7.9(b) represent the synchronizations for the conversion from one protocol to the other; in particular, we can notice the connections between the left and right parts of PN in Fig. 7.9(c) by places p_8 and p_9 : the arrows going from *Req4+* of the 4-phase handshake protocol to *Req2+* and *Req2-* of the 2-phase handshake protocol are represented by the connections of the place p_8 in Fig. 7.9(c), whereas p_9 with the connected arcs instead represents the red arrows between *Ack2+* and *Ack2-* of the 2-phase handshake protocol and *Ack4-* of the 4-phase handshake protocol in Fig. 7.9(b). Observe, for instance, the sequence *Ack2+*, *Req2-*. Under normal circumstances, this sequence would be able to progress more quickly. However, due to the required synchronization with the 4-phase handshake, the *Req2-* step is delayed, waiting for *Req4+* since the client communicates with the 4-phase protocol (as indicated by the red arrow in Fig. 7.9(b)). Furthermore, in Fig. 7.9(b) we observe two data transmission cycles. In the 4-phase protocol, both cycles are identical. However, in the 2-phase protocol, the first cycle features both signals *Req2* and *Ack2* rising to a high level, while in the second cycle, they fall to a low level. The dependencies with the 4-phase protocol remain consistent in both scenarios, as indicated by the red arrows: in the first cycle, *Req4+* triggers *Req2+*, and similarly, in the second cycle, *Req4+* leads to the signal change of *Req2-*. What does this imply? Essentially, the red arrows symbolize the protocol transition executed by the controller. Since the client operates on a 4-phase protocol, the *Req4+* signal must be transmitted before it can be converted into *Req2+* (or *Req2-*, depending on the cycle). The same principle applies to the *Ack* signal, which originates from the server as *Ack2+* (or *Ack2-*, varying by cycle) in the 2-phase protocol and is transformed into *Ack4-* in the 4-phase protocol. These dependencies are also evident in the Petri net; for instance, the *Req2-* transition can occur if there are tokens in p_6 and p_8 : p_6 indicates that *Ack2+* was activated (thus the next cycle starts with *Req2-* and not *Req2+*), and p_8 signifies that *Req4+* was activated (representing the initial 4-phase protocol request).

The conversion from PN to MSFSM consists of the following steps [23, 50]:

1. Minimal S-Covering [35, 72–74].
2. S-Component to Non-Interactive FSM mapping.
3. Synchronization Primitive Extraction and Integration into an MSFSM.

Minimal S-Covering

A *bounded* Free Choice Petri net (or Asymmetric-choice Petri net) satisfies the conditions of the Commoner's theorem (Theorem 1), or Commoner's Theorem for Asymmetric-choice systems

(Theorem 2), consequently the net is *well-formed*, i.e *live* and *bounded*. Therefore, by the S-Coverability theorem (Theorem 3), it is possible to obtain a minimal S-cover, decomposing the net into a set of S-components (strongly connected state machines).

An S-component S of a net $N = (P, T, F, M_0)$ can be obtained by a minimal siphon $D \subseteq P$ if it is strongly connected and $\forall t \in S : |\bullet t \cap D| = |t^\bullet \cap D| = 1$.

Algorithm 2: S-components decomposition [75]

Input: An FCPN $N = (P, T, F, M_0)$
Output: A set of S-component $S = \{N' \mid N' = (P', T', F', M'_0)\}$

```

1  $S \leftarrow \emptyset$ 
2 while  $\exists p \in P \wedge p \notin P'$  ; // An uncovered place exists
3 do
4    $P' \leftarrow \{p\}$ 
5    $T' \leftarrow \emptyset$ 
6   while  $\exists p \in P' \mid t \in \bullet p \wedge t \notin T'$  ; // An uncovered transition exists
7   do
8      $H \leftarrow \text{get-Handle}(P \cup T, P' \cup T', F, t)$ 
9      $P' \leftarrow P' \cup (H \cap P)$ 
10     $T' \leftarrow T' \cup (H \cap T)$ 
11     $M'_0 = M_0 \cap P'$ 
12     $N' = (P', T', F', M'_0)$ 
13     $S = S \cup N'$ 

```

Applying Algorithm 2, we find the set of S-components that covers the Petri net. The algorithm calls the *get – Handle* method, which consists in a DFS starting from a place $p \in P'$ and searching backward for transitions and places until it reaches a place $p' \in P'$. The navigation can proceed only through places and transitions that do not belong to P' , and every place can be checked only once because a handle does contain each node exactly once. The algorithm ends if all places in P' have handles covering all of their input transitions and when all places in the net are covered by at least one S-component.

If the original Petri net is Free-Choice, the complexity to achieve the minimal S-cover is $O(PT + P^2)$ [73], whereas for an Asymmetric-Choice Petri Net the aforementioned step is NP-Complete [74].

An example of this S-component decomposition starting from the net in Fig. 7.9 is visible in Fig. 7.10. The set of S-components, built from the minimal siphons of the net ($\{p_0, p_1, p_2, p_3\}$, $\{p_4, p_5, p_6, p_7\}$ and $\{p_0, p_5, p_7, p_8, p_9\}$), witnesses the S-coverability of the initial net since all places and transitions are covered by them. The first S-component covers signals *Req4* and *Ack4*, covering places p_0, p_2, p_3 and p_4 . The second S-component covers signals *Req2* and *Ack2* and places p_4, p_5, p_6 and p_7 . Lastly, the third S-component covers some transition already available in the previous S-components (*Req4+*, *Req2+*, *Ack2+*, *Ack4+*, *Req2-* and *Ack2-*), and the places p_0, p_5, p_7, p_8 and p_9 , so that the place coverage is now complete (places p_8 and p_9 were not covered by the first two S-components).

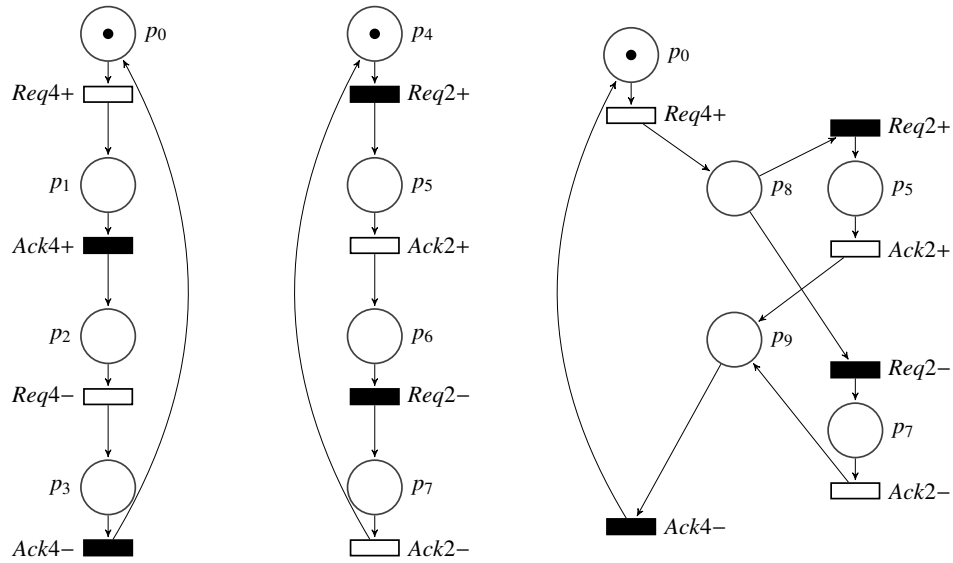


Fig. 7.10: S-component of 4-phases to 2-phases handshake converter.

S-Component to Non-Interactive FSM mapping

The conversion from an S-Component to an FSM is a 1:1 method split in the following steps:

1. Each place p_i is converted to a state s_i : if p_i is part of the initial marking, s_i becomes an initial state.
2. Each transition t is converted to an edge (s_i, s_j) where s_i is the state extracted from $\bullet t$ and s_j from $t\bullet$. When creating the next state and output functions, $\delta: I \times S \rightarrow S$ and $\lambda: I \times S \rightarrow O$ (for a Mealy machine), or $\lambda: S \rightarrow O$ (for a Moore machine), the type of transition (input / output) is kept.

The obtained FSMs preserve the implicit synchronizations inherited from the starting FCPN semantics. The aim of the next step will be to expose such synchronizations through the *Wait State* and *Transition Barrier* primitives.

Non-Interactive FSMs of the current example are shown in Fig. 7.11.

Synchronization Primitive Extraction and Integration

The *Transition Barrier* primitive is extracted by analyzing the flow relations of the S-components: if there is a common place p between different S-components, then $\bullet p$ and $p\bullet$ join the same barrier tb . In case of a common transition between two S-components (which is not in any fork or join in the initial PN), the places connected to the aforementioned transition will belong to at least two S-components and will be in the same transition barrier.

Occasionally, we might encounter situations where there is only a single transition barrier or multiple barriers, lacking common adjacent states. When extracting S-components and transition barriers from a Petri net (PN), each transition barrier of this kind essentially signifies a transition in the PN that involves both a join and a fork. In such scenarios, no common place

exists between different S-components. As a result, it becomes essential to examine the transitions in the original PN to identify and take care of any transition barriers that might have been overlooked. An example can be seen in Fig. 7.12: event b is in the transition barrier tb (Fig. 7.12(c)) and the transition barrier is created from an event in two S-components without common places (Fig. 7.12(b)). Similar cases can be observed with x and \bar{x} in Fig. 7.15(a) and \bar{a} in Fig. 7.19(b).

In Fig. 7.13 the states derived from the common places are (s_0, q_0) , (r_5, q_5) , and (r_7, q_7) . The pre and post transitions of the original places constitute six transition barriers:

- $tb_{Req4+} = (t_0, t_8)$
- $tb_{Ack4-} = (t_3, t_{12})$
- $tb_{Req2+} = (t_4, t_9)$
- $tb_{Ack2-} = (t_7, t_{13})$
- $tb_{Ack2+} = (t_5, t_{11})$
- $tb_{Req2-} = (t_6, t_{10})$

After the synchronization primitive extraction, the δ_i and λ functions of the FSMs need to be transformed to the Δ_i and Λ functions of the MSFSM. So the *Transition Barriers* previously defined can be represented as a set of *Wait State* transitions of different FSMs, which, to activate the next state function, share the same interlocked states $s_k \in i$ -th FSM and $s_j \in n$ -th FSM.

Therefore, in order to represent the synchronization constraint, the next state function of s_k and s_j ($\delta: I \times s_k \rightarrow s_p$ and $\delta: I \times s_j \rightarrow s_m$), respectively contribute to create the next state functions of the i -th and n -th FSM: $\Delta_i: I \times s_j \rightarrow s_p$ and $\Delta_n: I \times s_k \rightarrow s_m$.

In the handshake protocol converter, for instance, the barrier tb_{Req4+} corresponds to $W_1 = (s_0, Req4+, q_0)$ and $W_2 = (q_0, Req4+, s_0)$. So, considering the input $Req4+$, the next state functions of the MSFSM (one for each interacting FSM) are the following ones:

- $\Delta_1: (Req4+) \times q_0 \rightarrow s_1$
- $\Delta_3: (Req4+) \times s_0 \rightarrow q_8$

7.2.2 Original models with the addition of the clock

Introducing the clock signal, the FSMs of the MSFSM model are considered synchronous with the assumption that a state can be activated only at a global clock edge. Also, the Petri nets

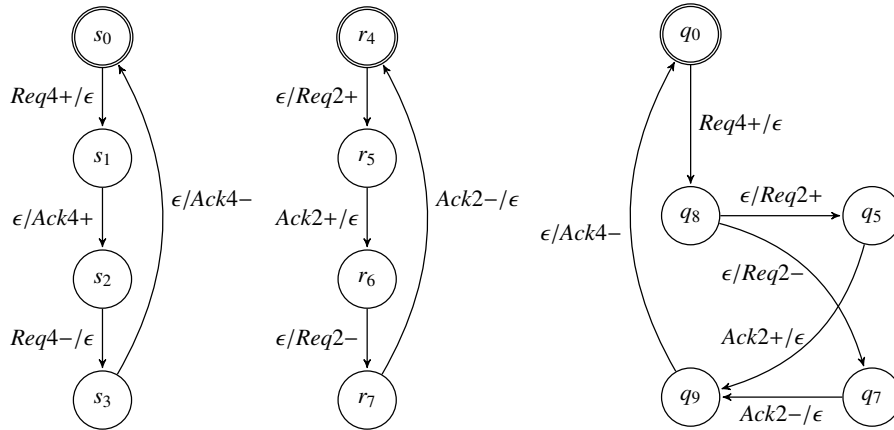


Fig. 7.11: 4-phase to 2-phase handshake converter after the creation of Non-Interactive FSMs.

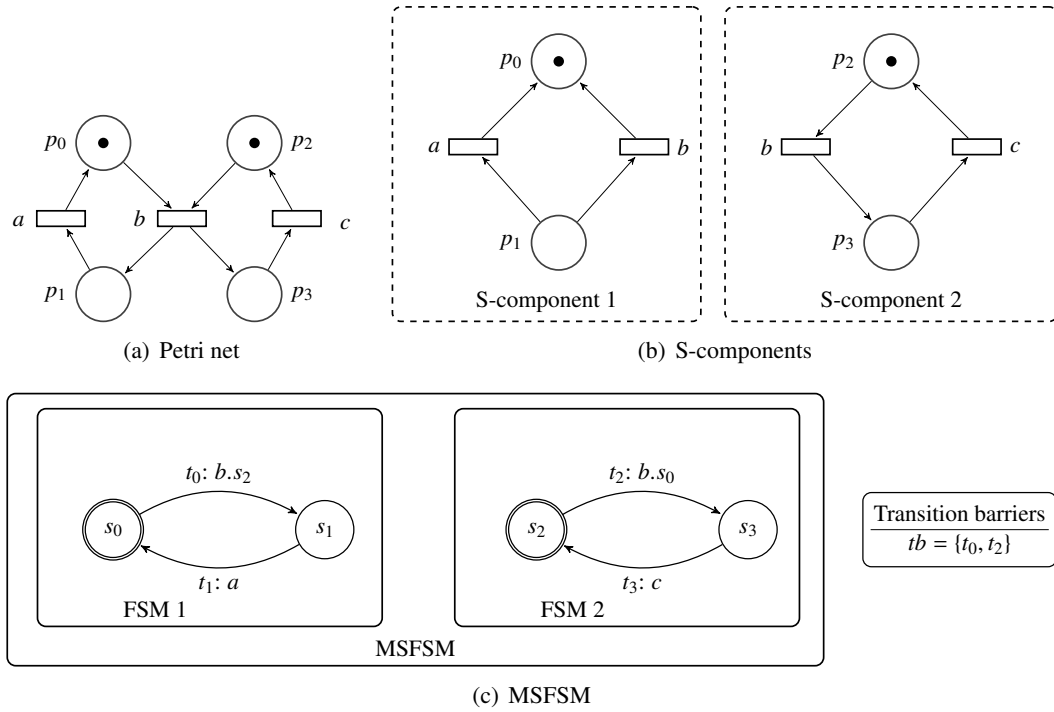


Fig. 7.12: Example showing MSFSM extraction from a Petri net.

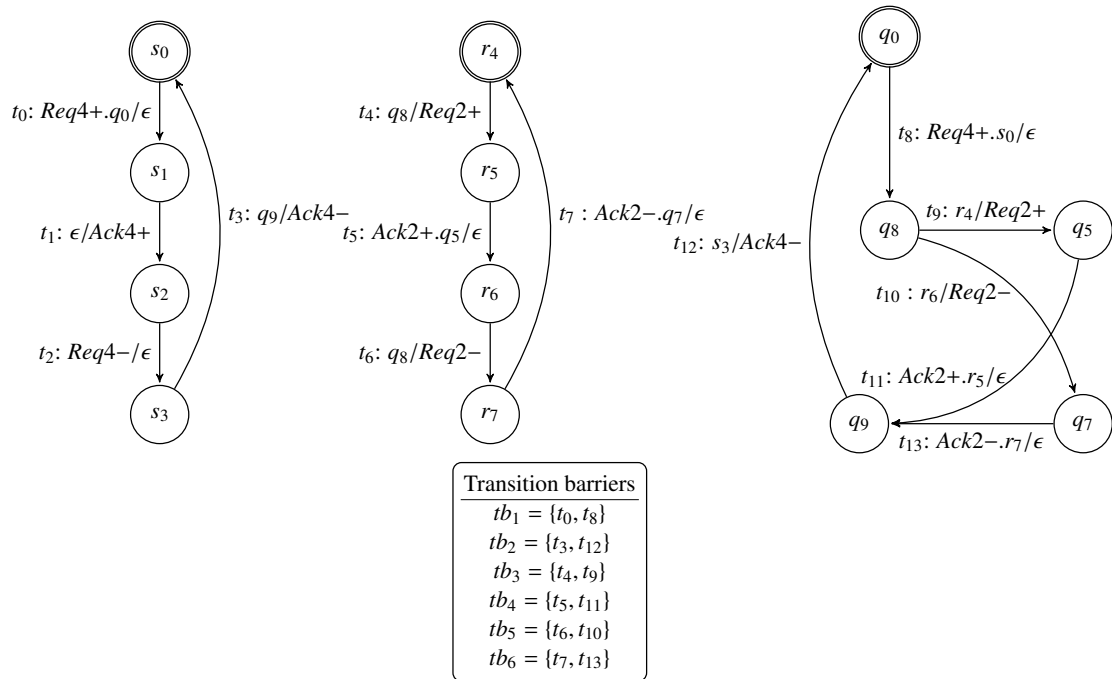


Fig. 7.13: MSFSM.

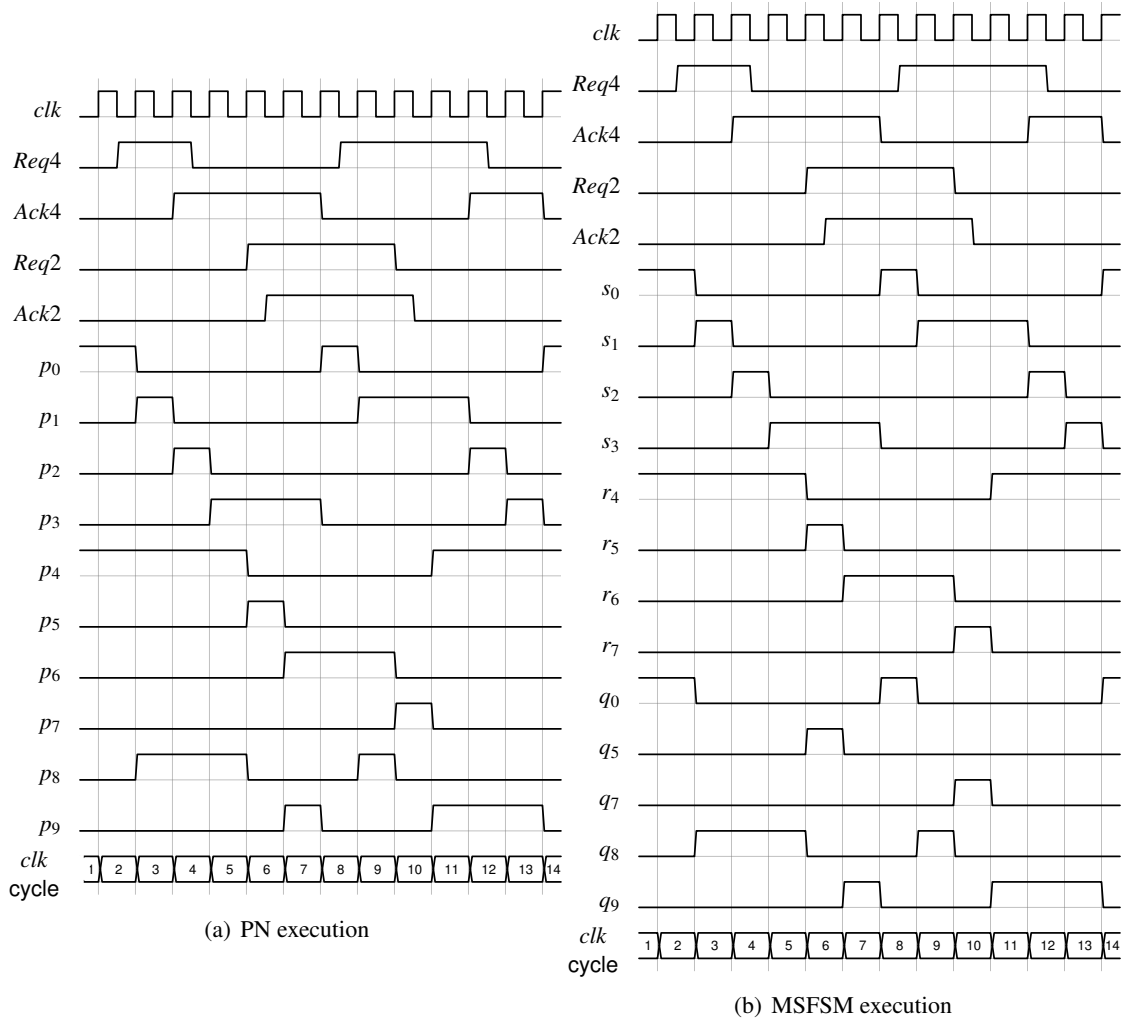


Fig. 7.14: Example of a synchronous execution of the PN representing 4-phase to 2-phase handshake protocol controller (a) and the execution of synchronous MSFSM (b).

follow the same assumption, therefore the PN's places that hold the current state of the system can be activated at a global clock edge, thus there cannot be firing concurrency between the activated transitions.

Fig. 7.14 shows a possible execution of the current 4-phase to 2-phase handshake protocol example seen both as a PN and as a derived FSM, representing its evolution with the signal *clk* from the initial marking until it is reached again. Handshake protocol is usually used in an asynchronous environment, but for example purpose, we can bring the model into a synchronous environment, assuming a clock slow enough to support communication between client and server. Note how MSFSM states evolve in the same way as the places of the initial PN (e.g. p_1 and s_1 , or p_9 and q_9). Additionally, the figure demonstrates that states in different FSMs, which are derived from the same place in the PN (indicated by matching indices), exhibit identical behaviors. For example, this can be seen in states such as s_0 and q_0 , or r_5 and q_5 . As a result, we have

cases like $p_0 = s_0 = q_0$, with the same behavior between a PN's place and different states of the derived FSMs.

Fig.7.15 presents another example that includes a Petri net, its corresponding MSFSM, and a potential evolution of signals. This example illustrates a C-Element, as described in [76], which is a sequential gate featuring m inputs and a single output. The behavior of this gate is such that when all inputs are asserted (or de-asserted), the output follows suit, becoming asserted (or de-asserted) as well.

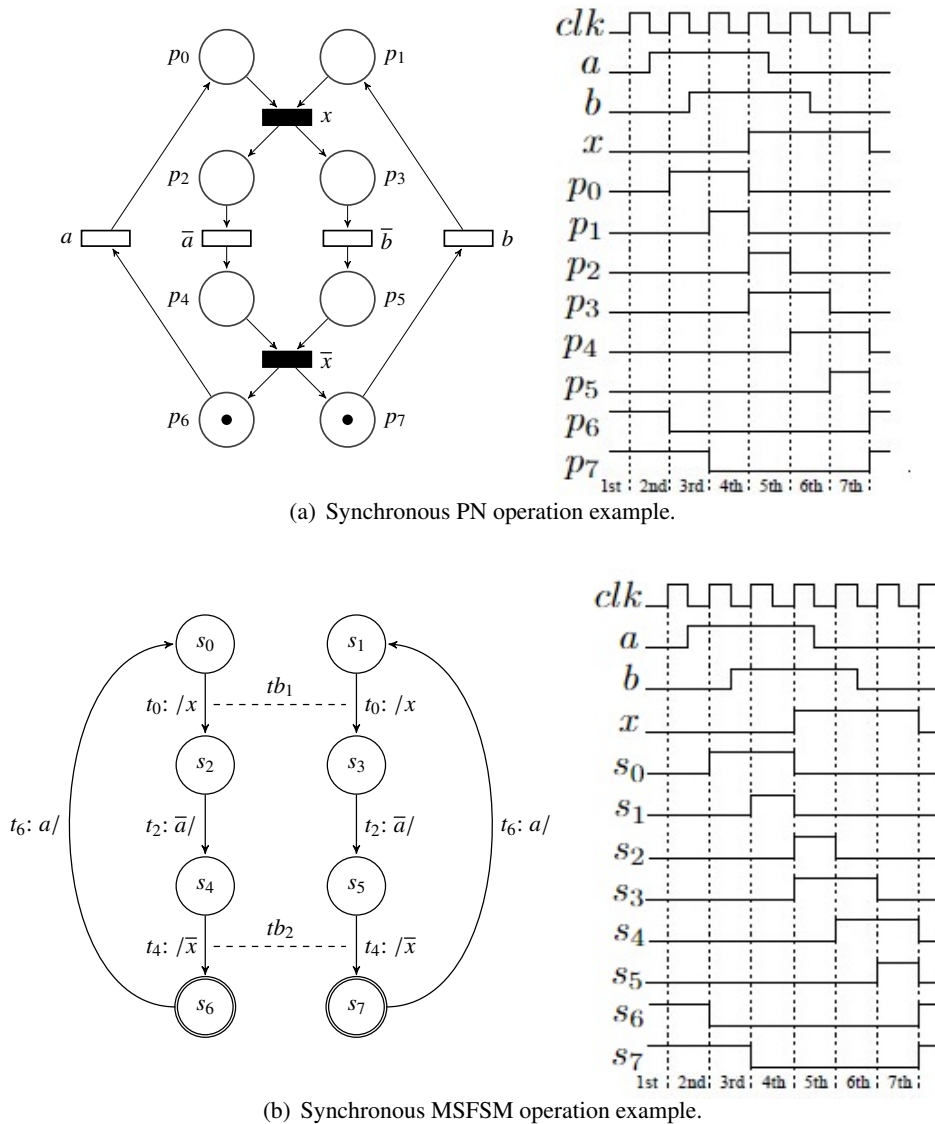


Fig. 7.15: PN to MSFSM transformation representing a synchronous behaviour [23].

Once the MSFSM is created, a synchronous MSFSM architecture can be designed (as it was shown in Fig. 7.1) since a method for the RTL description of MSFSMs was proposed. This part

will not be presented in this thesis; if the reader is interested in the RTL code, it can be found in [23, Section 6.3].

7.3 Synchronous elastic circuits

Since one of the purposes of the MSFSM model is the creation of synchronous circuits, we can consider that the proposed flow can also be used for synchronous elastic circuits [77].

Synchronous elastic circuits are tolerant to variations in computation and communication delays, combining synchronous discipline with the flexibility of elastic systems. Synchronous elastic systems are characterized by:

- **Global clock:** asynchronous elastic systems do not present a clock signal;
- **Elasticity with clock:** since there could be delays caused by unpredictable data arrival times or computation times, these variations can still be handled without the need to pass to asynchronous circuits;
- **Elastic buffers:** if the data arrives slowly the buffer can be stretched and vice versa.

7.4 MSFSM to PN conversion flow

In this section, the conversion flow from MSFSM to PN, originally proposed in [23], will be presented. This section is essential to understand the next one, where we will discuss some problems noticed in the aforementioned flow. The flow is based on the following two steps:

1. MSFSM to S-component mapping.
2. S-component merging to PN conversion.

Fig. 7.16 will be used as a current example of the conversion flow.

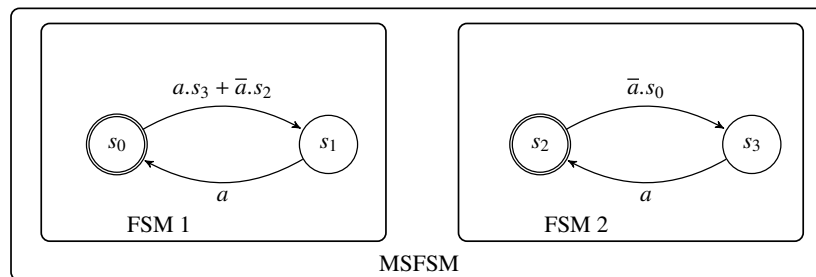


Fig. 7.16: Initial MSFSM.

7.4.1 MSFSM to S-component mapping

The purpose of this step is the conversion of the FSMs of a given MSFSM into a set of S-Components (Definition 22).

This conversion is always possible due to the following theorem.

Theorem 7. [23] *An FSM, represented by a strongly-connected State Machine, is transformable to an S-Component.*

Proof. Each triple (s_1, t, s_2) of FSM's δ is transformed to (s, t) and (t, s') tuples or $(s, t), (t, s_o), (s_o, o)$ and (o, s') tuples, according to whether an element with (s, t) does not exist in λ or it exists. In both cases, there is a path from place s to place s' for each transition connecting them. Thus, the Petri net's connectivity follows the FSM's connectivity and the Petri net is strongly connected.

The Moore type output signals depend only on next state functions, so that they are directly linked with places of the resultant Petri net, without the need to use transitions to be represented in the resultant S-component.

The transformation from a single FSM into an S-component is described in the following steps:

1. Conversion of each state s_i to a place of the S-component p_i : places related to initial states contain a token.
2. Each next state function Δ_i containing only input or output signals is used to create transitions, represented in the form (p_k, t, p_j) . If the Δ_i function connects two states s_k and s_j and contains both input and output signals, a place p_o is created and two transitions are created. The first transition is labeled with the input signal $f(I)$ and connected to p_k and p_o , being represented by (p_k, t_i, p_o) ; the second one takes the output signal $f(O)$ and is represented in the form (p_o, t_o, p_j) .

This method can be used for both Mealy and Moore machines: we can see the variant for Moore machines as a simplification with only input signals.

Fig. 7.17 shows the resultant S-components obtained from the MSFSM in Fig. 7.16.

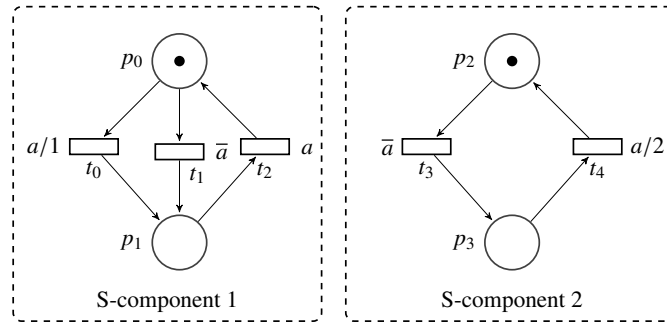


Fig. 7.17: S-components after the MSFSM to S-component mapping.

7.4.2 S-component merging to PN conversion

The last part of the conversion connects S-components in order to create the resultant Petri net. In case of transitions that belong to a *Transition Barrier*, they are merged into one transition.

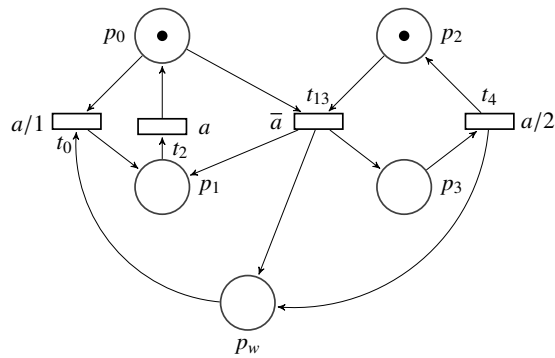
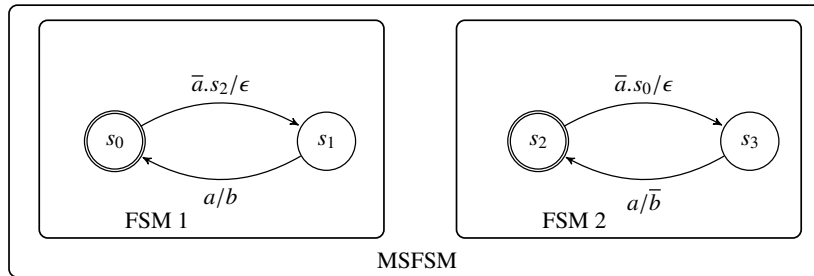


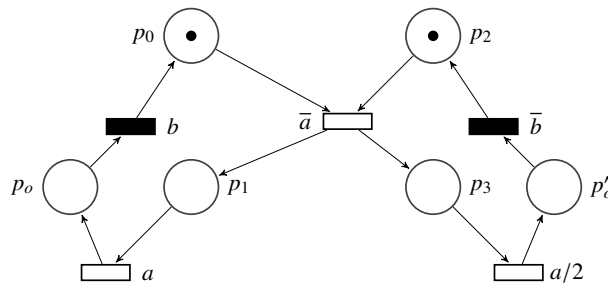
Fig. 7.18: Petri net derived from MSFSM in Fig. 7.16.

In the case of Wait State $W = (s_i, t, s_{i+1}, \dots, s_j)$, the transformation consists in the addition of a new place p_w in the resultant PN, such that $p_w \in \bullet t$ and for each $s_k \in \{s_{i+1}, \dots, s_j\} : p_w \in (\bullet s_k)^\bullet$.

In Fig. 7.18 we can see the representation of the PN derived from the MSFSM in Fig. 7.16 after merging the S-component into a PN. Here we can observe the implementation of the Wait State (thanks to the addition of the place p_w in the final PN), and also the presence of a transition barrier connecting the two S-components (thanks to the transition t_{13} derived from the union of transitions t_1 and t_3 , both labeled by \bar{a}).



(a)



(b)

Fig. 7.19: Example of transformation of an MSFSM with Mealy FSMs (a) into a Petri net (b).

Fig. 7.19 represents an example of transformation starting from an MSFSM containing Mealy FSMs. In this case we see the transformation of the outputs in PN transitions, where filled transitions represent the outputs and empty ones represent the inputs.

7.5 The role of the Wait State synchronization primitive

7.5.1 Issues with the MSFSM to PN conversion flow

The Wait State synchronization primitive represents a critical part of the MSFSM model, raising some issues for the correctness of the conversion. The focus of [23] was on the flow from PN to MSFSM where we never have the creation of a Wait State which is not interlocked with another one: we have always interlocked Wait States designed to become Transition Barriers, leaving the usage of single Wait States rarely exploited.

Given the definition of WS, the awaited state is represented as a signal which is set to “1” when the FSM containing the awaited state is currently in it, “0” otherwise. This is an expected semantics. The problem arises when we perform the MSFSM to PN conversion flow originally presented. Fig. 7.21 represents the PN derived from MSFSM in Fig. 7.20, where we notice that the original flow that introduces a new place p_w in the resultant PN represents different semantics. Specifically, the generated PN represents the Wait State as an unlimited buffer that counts how many times the PN was in p_3 , each time storing a token in p_w and consuming these tokens activating the transition a . Differently from the MSFSM semantics, in the Petri net we are able to produce the sequence “ bda ”. Furthermore, transforming the PN back into an MSFSM (Fig. 7.22), we can see an additional third FSM containing the state s_w derived from p_w and in this case the sequence forbidden in the original MSFSM is allowed: firing the transition barrier containing t_2 and t_6 and thus activating the event b , followed by t_4 and t_0 . Transition t_0 can be activated because after firing the transition barrier with t_6 , the third FSM is in s_w .

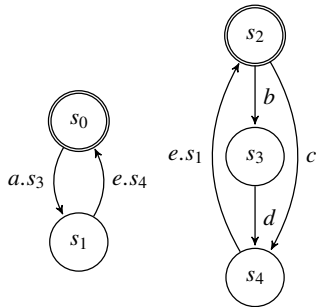


Fig. 7.20: MSFSM example.

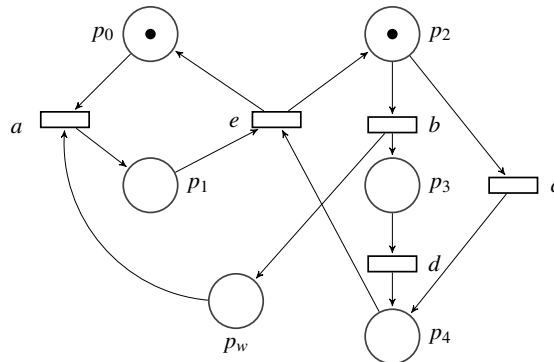


Fig. 7.21: Petri net derived from MSFSM in Fig. 7.20 with a Wait State represented by the place p_w .

7.5.2 Revised Wait State in the MSFSM to PN conversion flow

Since the current transformation of the Wait State synchronization primitive seems to be wrong, I present a conjecture with an alternative transformation for the aforementioned structure, in particular, each Wait State will be transformed into a PN self-loop.

Starting from the originally proposed flow, I propose to perform these three steps:

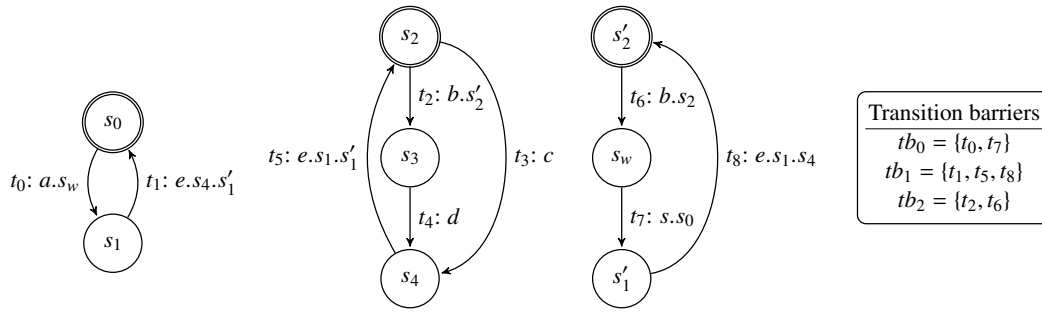


Fig. 7.22: MSFSM derived from PN in Fig. 7.21.

1. MSFSM to S-component mapping.
2. Enrichment of the S-components.
3. S-component merging to PN conversion.

The first step is the same as in Section 7.4.1. The second step represents a new integration of Wait States, and the last step is still the same as in the original flow, but without considering Wait States since they were already transformed in the second step.

Enrichment of the S-components

Until now, each FSM was transformed into an S-component but the synchronization primitives were left unused. In this section we will integrate the Wait States. Given a Wait State $W = (s_i, t, s_j)$ we have a transition t waiting for another FSM to be currently in the state s_j . Creating the Petri net, this constraint can be represented as a self-loop on the place p_j derived from the state s_j with the transition t in the self-loop. This self loop should be added to the S-component containing the place p_j . Furthermore we are sure that the addition of a self-loop to an S-component will not damage the S-component structure, since all places will remain strongly connected, keeping also $|\bullet t| = |t\bullet| = 1$ for each transition of the S-component.

In this section we will still use the running example of Section 7.4 (Fig. 7.16): in particular, before performing this new step, we have the two S-components in Fig. 7.17. In Fig. 7.23 we can see the evolution of the current example with the introduction of the self-loop in the second S-component representing the Wait State $W = (s_0, a, s_3)$, i.e. the transition a , starting from s_0 and waiting for s_3 .

S-component merging to PN conversion

This part of the conversion connects S-components in order to create the resultant Petri net. In case of transitions that belong to a Transition Barrier, they are merged into a single transition with a common label. Since the Wait States were added in the previous step, we do not need to add them anymore, avoiding the procedure presented in the original work.

Fig. 7.24 shows the final result: the transitions t_1 and t_3 , being in the same Transition Barrier, are merged in a single fork-join transition common to the two S-components. This transition cannot fire if both S-components do not have a token, respectively, in states p_0 and p_2 , i.e., it

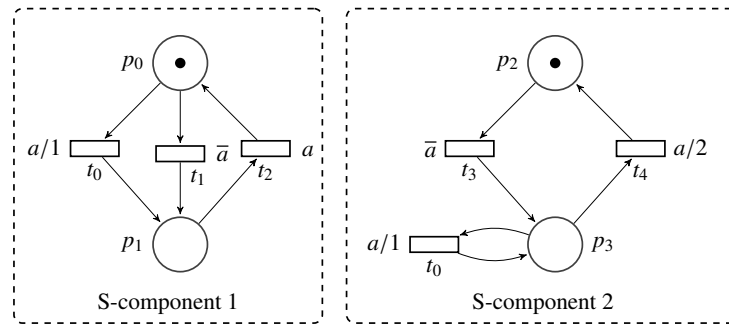


Fig. 7.23: S-components after the integration of the self loops.

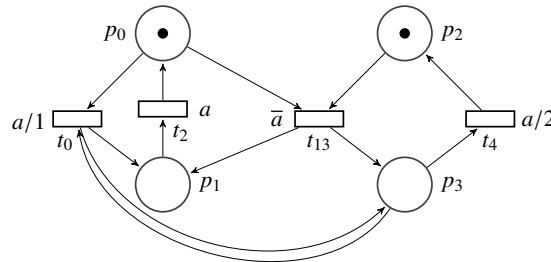


Fig. 7.24: Petri net after the complete conversion flow, starting from Fig. 7.16.

may fire only when both signals representing the states s_0 and s_2 in the MSFSM have value “1”. It is important to observe that the transition represents simultaneously a join and a fork, since it is a transition that synchronizes two concurrent paths. Moreover, in the final Petri net there are two self-loop flows (t_0, p_3) and (p_3, t_0) , representing a *Wait State*: t_0 cannot fire if there is no token in p_3 , i.e., t_0 cannot fire if the corresponding FSM is not in the state s_3 .

7.5.3 Self-loops in the PN to MSFSM conversion flow

Considering the originally proposed PN to MSFSM transformation flow, the initial PNs never had self-loops. The reason probably is the concentration of the research on Free-Choice Petri nets, overshadowing Asymmetric-choice Petri nets, a PN subclass which may present self-loops and which was considered as a valid starting point for MSFSM generation. It is possible to have also an FCPN with a self-loop, but the self-loop is trivial, modeling the possibility of firing an event an infinite number of times (Fig. 7.25). Usually, self-loops represent a constraint, involving a place of one S-component and a transition in two different S-components (transition a in Fig. 7.26, better seen after the S-component extraction). The fact that the transition is in two different S-components implies the presence of an Asymmetric-choice structure, denying the possibility to have an FCPN. In Fig. 7.26 the Asymmetric-choice structure is created by places p_0 and p_3 , and events a and b , where the event a is in the post-set of both places.

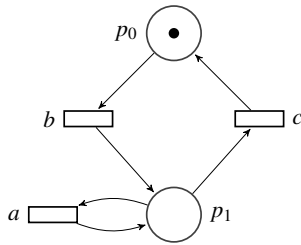


Fig. 7.25: Example of self-loop with the transition belonging to a single S-component where event a can be fired an infinite number of times before firing event c .

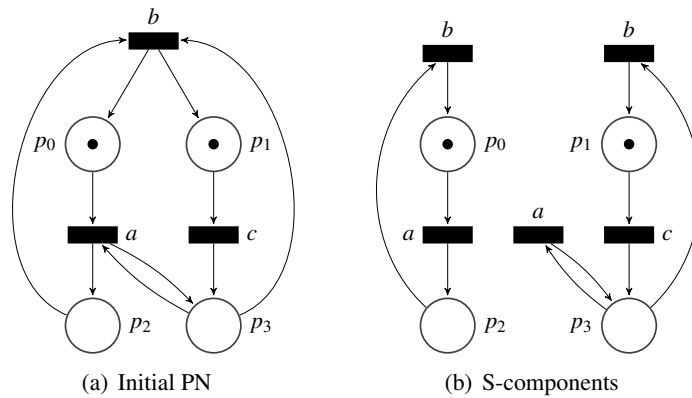


Fig. 7.26: Example of Petri net with self-loop.

7.6 A simple use case

Consider a warehouse with dozens of autonomous robots, where each robot moves dangerous packages. Robots are autonomous, knowing from where the package should be taken and where it must be delivered in the warehouse. Each robot has sensors, but for security reasons an additional mechanism is going to be implemented, especially because the robots move very fast. Thanks to sensors placed across the warehouse it is possible to give additional information to the robots, in particular, when a robot is moving to a crossroad, a signal is sent to the control system, which checks if other robots are going to cross the same point, and if so a stop signal is sent for all robots in one of the two perpendicular directions.

Fig. 7.27 represents the Petri net modeling the controller which activates the stop lights of a crossing. Say that the two perpendicular ways are called “A” and “B”, initially empty (places “A_EMPTY” and “B_EMPTY”). The places of the Petri net “A_GREEN” and “B_GREEN” represent the possibility of crossing the intersection in both ways. Once the token in one of the two places is removed, we suppose that in the related direction the intersection cannot be crossed. In case of incoming robots in both directions, in the first case the robot in direction B will intersect the crossroad. Next, the priority will alternate between the two roads. Let us see two possible executions analyzing the behavior when a robot arrives on the way A. First, the sensors capture the arrival of the robot activating the event “A_INCOMING_VEHICLE”. If way B is empty (token in “B_EMPTY”), the internal signal “FREE_ROAD_A” can be activated, to represent the robot intersecting the crossroad and returning back to the initial situation by means of a token in the place “A_EMPTY”. If there is a robot on the way B when a robot on the way B is arriving, the transition “A_RED” or “B_RED” will be activated, depending on the precedence, by removing the token from one of the places representing the “green light” (“A_GREEN” and “B_GREEN”). Only once all the vehicles passed, the green light on the other direction is activated. To manage the effective connections to the lights (in case of way A), it is enough to use the signals “A_RED” and “B_PASSED”: initially the green light is active and we have a token in “A_GREEN”, in case of the deactivation of the light, the event “A_RED” is activated, and once “B_PASSED” has been activated the green light turns back on.

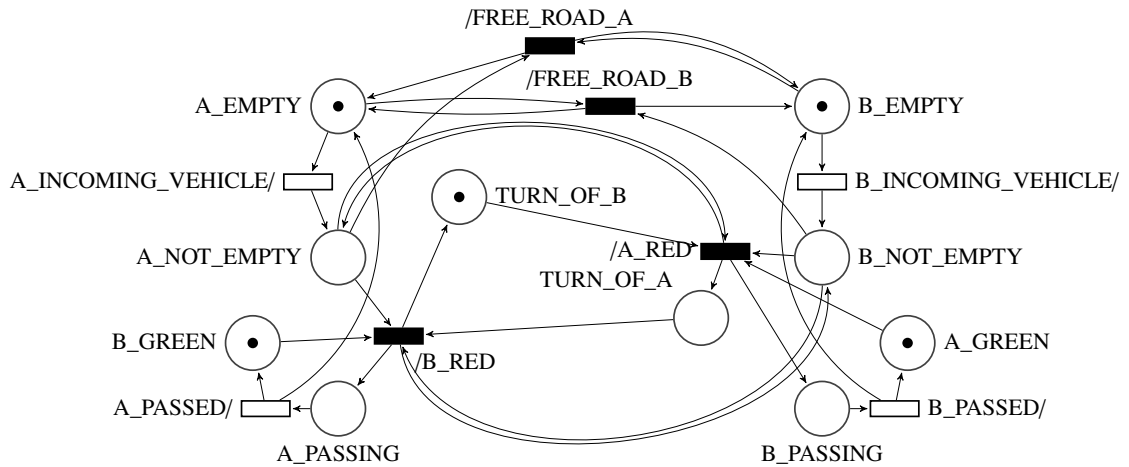


Fig. 7.27: Petri net representation of the controller managing crossroads warehouse.

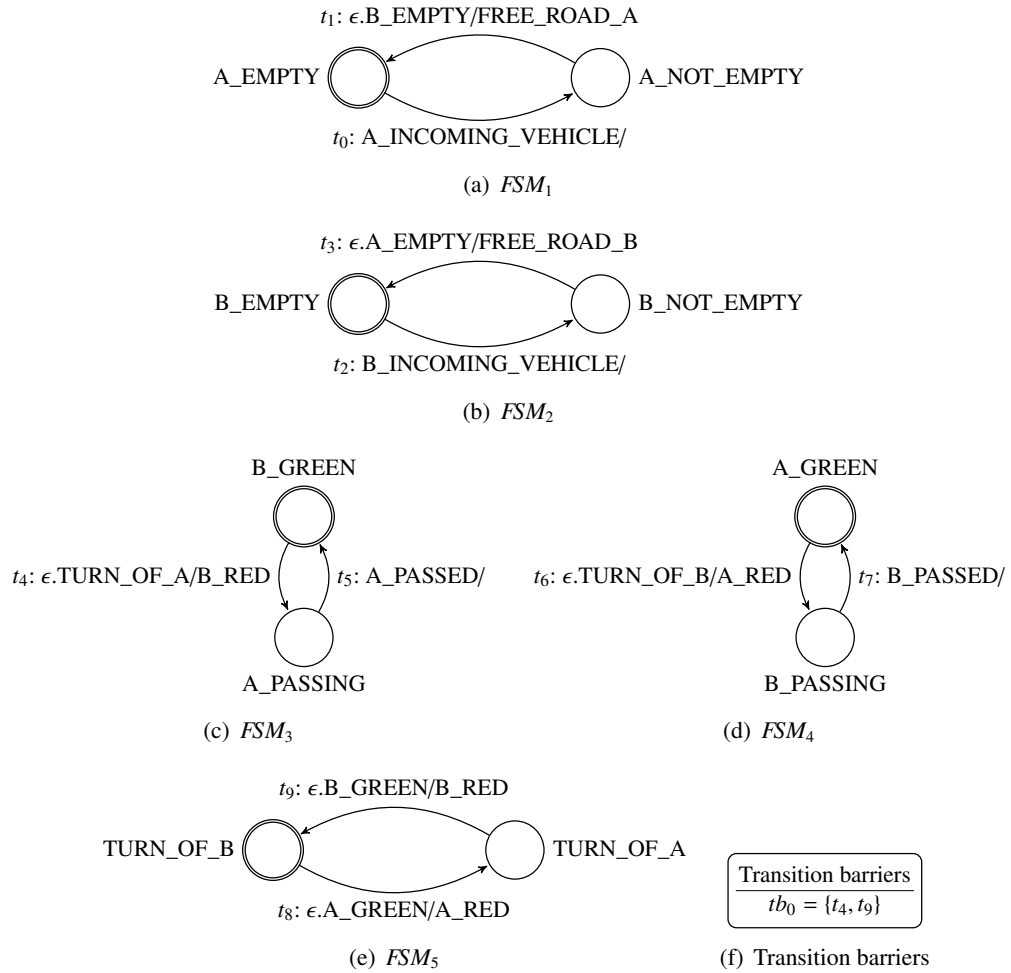


Fig. 7.28: MSFSM derived from PN in Fig. 7.27.

From the Petri net in Fig. 7.27, three S-components can be extracted generating the MSFSM in Fig. 7.28. Given the resultant MSFSM, we can immediately notice the difficulty in following the synchronizations between the FSMs, even if the structures of the single FSMs are very simple, preferring the reachability graph generated from the Petri net or the one generated by the synchronization of the FSMs of the MSFSM. Indeed, even if the MSFSM model could be used as an intermediate model for the analysis of the original PN subparts, its utility is apparent once a synchronous circuit is going to be designed.

Notice the presence of Wait States; it means that the current example does not represent a Free-Choice Petri net, in this case the PN is even outside the Asymmetric-choice subclass (we can see for example the following confusion structure $(B_EMPTY)^* = \{B_INCOMING_VEHICLE, FREE_ROAD_A\}$ and $(A_NOT_EMPTY)^* = \{FREE_ROAD_A, B_RED\}$), violating the restriction required for the validity of the MSFSM decomposition proposed in [23]. Instead, this example supports our conjecture based on the revised Wait State extraction, even though a formal proof of correctness in the general case is still missing.

In this chapter, the MSFSM model was presented, compared to Communicating Sequential Processes (CSP) and Synchronous Languages, in particular SynchChart, highlighting the reasons which led us to explore this model. A flow for the design of synchronous circuits starting from Petri nets and based on MSFSMs was presented. Special cases of possible resultant synchronous circuits were proposed, in particular synchronous elastic circuits. The opposite flow from MSFSMs to PNs was shown, explaining an inconsistency regarding the Wait State synchronization primitive in the original transformation flow, suggesting a revised version of the Synchronization Primitive and a modified transformation flow from MSFSMs to PNs. An explanation of self-loops in PNs was given, showing also a possible use case for MSFSMs.

Part IV

Conclusion

Final considerations

8.1 Main contributions of this thesis

The first part of the thesis explores the decomposition of Transition Systems and Petri Nets based on the theory of regions. In this part we introduced the notion of a set of excitation-closed Petri nets derived from a Transition System and we proved the existence of a bisimulation between the initial TS and the synchronous product of the reachability graphs of the derived PNs. We defined and implemented a flow to create a decomposition into a set of synchronizing SMs, and then we reported the experimental results. We defined and implemented a similar flow also for FCPNs, and then we proposed a new method based on encoding the excitation-closure property by means of BDDs, which allows the search of k FCPNs simultaneously instead of using a sequential approach. We have also shown that passing from SMs to FCPNs there is a huge difference in the complexity of the implementation, since FCPN minimization can transform a Free-Choice Petri net into Asymmetric-Choice one; furthermore, it is much more complex to guarantee that an FCPN decomposition contains only safe FCPNs with respect to the SM decomposition, for which limiting the initial marking guarantees safeness.

In the second part of this thesis, we focus on the discovery of the MSFSM model as a bridge for the synthesis of synchronous circuits. A comparison with similar models was made, including Communicating Sequential Processes and Synchronous Languages. After having presented the flows from Petri net to MSFSM and vice versa it was shown an inconsistency with the Wait State synchronization primitive in the original conversion flow, showing how self-loops in Petri nets could become Wait States passing to the MSFSM model. It was observed that this kind of mapping probably was not noticed by the original authors, since self-loops usually do not belong to either FCPN nor ACPNs. Due to lack of time, this last part was presented as a conjecture and was not formally proved for general Petri nets.

8.2 Considerations about the decomposition based on regions theory

In this thesis, we described a new method for the decomposition of transition systems into a synchronous composition of Petri nets, with a special focus on the decomposition into sets of State Machines and Free-choice Petri nets.

The experimental results demonstrate that the decomposition algorithm can be run on transition systems with up to one million states, therefore it is suitable to handle real cases.

The resultant set of synchronizing FCPNs offers a good complexity trade-off compared to other more expressive classes of PNs [28] vs. decomposition into completely sequential SM.

Since the generation of minimal regions is currently a computational bottleneck, future work could overcome this limitation, due to improvements in the efficiency of last-generation MIS and SAT solvers and HPC¹.

The power of High Performance Computing can be very useful since the generation of minimal regions is highly parallelizable. HPC can also be exploited in other steps of the decomposition algorithm, e.g., different MIS computations could be performed simultaneously applying constraints to each parallel computation (e.g., assigning a state to each thread and forcing it to be in the MIS result). Also, the encoding of excitation-closure for the search of k PNs simultaneously is highly parallelizable since a set of constraints is extracted for each event independently.

The possibility of extracting hidden knowledge from unstructured data suggests that the presented decomposition flow based on regions may be applied to process mining, especially in Business Process Management (BPM) field, deriving FCPNs or SMs for different parts of the mined behavior and helping to represent the entire system as a set of connected concurrent subparts. A TS can be created directly from a set of traces [15, 16] and then given as input to the presented decomposition algorithm. As mentioned previously, there are advantages in decomposing into a set of FCPNs, rather than representing the entire behavior with only one PN, especially when the PN is very complex. The decomposition into simpler FCPN/SM structures could provide benefits, especially if the aim is to identify parts of the total behavior.

Another direction for the improvement of the presented decomposition algorithms could be an extension to k -bounded PNs. In [38] the possibility to derive a bounded PN starting from a transition system, called k -ECTS, was proved. This result is a promising indicator for the existence of a decomposition flow for the achievement of multiple k -bounded PNs. Furthermore, the example in the introduction section shows how k -bound PNs are important in the representation of real cases, significantly expanding the usability of the proposed work.

8.3 Considerations about MSFSM model

We have seen that the MSFSM model can be used for the synthesis of synchronous circuits without time and space explosion. I tried to exploit the usage of the MSFSM model also in the opposite direction, starting from a set of synchronizing FSMs and obtaining an MSFSM, used for property verification. However, this approach, particularly in the field of verification, proved to be suboptimal for the MSFSM model. This is especially true during manual verification, where there is a natural inclination to rely on a canonical model, such as FSMs or Petri nets, for comparative analysis. Similarly, when the verification process is automated, the MSFSM model does not offer any distinct advantages.

¹ High Performance Computing: aggregation of computing power to solve problems too complex to be solved by a normal desktop computer or workstation.

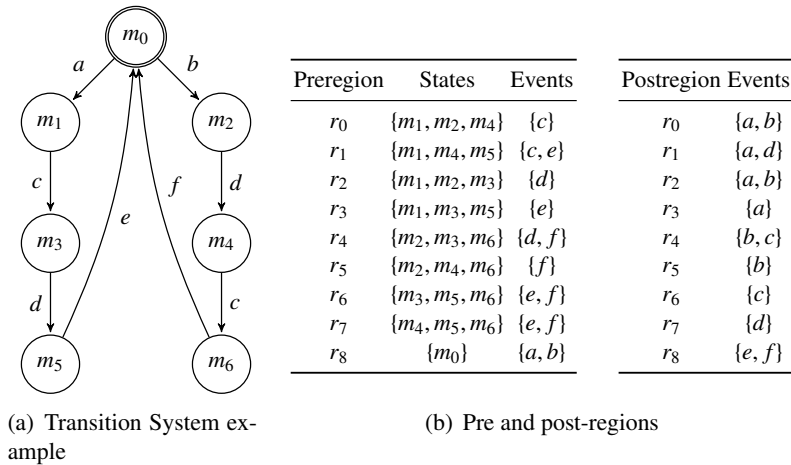


Fig. 8.1: Monolithic Transition System and its pre and post regions

As future work, I would like to mention the proof of the presented conjecture: the equivalence between MSFSM and PN models considering the newly introduced Wait State synchronization primitive. This proof would represent an extension of the proof presented in [23], limited to FCPNs.

8.4 PN decomposition: MSFSM vs. Regions Theory

Starting from a Petri net, the MSFSM decomposition can be performed directly; when using the theory of regions instead, the reachability graph extraction should be performed and from the derived transition system the minimal regions would be obtained. What is the main difference between the two flows? We can view MSFSM decomposition as a selection of structures already embedded in the PN (S-components) with an addition of synchronizations, which also are already present in the PN but represented in a different way. The decomposition based on regions instead looks into the behaviour of the Petri net searching to extract information not yet explicit, and this step is done extracting the regions from the transition system: each region can be associated to a place of a possible PN, but calculating all minimal regions we may find regions associated to places that did not appear in the original PN. Regions allow a deeper exploration than MSFSMs, relying on the fact that the satisfaction of the excitation-closure property yields PN behaviours. As a result we could have better minimized results with regions than with MSFSMs. An example can be seen comparing Figs. 8.3 and 8.4. Observing the two decompositions, we notice that the first two components are isomorphic, and that the MSFSM decomposition was unable to recognize that the fourth component (FSM_4) is completely redundant (see Figs. 8.5 and 8.6).

Looking at the results of the final model, the theory of regions has the advantage of using synchronization based on parallel composition, as we have seen in Sec. 7.1. MSFSMs instead are synchronized with their own synchronization primitives, which are not very intuitive, but despite the new synchronization method, this model can be used for the synthesis of synchronous

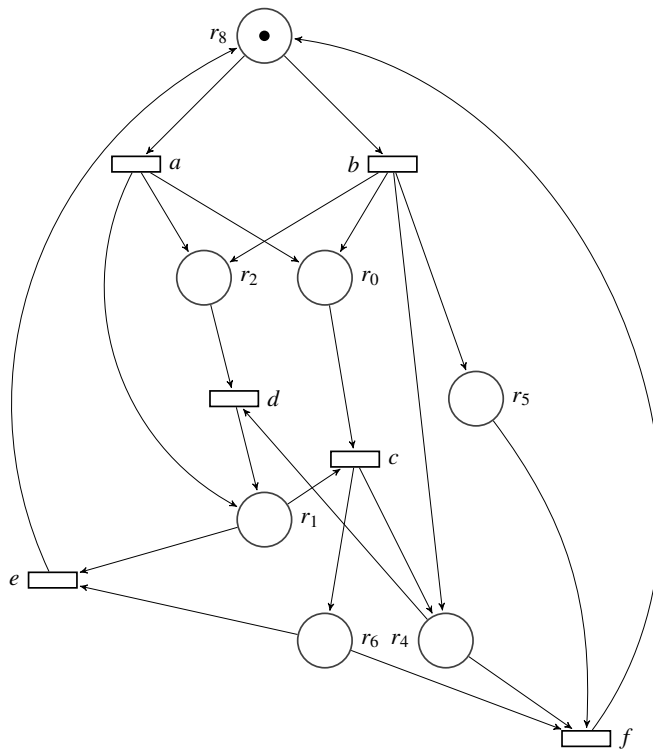


Fig. 8.2: PN derived from TS in Fig. 8.1.

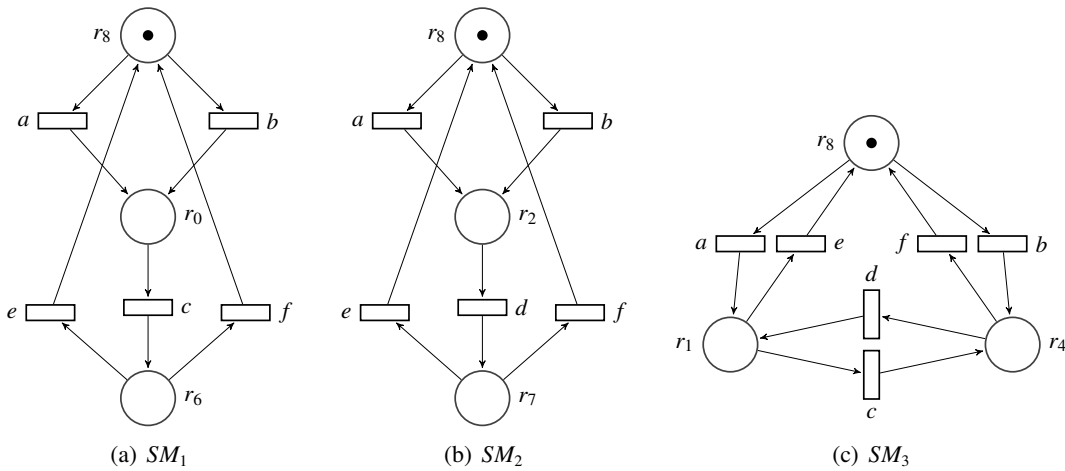


Fig. 8.3: SM decomposition of PN in Fig. 8.2.

circuits (Sec. 7.2). Synchronizing PN, instead, are more suitable for system modeling and protocol design. Furthermore, the creation of structures from event logs may be possible, as we mentioned in Sec. 8.2.

Another big difference between the two approaches is represented by the MSFSM limitations: the decomposition can be done starting only from a Free-Choice or Asymmetric-Choice

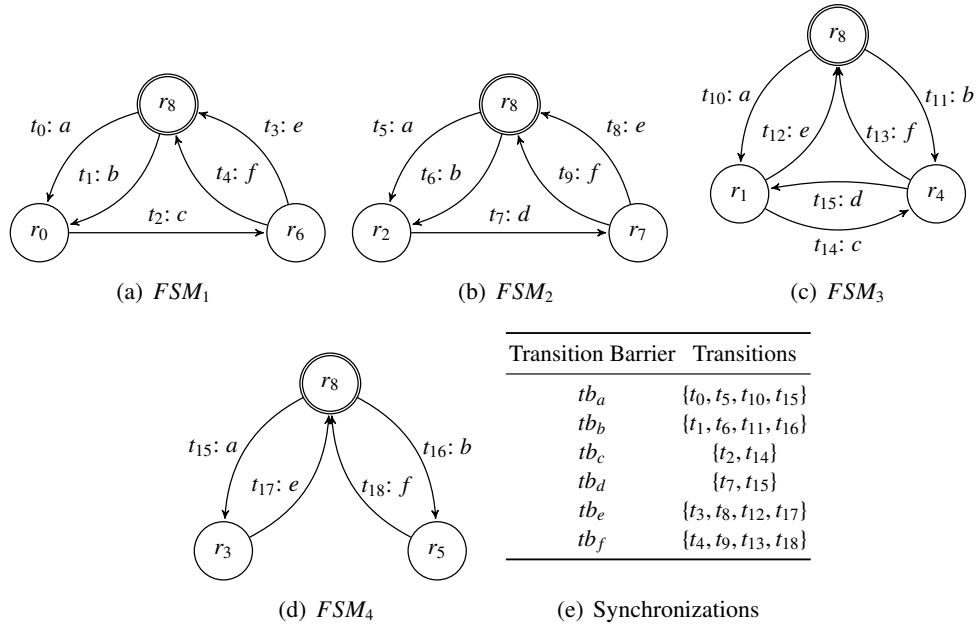


Fig. 8.4: MSFSM derived from PN in Fig. 8.2.

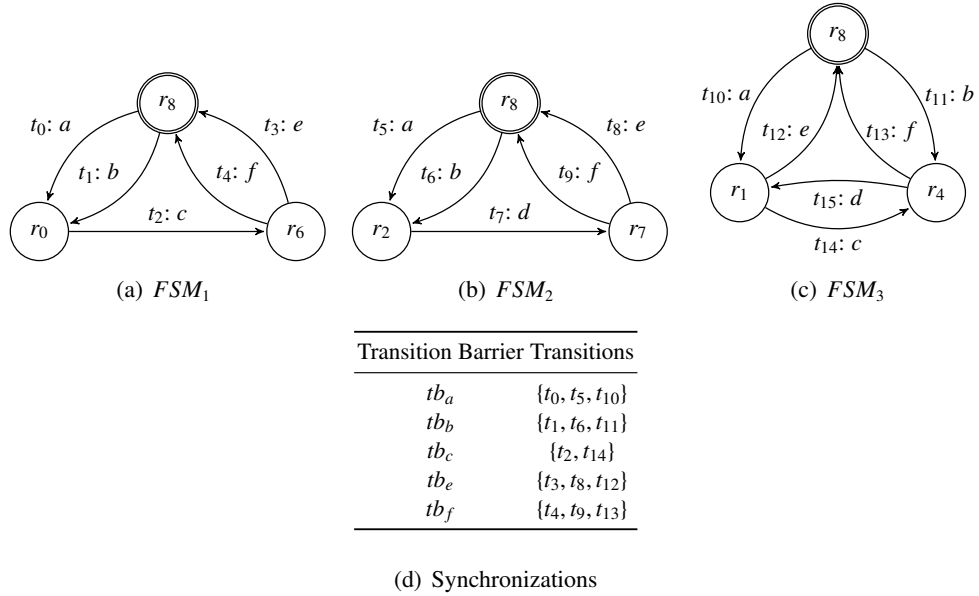
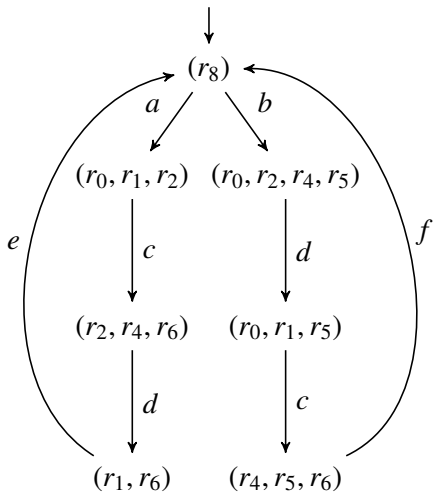
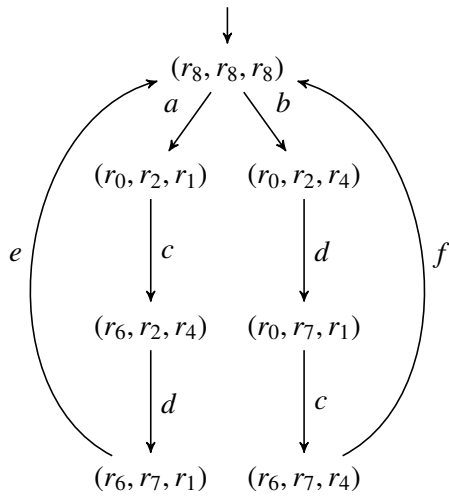


Fig. 8.5: MSFSM derived from PN in Fig. 8.2 without FSM_4 of Fig. 8.4.

Petri net with STG labels, meanwhile for the decomposition based on the theory of regions, it is sufficient to have a safe Petri net.



(a) Reachability graph of PN in Fig. 8.2



(b) Reachability graph of MSFSM in Fig. 8.5

Fig. 8.6: Reachability graphs of PN in Fig. 8.2 and MSFSM in Fig. 8.5

References

1. S. C. White and S. S. Sarvestani, "Comparison of Security Models: Attack Graphs Versus Petri Nets," in *Advances in Computers*. Elsevier, 2014, vol. 94, pp. 1–24.
2. Gartner. Gartner Glossary - Business Process Management (BPM). [Online]. Available: <https://www.gartner.com/en/information-technology/glossary/business-process-management-bpm>
3. W. M. P. van der Aalst, "The application of Petri nets to workflow management," *Journal of circuits, systems, and computers*, vol. 8, no. 01, pp. 21–66, 1998.
4. —, "Making Work Flow: On the Application of Petri Nets to Business Process Management," in *Application and Theory of Petri Nets 2002*, J. Esparza and C. Lakos, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 1–22.
5. —, "Challenges in business process management: Verification of business processes using Petri nets," *Bulletin of the EATCS*, vol. 80, no. 32, pp. 174–199, 2003.
6. —, "Business process management demystified: A tutorial on models, systems and standards for workflow management," *Lectures on concurrency and petri nets*, 2004.
7. —, "Using free-choice nets for process mining and business process management," in *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*. IEEE, 2021, pp. 9–15.
8. —, "Business process management as the "Killer App" for Petri nets," *Software & Systems Modeling*, vol. 14, no. 2, pp. 685–691, 2015.
9. —, "Decomposing process mining problems using passages," in *International Conference on Application and Theory of Petri Nets and Concurrency*. Springer, 2012, pp. 72–91.
10. —, "Decomposing Petri nets for process mining: A generic approach," *Distributed and Parallel Databases*, vol. 31, no. 4, pp. 471–507, 2013.
11. H. Verbeek and W. M. P. van der Aalst, "Decomposed process mining: The ILP case," in *International Conference on Business Process Management*. Springer, 2014, pp. 264–276.
12. J. de San Pedro and J. Cortadella, "Mining structured Petri nets for the visualization of process behavior," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 839–846.
13. D. Taibi and K. Systä, "From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining," in *CLOSER*, 2019, pp. 153–164.
14. W. M. P. van der Aalst, *Discovering Directly-Follows Complete Petri Nets from Event Data*. Cham: Springer Nature Switzerland, 2022, pp. 539–558. [Online]. Available: https://doi.org/10.1007/978-3-031-15629-8_29
15. W. M. P. van der Aalst, V. Rubin, H. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting," *Software & Systems Modeling*, vol. 9, no. 1, p. 87, 2010.
16. J. Carmona, J. Cortadella, and M. Kishinevsky, "Divide-and-conquer strategies for process mining," in *International Conference on Business Process Management*. Berlin, Heidelberg: Springer, 2009, pp. 327–343.
17. V. Teren, J. Cortadella, T. Villa *et al.*, "Seto: a framework for the decomposition of Petri nets and transition systems," in *Proceedings of 26th Euromicro Conference on Digital System Design (DSD)*, 2023, pp. 669–677.
18. A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy 2008)*, Pasadena, CA, 2008, pp. 11–15.

19. T. Philipp and P. Steinke, “PBLib – A Library for Encoding Pseudo-Boolean Constraints into CNF,” in *Theory and Applications of Satisfiability Testing – SAT 2015*, ser. Lecture Notes in Computer Science, M. Heule and S. Weaver, Eds. Springer International Publishing, 2015, vol. 9340, pp. 9–16.
20. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, “Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers,” *IEICE Transactions on Information and Systems*, vol. 80, no. 3, pp. 315–325, 1997.
21. O. Bunte, J. F. Groote, J. J. A. Keiren, M. Laveaux, T. Neele, E. P. de Vink, W. Wesselink, A. Wijs, and T. A. C. Willemse, “The mCRL2 Toolset for Analysing Concurrent Systems,” in *Tools and Algorithms for the Construction and Analysis of Systems*, T. Vojnar and L. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 21–39.
22. J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, “Graphviz—open source graph drawing tools,” in *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9*. Springer, 2002, pp. 483–484.
23. P. M. Mattheakis, “Logic Synthesis of Concurrent Controller Specifications,” Ph.D. dissertation, University of Thessaly, 2013.
24. P. M. Mattheakis and C. P. Sotiriou, “Polynomial Complexity Asynchronous Control Circuit Synthesis of Concurrent Specifications Based on Burst-Mode FSM Decomposition,” in *2013 26th International Conference on VLSI Design and 2013 12th International Conference on Embedded Systems*, Jan 2013, pp. 251–256.
25. P. M. Mattheakis, C. P. Sotiriou, and P. A. Beerel, “A polynomial time flow for implementing free-choice Petri nets,” in *2012 IEEE 30th International Conference on Computer Design (ICCD)*, Sep. 2012, pp. 227–234.
26. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, “Deriving Petri nets from finite transition systems,” *IEEE Transactions on Computers*, vol. 47, no. 8, pp. 859–882, Aug 1998.
27. A. Mazurkiewicz, “Compositional semantics of pure place/transition systems,” *Fundamenta Informaticae*, vol. 11, no. 4, pp. 331–355, 1988.
28. J. Esparza, “Decidability and complexity of Petri net problems—an introduction,” in *Advanced Course on Petri Nets*. Springer, 1996, pp. 374–428.
29. T. Murata, “Petri nets: Properties, analysis and applications,” *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.
30. E. Badouel, L. Bernardinello, and P. Darondeau, *Petri net synthesis*. Berlin: Springer, 2015.
31. J. L. Peterson, “Petri Nets,” *ACM Comput. Surv.*, vol. 9, no. 3, pp. 223–252, Sep. 1977. [Online]. Available: <http://doi.acm.org/10.1145/356698.356702>
32. L. Bernardinello, “Synthesis of net systems,” in *Application and Theory of Petri Nets 1993*, M. Ajmone Marsan, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 89–105.
33. M. Hack, “Extended State-Machine Allocatable Nets, an Extension of Free Choice Petri Nets Results,” Cambridge, Massachusetts, 1974.
34. K. Jensen, “Coloured Petri nets and the invariant-method,” *Theoretical computer science*, vol. 14, no. 3, pp. 317–336, 1981.
35. P. Kemper, “O(PT) - Algorithm to Compute a Cover of S-components in EFC-nets,” 1994.
36. A. A. Kalenkova, I. A. Lomazova, and W. M. P. van der Aalst, “Process model discovery: A method based on transition system decomposition,” in *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, 2014, pp. 71–90.
37. A. Mokhov, J. Cortadella, and A. de Gennaro, “Process windows,” in *2017 17th International Conference on Application of Concurrency to System Design (ACSD)*. IEEE, 2017, pp. 86–95.
38. J. Carmona, J. Cortadella, and M. Kishinevsky, “New region-based algorithms for deriving bounded Petri nets,” *IEEE Transactions on Computers*, vol. 59, no. 3, pp. 371–384, 2009.
39. E. Best, T. Hujsa, and H. Wimmel, “Sufficient conditions for the marked graph realisability of labelled transition systems,” *Theoretical Computer Science*, vol. 750, pp. 101–116, 2018.
40. V. Teren, J. Cortadella, and T. Villa, “Decomposition of transition systems into sets of synchronizing state machines,” in *2021 24th Euromicro Conference on Digital System Design (DSD)*. IEEE, 2021, pp. 77–81.
41. ———, “Decomposition of transition systems into sets of synchronizing Free-choice Petri Nets,” in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 165–173.
42. F. Somenzi, “CUDD: CU decision diagram package release 2.5. 0,” *University of Colorado at Boulder*, 2012.

43. NetworkX developer team, “NetworkX,” 2014. [Online]. Available: <https://networkx.org/>
44. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, “Synthesizing Petri nets from state-based models,” in *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*. IEEE, 1995, pp. 164–171.
45. J. Carmona, J.-M. Colom, J. Cortadella, and F. García-Vallés, “Synthesis of asynchronous controllers using integer linear programming,” *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 25, no. 9, pp. 1637–1651, 2006.
46. V. Khomenko, M. Koutny, and A. Yakovlev, “Detecting state encoding conflicts in STG unfoldings using SAT,” *Fundamenta Informaticae*, vol. 62, no. 2, pp. 221–241, 2004.
47. J. Carmona, J. Cortadella, and M. Kishinevsky, “Genet: A Tool for the Synthesis and Mining of Petri nets,” in *2009 Ninth International Conference on Application of Concurrency to System Design*, Augsburg, Germany, 2009, pp. 181–185.
48. E. Gansner, E. Koutsofios, S. North, and K. P. Vo, “A Technique for Drawing Directed Graphs,” *Software Engineering, IEEE Transactions on*, vol. 19, no. 3, pp. 214 – 230, 04 1993.
49. V. Teren, J. Cortadella, and T. Villa, “Generation of synchronizing state machines from a transition system: A region-based approach,” *International Journal of Applied Mathematics and Computer Science (AMCS)*, vol. 33, no. 1, pp. 133–149, 2023.
50. P. M. Mattheakis, C. P. Sotiriou, and P. A. Beerel, “A polynomial time flow for implementing free-choice Petri-nets,” in *2012 IEEE 30th International Conference on Computer Design (ICCD)*, 2012, pp. 227–234.
51. C. A. R. Hoare, “Communicating sequential processes,” *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.
52. J. Davies and S. Schneider, “A brief history of Timed CSP,” *Theoretical Computer Science*, vol. 138, no. 2, pp. 243–271, 1995.
53. J. Sun, Y. Liu, J. S. Dong, and C. Chen, “Integrating specification and programs for system modeling and verification,” in *2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*. IEEE, 2009, pp. 127–135.
54. K. Seidel, “Probabilistic communicating processes,” *Theoretical Computer Science*, vol. 152, no. 2, pp. 219–249, 1995.
55. M. Roggenbach, “CSP-Casl—a new integration of process algebra and algebraic specification,” *Theoretical Computer Science*, vol. 354, no. 1, pp. 42–71, 2006.
56. M. Leuschel and M. Fontaine, “Probing the depths of CSP-M: A new FDR-compliant validation tool,” in *Formal Methods and Software Engineering: 10th International Conference on Formal Engineering Methods, ICFEM 2008, Kitakyushu-City, Japan, October 27-31, 2008. Proceedings 10*. Springer, 2008, pp. 278–297.
57. T. Gibson-Robinson, P. Armstrong, A. Boulgakov, and A. Roscoe, “FDR3 — A Modern Refinement Checker for CSP,” in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, E. Ábrahám and K. Havelund, Eds., vol. 8413, 2014, pp. 187–201.
58. J. Woodcock and A. Cavalcanti, “The semantics of Circus,” in *International Conference of B and Z Users*. Springer, 2002, pp. 184–203.
59. J. Davies and J. Woodcock, “Using Z,” *Specification Refinement and Proof. Series in Computer Science*, 1996.
60. C. Fischer, “CSP-OZ: a combination of Object-Z and CSP,” *Formal Methods for Open Object-based Distributed Systems: Volume 2*, pp. 423–438, 1997.
61. G. Smith, *The Object-Z specification language*. Springer Science & Business Media, 2012, vol. 1.
62. R. Acosta-Bermejo, “Rejo Langage d’Objects Réactifs et d’Agents,” Ph.D. dissertation, École Nationale Supérieure des Mines de Paris, 2003.
63. A. Charles, “Representation and analysis of reactive behaviors: A synchronous approach,” in *Computational Engineering in Systems Applications, CESA*, vol. 96, 1996, pp. 19–29.
64. F. Maraninchi, “The Argos language: Graphical representation of automata and description of reactive systems,” in *IEEE Workshop on Visual Languages*, vol. 3. Citeseer, 1991.
65. D. Harel and A. Pnueli, “On the development of reactive systems,” in *Logics and models of concurrent systems*. Springer, 1984, pp. 477–498.
66. F. X. Dormoy, “Scade 6 a model based solution for safety critical software development,” in *Embedded Real Time Software and Systems (ERTS2008)*, 2008.

67. G. Berry, "Synchronous design and verification of critical embedded systems using SCADE and Esterel," *Lecture Notes in Computer Science*, vol. 4916, pp. 2–2, 2008.
68. D. Harel, "Statecharts: A visual formalism for complex systems," *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.
69. F. Boussinot and R. De Simone, "The ESTEREL language," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1293–1304, 1991.
70. C. André, "Computing SyncCharts reactions," *Electronic Notes in Theoretical Computer Science*, vol. 88, pp. 3–19, 2004.
71. ———, "Semantics of SyncCharts," *I3S Laboratory, Sophia-Antipolis, France, Tech. Rep. ISRN I3S/RR–2003–24–FR*, 2003.
72. J. Desel, *Free choice Petri nets*. Cambridge New York: Cambridge University Press, 1995.
73. T. Nishimura, D.-I. Lee, S. Kodama, and S. Kumagai, "Decomposition algorithms for live and safe free choice nets," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 78, no. 1, pp. 1–12, 1995. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/ecjc.4430780101>
74. M. Yamauchi and T. Watanabe, "Algorithms for Extracting Minimal Siphons Containing Specified Places in a General Petri Net," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 82, no. 11, pp. 2566–2575, 1999.
75. P. Kemper and F. Bause, "An Efficient Polynomial-Time Algorithm to Decide Liveness and Boundedness of Free-Choice Nets," in *Application and Theory of Petri Nets*, K. Jensen, Ed. Springer Berlin Heidelberg, 1992, pp. 263–278.
76. J. Sparsø and S. Furber, *Principles of asynchronous circuit design*. Springer, 2002.
77. J. Carmona, J. Cortadella, M. Kishinevsky, and A. Taubin, "Elastic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1437–1455, 2009.

List of Figures

1.1	Transition system representing a car manufacturing system with a low level of detail.	4
1.2	Transition system representing a car manufacturing system with a higher level of detail, representing the production of single materials and car parts where m_1 , m_2 and m_3 represent the production of three different materials and p_1 and p_2 represent the production of car parts.	5
1.3	Petri net representation of the car factory of Fig. 1.2.	6
1.4	Petri net decomposition representing an extended version of the car factory in Fig. 1.3.	7
1.5	Petri net representing the composition of PNs in Figs. 1.4(b), 1.4(d) and 1.4(e).	8
1.6	Relationships among the models discussed in this thesis.	11
2.1	Two different FSM representations of the same system.	13
2.2	Example of transition system.	13
2.3	Example of synchronous product.	15
2.4	Petri net example.	15
2.5	Reachability graph of Petri net in Fig. 2.4.	15
2.6	PN hierarchy.	16
2.7	Modelling power of different Petri Net classes [31].	18
2.8	Example of <i>enter</i> property for event a , <i>exit</i> property for event d and <i>no_cross</i> property for event b respect to the highlighted region.	19
2.9	Framework for Petri net synthesis.	21
2.10	Example of expansion tree for event b of TS in Fig. 2.2.	23
2.11	TS before label splitting (a) and ECTS after label splitting (b).	24
2.12	Example showing the different Petri nets derived from optimal and sub-optimal label splitting.	25
2.13	Truth table for the function f	27
2.14	Binary Decision Tree for the function f	27
2.15	Reduced Ordered Binary Decision Diagram for the function f	27

2.16	Reduced Ordered Binary Decision Diagram for the function f created with a different variable order with respect to the ROBDD in Fig. 2.15.	27
3.1	Example of decomposition: PN_1 and PN_2 are derived from PN_0	33
3.2	Transition system example.	35
3.3	Three FCPNs distilled from the TS in Fig. 3.2.	35
3.4	Reachability graphs of the FCPNs in Fig. 3.3.	35
3.5	Synchronous product of reachability graphs in Fig. 3.4.	36
3.6	Two unsafe Free-choice Petri nets with a safe synchronization.	37
3.7	Synchronous product of the reachability graphs of FCPNs in Fig. 3.6.	37
3.8	Example of transition system with label splitting required for the decomposition based on regions theory.	40
3.9	SM decomposition derived from TS in Fig. 3.8.	40
3.10	Synchronous product of the reachability graphs of SMs in Fig. 3.9.	41
3.11	Example of a TS which cannot be transformed into an MG.	41
3.12	Example ECTS	42
4.1	Transition system of the current example.	46
4.2	All SMs created from TS in Fig. 2.2.	48
4.3	Excitation-closed subset of state machines selected from the SMs in Fig. 4.2 as result of the greedy algorithm.	49
4.4	LTS representing the composition $RG(SM_4) RG(SM_5)$ of SMs in Fig. 4.3.	49
4.5	ECTS.	49
4.6	SMs obtained with the MIS solver from the TS of Fig. 4.5.	50
4.7	SMs of Fig. 4.6 after the removal of label e in SM_b	50
5.1	Transition system of Fig. 3.2 and a bisimilar Petri net.	54
5.2	Four SMs distilled from the TS in Fig. 5.1.	54
5.3	Three FCPNs distilled from the TS in Fig. 5.1.	54
5.4	Current example ECTS.	57
5.5	Binary Decision Tree for EC of event f of TS in Fig. 5.4 representing the choice of the regions in the following order: r_1, r_7, r_8	58
5.6	Binary Decision Diagram for EC of event f of TS in Fig. 5.4, following the same variable order of BDT in Fig. 5.5.	58
5.7	Free-choice Petri nets derived from LTS in Fig. 5.4.	61
5.8	Example with merge of two not disjoint regions.	63
5.9	Example of the loss of free-choice property after the removal of event b	64
5.10	Choice (a) and join (b) used in the proof.	65
5.11	Removing events b, c, d and e as a collateral effect also event a is removed.	65
6.1	Trend of the exact algorithm for the decomposition of a TS compared to the approximate version.	69

7.1	Example of synchronous MSFSM architecture [23].	84
7.2	Wait State example.	84
7.3	Transition Barrier example.	85
7.4	Evolution of the synchronous approach.	86
7.5	SyncChart FSM notations [71].	87
7.6	Hierarchy representation example with SyncCharts [71].	88
7.7	Concurrency representation example with SynchCharts (2-bit binary counter) [71].	88
7.8	2-bit binary counter MSFSM.	89
7.9	4-phases to 2-phases handshake converter.	90
7.10	S-component of 4-phases to 2-phases handshake converter.	93
7.11	4-phase to 2-phase handshake converter after the creation of Non-Interactive FSMs.	94
7.12	Example showing MSFSM extraction from a Petri net.	95
7.13	MSFSM.	95
7.14	Example of a synchronous execution of the PN representing 4-phase to 2-phase handshake protocol controller (a) and the execution of synchronous MSFSM (b).	96
7.15	PN to MSFSM transformation representing a synchronous behaviour [23].	97
7.16	Initial MSFSM.	98
7.17	S-components after the MSFSM to S-component mapping.	99
7.18	Petri net derived from MSFSM in Fig. 7.16.	100
7.19	Example of transformation of an MSFSM with Mealy FSMs (a) into a Petri net (b).	100
7.20	MSFSM example.	101
7.21	Petri net derived from MSFSM in Fig. 7.20 with a Wait State represented by the place p_w .	101
7.22	MSFSM derived from PN in Fig. 7.21.	102
7.23	S-components after the integration of the self loops.	103
7.24	Petri net after the complete conversion flow, starting from Fig. 7.16.	103
7.25	Example of self-loop with the transition belonging to a single S-component where event a can be fired an infinite number of times before firing event c .	104
7.26	Example of Petri net with self-loop.	104
7.27	Petri net representation of the controller managing crossroads warehouse.	105
7.28	MSFSM derived from PN in Fig. 7.27.	105
8.1	Monolithic Transition System and its pre and post regions	111
8.2	PN derived from TS in Fig. 8.1.	112
8.3	SM decomposition of PN in Fig. 8.2.	112
8.4	MSFSM derived from PN in Fig. 8.2.	113
8.5	MSFSM derived from PN in Fig. 8.2 without FSM_4 of Fig. 8.4.	113
8.6	Reachability graphs of PN in Fig. 8.2 and MSFSM in Fig. 8.5	114

List of Tables

2.1	Minimal regions of the TS in Fig. 2.2.	20
2.2	Pre-regions and ESs for each event of the TS in Fig. 2.2.	20
2.3	Possible expansions given a set of states with a combination of properties respect to a given event.	22
3.1	Regions derived from TS in Fig. 3.8.	40
3.2	Minimal regions derived from ECTS in Fig. 3.12.	42
3.3	Excitation sets for each event of ECTS in Fig. 3.12.	42
4.1	Adjacency matrix representing the edges (value 1) between vertices of the Graph G created from the regions of the TS in Fig. 4.1.	46
4.2	Minimal regions of the transition system in Fig. 4.5.	49
4.3	Pre-regions for each event of the transition system in Fig. 4.5.	49
5.1	Minimal regions of TS in Fig. 5.4.	58
5.2	Excitation sets, pre-regions and post-regions for each event of TS in Fig. 5.4.	58
5.3	Truth table for EC of event f of TS in Fig. 5.4.	58
5.4	Sets of clauses to satisfy excitation-closure for each event of TS in Fig. 5.4.	59
6.1	TS statistics and CPU time for each decomposition step including the time spent to generate the regions.	68
6.2	Impact of each optimization step in terms of places (P) and transitions (T)	69
6.3	Number of places (P), transitions (T) and arc crossings (C) of the original transition systems vs. derived Petri nets vs. product of SMs and SM details.	70
6.4	CPU time and results of the exact decomposition algorithm.	70
6.5	Number of final SMs derived using an approximate algorithm for the search of new SMs and different approaches for the removal of redundant SMs, i.e. greedy, exact and a mixed approach.	71
6.6	Comparison between sequential SM search using previously created heuristics (sequential version) and the new approach (simultaneous version) directly encoding excitation-closure property.	72

6.7	Comparison between sequential and simultaneous FCPN search.	74
6.8	Comparison of different techniques to ensure safe FCPNs or a combination of safe SMs and FCPNs.	75
6.9	Trivial safe search performed with timeouts.	77
6.10	Results achieved on sequential safe search with the usage of counters.	77
6.11	Results achieved on sequential safe search with the usage of improved counters. .	78
6.12	Comparison between the best approaches to decompose an LTS into a set of synchronizing FCPNs with and without guarantee on the safeness of the components.	78
6.13	Comparison between standard sequential FCPN search and a variation of the same algorithm without resetting the set of learned clauses related to the FCPNs forbidden because part of a solution with multiple FCPNs.	79

List of Acronyms

ACPN	Asymmetric-Choice Petri net
BDD	Binary Decision Diagram
BDT	Binary Decision Tree
BPM	Business Process Management
EC	Excitation-closure
ECTS	Excitation-closed Transition System
EFCPN	Extended Free-Choice Petri net
ES	Excitation set
FCPN	Free-Choice Petri net
FSM	Finite State Machine
HPC	High Performance Computing
IT	Information technology
MIS	Maximal Independent Set
MSFSM	Multiple Synchronized Finite State Machine
OT	Operational technology
PN	Petri net
RG	Reachability Graph
SAT	Boolean satisfiability problem
SM	State Machine
SS	Switching set
STG	State Transition Graph
TS	Transition System
UNSAT	Boolean Unsatisfiability