TESIS DE LA UNIVERSIDAD DE ZARAGOZA

2013

126

Juan Pablo López Grao

# Contributions to the deadlock problem in multithreaded software applications observed as Resource Allocation Systems

Departamento
Informática e Ingeniería de Sistemas

Director/es
Colom Piazuelo, José Manuel



ISSN 2254-7606

Prensas de la Universidad
Universidad Zaragoza

Universidad Zaragoza
1542

Tesis Doctoral

# CONTRIBUTIONS TO THE DEADLOCK PROBLEM IN MULTITHREADED SOFTWARE APPLICATIONS OBSERVED AS RESOURCE ALLOCATION SYSTEMS


Autor

Juan Pablo López Grao

Director/es

Colom Piazuelo, José Manuel

**UNIVERSIDAD DE ZARAGOZA**

Informática e Ingeniería de Sistemas


2013

# Contribuciones al problema de bloqueo en aplicaciones software multihilo abordadas como Sistemas de Asignación de Recursos

Juan Pablo López Grao

TESIS DOCTORAL

Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Director: José Manuel Colom Piazuelo

Julio 2013

# Contributions to the deadlock problem in multithreaded software applications observed as Resource Allocation Systems

Juan Pablo López Grao

PhD THESIS

Departament of Computer Science and Systems Engineering
University of Zaragoza

PhD advisor: José Manuel Colom Piazuelo

July 2013

To Esther (I love you!),

to Juan, our little masterpiece (thanks God you resemble Mom!),

and to our much-awaited Pablo.

‘‘If you can keep your head when all about you
Are losing theirs and blaming it on you,
If you can trust yourself when all men doubt you,
But make allowance for their doubting too;
If you can wait and not be tired by waiting,
Or being lied about, don’t deal in lies,
Or being hated don’t give way to hating,
And yet don’t look too good, nor talk too wise:

If you can dream - and not make dreams your master;
If you can think - and not make thoughts your aim;
If you can meet with Triumph and Disaster
And treat those two impostors just the same;
[...] If you can fill the unforgiving minute
With sixty seconds’ worth of distance run,
Yours is the Earth and everything that’s in it,
And -which is more- you’ll be a Man, my son!”

Rudyard Kipling

# Acknowledgements

I feel very privileged to have had José Manuel Colom as my PhD advisor. In him I found a model of excellence, a devoted master, and a loyal friend. I am forever indebted to him for his discreet patience, wisdom and generosity, and I can only apologize for any time I could not live up to them.

Thanks also to José Javier Merseguer and Javier Campos, whom I luckily started working with in 2002. This was a substantial step in my career. It was a pleasure to work alongside you!

I am also grateful to Manuel Silva for granting me the honour of serving the Grupo de Ingeniería de Sistemas de Eventos Discretos (GISED), where I could meet some gifted colleagues whose excellent work also inspired the thesis. *If I have seen further...*

Many thanks to Cristian Mahulea and Jorge Júlvez, whose support and friendship was much rewarding during the long and hard journey, as well as to Miguel Ángel Barcelona, Daniel Iriarte, Luis Montesano and Yolanda Villate, who gave encouragement and enlightening advice to me when I needed them the most.

Thanks to the I3A and DIIS staffs for their willingness and efficiency, as well as to the institutions which provided financial support to our research work: namely, by means of an FPI grant from the former Spanish Ministry of Science and Technology, the CICYT-FEDER projects TIC2001/1819 and DPI2006-15390, supported by the Spanish Ministry of Science and Innovation (through former incarnations), and the project PM063/2007 supported by the Aragonese Government.

Last, but not least, thank you very much to my dearest wife Esther and little son Juan (you are my permanent inspiration!), to my parents and brothers, as well to the rest of my family and friends from the DIISasters, Waltrapas, Sabiñán (LaCuadrillica), RetroAcción/S3P, and Azulito groups. Your warm support also led me here.

# Contribuciones al problema de bloqueo en aplicaciones software multihilo abordadas como Sistemas de Asignación de Recursos

## Resumen

Desde el punto de vista de la competencia por recursos compartidos sucesivamente reutilizables, se dice que un sistema concurrente compuesto por procesos secuenciales está en situación de bloqueo si existe en él un conjunto de procesos que están indefinidamente esperando la liberación de ciertos recursos retenidos por miembros del mismo conjunto de procesos. En sistemas razonablemente complejos o distribuidos, establecer una politica de asignación de recursos que sea libre de bloqueos puede ser un problema muy difícil de resolver de forma eficiente. En este sentido, los modelos formales, y particularmente las redes de Petri, se han ido afianzando como herramientas fructíferas que permiten abstraer el problema de asignación de recursos en este tipo de sistemas, con el fin de abordarlo analíticamente y proveer métodos eficientes para la correcta construcción o corrección de estos sistemas. En particular, la teoría estructural de redes de Petri se postula como un potente aliado para lidiar con el problema de la explosión de estados inherente a aquéllos. En este fértil contexto han florecido una serie de trabajos que defienden una propuesta metodológica de diseño orientada al estudio estructural y la correspondiente corrección física del problema de asignación de recursos en familias de sistemas muy significativas en determinados contextos de aplicación, como el de los Sistemas de Fabricación Flexible. Las clases de modelos de redes de Petri resultantes asumen ciertas restricciones, con significado físico en el contexto de aplicación para el que están destinadas, que alivian en buena medida la complejidad del problema.

En la presente tesis, se intenta acercar ese tipo de aproximación metodológica al diseño de aplicaciones software multihilo libres de bloqueos. A tal efecto, se pone de manifiesto cómo aquellas restricciones procedentes del mundo de los Sistemas de Fabricación Flexible se muestran demasiado severas para aprehender la versatilidad inherente a los sistemas software en lo que respecta a la interacción de los procesos con los recursos compartidos. En particular, se han de resaltar dos necesidades de modelado fundamentales que obstaculizan la mera adopción de antiguas aproximaciones surgidas bajo el prisma de otros dominios: (1) la necesidad de soportar la anidación

de bucles no desplegables en el interior de los procesos, y (2) la posible compartición de recursos no disponibles en el arranque del sistema pero que son creados o declarados por un proceso en ejecución. A resultas, se identifica una serie de requerimientos básicos para la definición de un tipo de modelos orientado al estudio de sistemas software multihilo y se presenta una clase de redes de Petri, llamada $PC^2R$, que cumple dicha lista de requerimientos, manteniéndose a su vez respetuosa con la filosofía de diseño de anteriores subclases enfocadas a otros contextos de aplicación. Junto con la revisión e integración de anteriores resultados en el nuevo marco conceptual, se aborda el estudio de propiedades inherentes a los sistemas resultantes y su relación profunda con otros tipos de modelos, la confección de resultados y algoritmos eficientes para el análisis estructural de vivacidad en la nueva clase, así como la revisión y propuesta de métodos de resolución de los problemas de bloqueo adaptadas a las particularidades físicas del dominio de aplicación. Asimismo, se estudia la complejidad computacional de ciertas vertientes relacionadas con el problema de asignación de recursos en el nuevo contexto, así como la traslación de los resultados anteriormente mencionados sobre el dominio de la ingeniería de software multihilo, donde la nueva clase de redes permite afrontar problemas inabordables considerando el marco teórico y las herramientas suministradas para subclases anteriormente explotadas.

# Contents

# List of Tables

# List of Figures

# List of Acronyms

**AER** Associated Expansion Record.

**AGV** Automated Guided Vehicle.

**b-SPQR** borrower SPQR.

**co-NP** complement of NP.

**co-NP-complete** co-NP complete.

**co-NP-hard** co-NP hard.

**CPR** Confluent Processes with Resources.

**CSS** Cooperating Sequential Systems.

**e-Gadara** extended Gadara.

**EQ** Equal Conflict.

**ERCN** Extended Resource Control Net.

**FMS** Flexible Manufacturing System.

**GMEC** Generalized Mutual Exclusion Constraint.

**ILP** Integer Linear Programming.

**ILPP** Integer Linear Programming Problem.

**L-S$^3$PR** Linear S$^3$PR.

**NP** Non-deterministic Polynomial time.

**NP-complete** NP complete.

**NP-easy** NP easy.

**NP-hard** NP hard.

**NS-RAP** Non-Sequential Resource Allocation Process.

**OBDD** Ordered Binary Decision Diagram.

**P** Deterministic Polynomial time.

**P/T** Place/Transition.

**PC²R** Processes Competing for Conservative Resources.

**PLP** Plain Lender Process.

**PNR** Process Nets with Resources.

**PSPACE-complete** NP complete.

**QoS** Quality of Service.

**RAP** Resource Allocation Problem.

**RAS** Resource Allocation System.

**RSVP** Resource Reservation Protocol.

**S³PR** System of Simple Sequential Processes with Resources.

**S⁴PR** S$^4$PR *(not an actual acronym!)*.

**S⁴PR-BM** S$^4$PR-Bad-Marking.

**S⁴PR-DD** S$^4$PR-Deadlock-Detection.

**S⁴PR-NL** S$^4$PR-Non-Liveness.

**S⁴PR-PIM** S$^4$PR-Path-to-Initial-Marking.

**S⁴PR-RIM** S$^4$PR-Reachable-Initial-Marking.

**S⁴PR-SD** S$^4$PR-Spurious-Detection.

**S⁵PR** S$^5$PR *(not an actual acronym!)*.

**S$^*$PR** S$^*$PR *(not an actual acronym!)*.

**SAT** Satisfiability.

**SB** Structurally Bounded.

**SD** Structural Directedness.

**SIP** Structurally Implicit Place.

**SL** Structurally Live.

**S$^n$PR** S$^n$PR = {L-S$^3$PR, S$^3$PR, S$^4$PR, S$^5$PR, S$^*$PR}.

**SOAR$^2$** S$^4$PR with Ordered Allocation and Release of the Resources.

**SPQR** System of Processes Quarrelling over Resources.

# Introduction

Much has changed for computer science since RADM Grace M. Hopper allegedly found the first *computer* bug in History [Kid98]. Its presence in our society has strengthened, been diversified and interconnected to the point that the very notion of what we consider a computer has considerably blurred. The ancient chimera of an ubiquitous computing [Wei91] gradually seems to step into a tangible reality. Gradually, hand in hand with this phenomenon, we become increasingly dependent on computer systems. The scenarios in which the safety of these systems is a prerequisite are more and more diverse. The motivation for such interest transcends the economic level, often affecting the very own human security, even beyond the classic safety critical domains, such as healthcare or aerospace engineering [LT93].

While ever smaller, more lightweight and mobile computing devices become integrated into our daily lives and transgress the classical notion of *ordinateur*, the need for devices with an ever-increasing computing capacity remains intact. The increasingly more-difficult-to-keep[1] Moore's Law (which characterises the rate of increase in the transistor density on an integrated circuit with respect to the time) has prompted the emergence of multicore computing as a paradigm to stay with us for some time. In this context, parallel programming is of particular significance in order to take advantage of the computational capabilities of the new devices. On the other hand, it is no secret that, considering the current methodological approaches in software development, these systems are more error-prone.

This whole amazing revolution we are witnessing can often be neatly perceived in the proliferation of highly concurrent computer systems. In that sense, it is worth stressing the increasing importance that lies in the need to understand the problems and phenomena that emerge in such systems, as well as in the need to provide solutions and methodologies to help building increasingly more secure concurrent systems. Just as structured programming helped underpinning an orderly and modular construction of large software systems, the emergence or evolution of new paradigms of software design and development may be necessary in order to at least alleviate the problems

[1] Still, the validity of Moore's Law is being revisited by recent studies [Hel13]

and risks inherent to the construction of large concurrent software.

Until such a thing happens, the need for tools that help engineers and developers in the construction and debugging of reliable products is undeniable. Beyond the controversies over their intricacy and involved cost, it is pointless to neglect the path followed by formal methods as tools to understand the very own nature of systems and address these problems from a systemic focus. Or as sound means to, in other words, put more engineering into software engineering. In that sense, much of the success of this kind of approaches relies in our ability to bring them closer to the development teams, providing accessible and understandable tools that abstract the scientifically complex part and supplement their work throughout the product life cycle.

The software verification field attempts to answer questions about the system's compliance with properties of good behaviour, just as software validation deals with the question of whether the software is consistent with the design specification. Both branches have been widely approached from the viewpoint of formal methods. In particular, within the context of software verification, formal models (such as Resource Allocation Graphs [CES71]) have been widely exploited in the quest for characterising and finding solutions to deadlock problems that may eventually emerge in the context of concurrent software.

Unfortunately, the deadlock problem is, in general, very complex, to the extent that a perfect solution to the problem of determining whether a program may deadlock is unaffordable under the current paradigm of computation. However, this does not exclude the plausibility of finding efficient solutions to this problem addressed in a less general form. Indeed, the mathematical essence of formal methods can be useful when trying to grasp the own nature of such a complex problem.

Broadly speaking, software deadlocks can be classified into two categories. On the one hand, deadlocks induced by unordered and concurrent access to shared resources. On the other hand, deadlocks in message passing (e.g., due to flooding of message buffers), usually being the detection and correction of the latter more difficult.

In this thesis, we focus on the study of the deadlock problem due to shared resource allocation in multithreaded software. This problem becomes even more relevant in the aforementioned current context in computing, where there exists a blooming of systems with many concurrent processes that need to access shared resources in mutual exclusion; being the latter physical (e.g., databases, files) or virtual (e.g., services protected by mechanisms such as semaphores [Dij67]). Edward G. Coffman defined four necessary conditions for the existence of such kind of deadlocks [CES71] (namely: mutual exclusion, hold and wait, no preemption and circular wait). However, a general characterisation remains elusive from a structural point of view.

On the other hand, Petri nets [Mur89] have a success story as models aimed at the study of *Resource Allocation Systems (RASs)* from a systemic perspective [Col03, LZ09]. An RAS is a discrete event system in which a set of concurrent processes

coexist, and these must compete in order to be granted the allocation of some shared resources. Deadlocks arise when a set of processes is indefinitely waiting for resources that are already held by other processes of the same set [CES71]. From a qualitative standpoint, the Resource Allocation Problem (RAP) consists in meeting the demand for resources of the set of processes while dealing with the set of potential system deadlocks.

Petri nets constitute a fertile ground to deal with such deadlocks. Many real-world RASs can be abstracted into a conceptualisation constructed around two entities: processes and resources. Petri nets are constructively simple models which feature an appealing graphical representation for modelling these abstractions [Col03]. Besides, there exist powerful structural results for certain subclasses of Petri nets for RASs which enable powerful analysis and synthesis techniques for identifying and fixing potential or factual deadlocks [ECM95, PR01, TGVCE05]. In the end, the corrections computed for the model are deployed over the real-world system.

This methodology has been successfully applied to Flexible Manufacturing Systems (FMSs) where processes follow predefined production plans and resources can be artifacts such as robots, machines or conveyor belts, or passive elements such as storage area. Diverse classes of Petri net models, such as System of Simple Sequential Processes with Resources ($S^3PR$) [ECM95], $S^4PR$ [PR01, TGVCE05] and many others [JXP02, XJ99] were defined for this aim, with specific attributes for modeling different configurations of FMSs.

However, all of them prove insufficient for modelling the RAP in multithreaded software [LGC12]. Nevertheless, the structure of this category of RASs introduces new challenges due to the particularities of programming languages which will be addressed in the following chapters.

In short, the main goal of the thesis is to provide a profound and more general insight on the liveness problem in RASs with sequential processes and serially reusable resources, particularly from the perspective of dealing with multithreaded software correction. This includes the proposal of a Petri net-based design methodology, the review and integration of previous results in the new framework, the production of efficient liveness analysis algorithms for the new Petri net class, as well as the proposition of efficient methods for overcoming deadlock problems in the real system. The theoretical complexity of various aspects related to the RAP will also be explored and heuristics and relaxations will be proposed to alleviate that computational complexity.

The thesis is organised as follows:

1. A major problem in the literature with regard to the study of the RAP using Petri nets is the disparity of proposed models, often with a strong overlap in terms of its modeling capability. In Chap. 1, a state of the art review is addressed and enriched through the establishment of taxonomic categories that allow capturing the different capabilities of each kind of model. Thus, the

groundwork is laid to shed light on the shortcomings of each of these models on fully capturing the RAS vision of multithreaded software. This is materialised in the next chapter.

2. The modelling needs of this kind of systems are discussed in Chap. 2. As a result, a new class of Petri nets, named Processes Competing for Conservative Resources ($PC^2R$), is introduced. The $PC^2R$ class provides a framework in which the above results are adequately encompassed and expanded. This completes the map of the RASs with sequential processes and resources used conservatively.

   Moreover, a categorisation of the intrinsic structural and behavioural properties of $PC^2R$ and subclasses is conducted in the chapter. Unlike for other well-known classes such as free choice nets [Hil85], a direct relationship between the deadlock-freeness property (i.e., the enabling of at least one transition is ever granted) and system liveness (i.e., no transition is ever dead) does not generally exist in the context of RASs. Thus, it is possible to find nets in which some transitions are dead while other parts of the net can normally progress. This is true for almost all known Petri net models for RASs, except for very restricted subclasses. This and other properties (non-directedness, existence of home states, etc.) are characteristic of the new models and delimit the difficulties to be found in this kind of systems to implement earlier results or provide efficient solutions to the RAP. In this chapter, the properties that affect liveness in $PC^2R$ nets are studied in depth, as well as how they extrapolate (or not) to its subclasses. Additionally, transformation rules that allow us to study this kind of systems throughout syntactically simpler models are introduced.

3. Chapter 3 tackles the analysis and synthesis of multithreaded software systems modelled by means of $PC^2R$ nets regarding the liveness property. Most state-of-the-art techniques for liveness enforcing introduced in the context of FMSs are essentially based on a half-behavioural, half-structural liveness characterisation. Unfortunately, this characterisation does not apply to the more general context of $PC^2R$ nets. This is illustrated in the chapter, while new results are provided delimiting, through necessary or sufficient conditions, the liveness problem for $PC^2R$ nets. Additionally, new properties are presented and analysed that mark the point of disruption to the previously known characterisation, and show the roadpath for attempts to extend previous techniques and results on RAS models to the new application domain. The discussion crystallises in the proposal of a new methodological framework to enforce liveness on $PC^2R$ models that makes use of the particular characteristics of the new application domain. This is accomplished through the introduction of a toolbox that is offered to the software engineer so as to correct the multithreaded software system. This toolbox relies

on the application of previously acquired knowledge, such as the technique of privatisation of resources, which serves as an alternative or complement to the approximation of the classical approaches of addition of virtual resources based on integer linear programming to the multithreaded software field.

4. Gadara [WLR+09] is a subclass of $PC^2R$ for which certain assumptions are taken with relation to the nature of the decisions which the threads can take during their execution. In Chap. 4, Gadara nets are approached from the standpoint of what has been learnt so far. In parallel, a novel proof of the structural characterisation of liveness for this kind of nets is provided. Furthermore, Gadara nets are proved to be close to another subclass of $PC^2R$ nets rather exploited in the literature: the $S^4PR$ class.

5. Another contribution of this thesis is the statement of the computational complexity of various problems related to liveness analysis and synthesis in RASs. Accompanied by an appropriate in-depth literature review on previous work in RAS complexity, Chap. 5 presents new results describing the complexity of issues related to the RAP, either directly (e.g., the problem of deciding whether a net system is live, the problem of deciding whether a marking inevitably leads to deadlock) or indirectly (e.g., the problem of detecting spurious markings).

The thesis finishes with a brief summary of the conclusions of the work, as well as a discussion on the opened lines of future research.

## Some notational conventions

Throughout this thesis, certain notational conventions have been adopted which are worth further attention. For the sake of convenience, they are grouped and explained below.

### On caption location and item numbering

Five categories of items are numbered independently:

1. Chapters, sections, and subsections
2. Tables
3. Figures
4. Algorithms
5. Theorem-like annotations (namely: theorems, lemmas, propositions, properties, remarks and problems)

Chapters, sections and subsections are numbered according to the usual LaTeX convention. This means that all of them get decimal numbering (except the appendixes, which get a letter) and the numbering of sections and subsections is prefixed by the chapter number or chapter and section numbers, respectively. Items in the last four categories are numbered consecutively in order of appearance within the thesis: chapter number first, then numbered sequentially within each chapter, e.g.: Table 1.1, Figure 4.2. Note that theorem-like annotations are grouped in a single category, and therefore the numbering sequence applies to all these items; e.g. Theorem 3.4 follows Proposition 3.3.

The unique exception to the above general rule is in Chap. 4. Three transformation rules are defined in that chapter, which (for the sake of readability) are simply numbered 1, 2 and 3, without any chapter number prefix.

The format of titles and captions of tables and figures are as consistent as possible throughout the thesis. In general, they are placed under the floating element, except in the case of large tables that span through several pages. In those cases, the caption is placed over the table top, in the first page. Table headings (if any) are repeated on the second and subsequent pages. As a exception, the caption of algorithms is always provided at the top of them.

## Petri net-related notational conventions

The basic Petri net terminology and notation required to follow this thesis is described in Appendix A. However, a few remarks are in place at this point, since they affect many of the figures throughout the thesis.

First, net markings are usually denoted in the form: $[\mathrm{P}_1^{K_1}, \mathrm{P}_2^{K_2}, ..., \mathrm{P}_n^{K_n}]$, with $K_1$, ..., $K_n \in \mathbb{N}^+$. For a marking $\mathbf{m}$ denoted in this way, the naturals $K_1$, ..., $K_n$ represent the values of the vector $\mathbf{m}$ corresponding to the places $\mathrm{P}_1$, ..., $\mathrm{P}_\mathrm{n}$ of the net, i.e., $\forall i \in [1, n] : \mathbf{m}[\mathrm{P_i}] = K_i$. If $\mathrm{P_i}$ has no superscript, it is assumed that $K_i = 1$. The rest of components of the vector $\mathbf{m}$ are assumed to be zero-valued.

Sets are denoted following the usual curly bracket notation, e.g., $\{e_1, e_2, ..., e_n\}$. Consequently, the sets of markings (such as, e.g., livelocks) are usually denoted using curly brackets as well. For instance, $\{[\mathrm{A0}, \mathrm{B2}, \mathrm{C0}, \mathrm{D1}, \mathrm{R1}], [\mathrm{A1}, \mathrm{B2}, \mathrm{C0}, \mathrm{D1}]\}$ denotes a set containing the net markings $[\mathrm{A0}, \mathrm{B2}, \mathrm{C0}, \mathrm{D1}, \mathrm{R1}]$ and $[\mathrm{A1}, \mathrm{B2}, \mathrm{C0}, \mathrm{D1}]$.

Besides, p-semiflows are often represented as marking invariants in the form: $K_1 \cdot \mathbf{m}[\mathrm{P}_1] + K_2 \cdot \mathbf{m}[\mathrm{P}_2] + (...) + K_n \cdot \mathbf{m}[\mathrm{P_n}] = K'$, with $K_1$, ..., $K_n$, $K' \in \mathbb{N}^+$. In the first part of the equality, the naturals $K_1$, ..., $K_n$ represent the values of the vector $\mathbf{y}$ corresponding to each place $\mathrm{P}_1$, ..., $\mathrm{P_n}$ of the net, i.e., $\forall i \in [1, n] : \mathbf{y}[\mathrm{P_i}] = K_i$. The rest of components of the vector are assumed to be zero-valued. Meanwhile, $K'$ is the result of the weighted sum of tokens in the net for a given initial marking, where $\mathbf{y}$ determines the weighting (in other words: $K' = \mathbf{y}^T \cdot \mathbf{m_0}$).

# Chapter 1

# Resource Allocation Systems: A facet of Discrete Event Systems


## Summary

In the last years, the application of formal models, as Petri nets [Mur89], to the RAP has been a fruitful approach from a double perspective. First, thanks to the consolidation of an abstraction process of systems leading to models structured around the concepts of processes and resources, which can be easily translated into Petri nets. Second, thanks to the unveiling of analytical results characterising deadlock states, as well as methods to amend the problem. As a matter of fact, much of the success is based on the intensive use of structural reasoning on the Petri net model, unlike other more traditional approaches based on the state space under which real systems can hardly be considered. The process of abstraction enables the implementation of these methods to a wide range of engineering domains such as logistics, multiprocessor interconnection networks or distributed systems, although manufacturing is yet predominant.

Nonetheless, the assumption of syntactic restrictions with a physical meaning is a common practice for most application domains, such as, e.g., FMSs. Almost as a corollary, a myriad of different models exist, with subtle syntactic variances between. In this chapter, a consistent and structured view is provided on the different Petri net models for RASs with sequential processes, highlighting modelling capabilities independently from the target application domains. By doing so, the shortcomings of earlier approaches brought to the multithreaded software domain are made evident.

## 1.1 Introduction

Economical, spatial, technical: whatever the reason, resource scarceness is a traditional scenario in diverse systems engineering disciplines. Loosely speaking, an RAS is a discrete event system in which a finite set of (scarce) resources is shared among a set of concurrent sequential processes. From a different perspective, we consider that an RAS is a view of a system from which we study the problems related to the use of shared resources. This means that in an RAS view of a system we can ignore the explicit causal relations imposed by a process to another by means of message passing, for example. Consequently, once the resource related matters are solved, we can proceed with the next phase of the analysis/design of the system in which other facets of it are incorporated. This description or view has been successfully applied on a broad family of systems which range in disciplines such as manufacturing, distributed computing, operations research, networking or logistics. Thanks to a prior process of abstraction which is inherent in the discipline, RASs can be conceptualised under system models conceived around two distinct entities: processes and resources.

Consequently, syntax restrictions on formal models for RASs can be roughly classified according to two basic criteria [RLF97]: first, to the structure (control flow) of the concurrent processes; second, to the way the resources are used by these processes. Beyond this coarse-grained categorisation, the variety of domain-specific physical constraints has led to a proliferation of works which study subtly different abstract models for RASs with strong structural restrictions. Most often these feature an expressive power well adapted to specific application domains, while powerful solutions are known for them. From this perspective, and taking into account some usual physical constraints in the context of flexible manufacturing systems (sequential processes, non-consumable resources, etc.), the success of some Petri net-based methodologies for studying the deadlock problem seems well justified.

Generally, the aforementioned abstract models are used to study the RAP: the procedure in serving the processes requirements for resources, according to their own resource usage policy, while accomplishing a certain goal. In fact, such a generic definition encompasses different problems from a qualitative or quantitative standpoint. In quantitative terms, the concept relates to the ability of optimising a system performance function [KS89]. In qualitative terms, it is widely associated to satisfying successfully the requests for resources made by the processes, ensuring that no process ever falls in a deadlock [LT79]: the focus of this thesis.

Although other models of concurrency have also been considered [FMMT97], Petri nets [Mur89] have arguably taken a leading role in the family of formal models used for dealing with the RAP [ECM95, ER04]. One of the strengths of this approach is the smooth mapping between the main entities of RASs and the basic elements of Petri nets.

This fact is well recognised in the domain of FMSs, where Petri net models for RASs have widely succeeded since the seminal work on the matter was introduced [ECM95]. This is based upon two solid pillars: 1) the definition of a rich syntax from a physical point of view, which enables the natural expression of a wide disparity of plant configurations; and 2) the contribution of sound scientific results which let us characterise deadlocks from the model structure, as well as provide a well-defined methodology to automatically correct them in the real system.

Section 1.2 focuses on RASs from the point of view of the key role played by the process of abstraction in obtaining tractable models for different application domains. In that sense, common elements in the abstraction of processes and resources are categorised from a systemic point of view. This is a relatively unexplored approach to the problem, since most works start from the specific study of an application domain. From this general framework, the different state-of-the-art Petri nets classes are categorised in Sect. 1.3. Finally, in Sect. 1.4, the class of $S^4PR$ nets is introduced as a starting paradigm of the methodology. To this aim, a simple example from the field of FMSs is presented.

## 1.2 The role of abstraction

As introduced above, the methodological approach discussed in this thesis is based primarily on obtaining manageable formal models from the observation of real-world systems. To this end, we must discard those irrelevant details for the property to be studied (in this case, for the emergence of deadlocks due to an inappropriate resource allocation sequence), while preserving those relevant aspects. Such process is known as the *process of abstraction* [GW92].

In this sense, an RAS constitutes a facet of a real world system [Zei84], i.e., a particular, restricted, non-exclusive view of it. For instance, in the context of multithreaded software, deadlocks can occur not only due to some resource allocation order, but also to message passing between different threads. Within the scope of this thesis, an RAS view of such systems will be addressed in which details of the message passing are ignored. Therefore, this kind of deadlocks is not reflected in the resulting models. The philosophy of the methodology proposed in this thesis consists in, as discussed in Chap. 2, first resolving the deadlock problems through an RAS vision, moving later to address the deadlock problems caused by other communication/synchronisation paradigms. In short, an RAS is just the intellectual product resulting from the application of a process of abstraction on the system studied.

The importance of the abstraction process must be emphasised because in fact it implies a particular methodology for the design of complex systems. In effect, a system is not a pure RAS in general because of processes that impose causality relations to other processes in the system. Consequently, when we retain processes

and resources ignoring the rest of relations in the system we advocate for analysing and correcting the problems associated to the use of resources. Once this goal is accomplished, the designer can concentrate in other systemic problems, such as those related to performance, scheduling or others. In fact, this is the design flow assumed in this thesis. This approach to system design is the deep reason to promote structural methods to correct or fix problems. Indeed, such methods can survive to further steps in the design concerning performance or optimisation.

Through the process of abstraction we obtain higher-level items such as resources and processes, and relations between them. The concept of what is and is not a process (or resource) is dependent on the application domain to be studied. In fact, although in this section we discuss the process of abstraction as an entity which is abstract in itself, in practice it makes no sense to speak of a single process of abstraction. On the contrary, it would be more reasonable to speak of a family of them, each one closely linked to an application domain and even sometimes to a particular case study.

In this regard, it may suffice to exemplify the last observation through a particularity. In the process of abstraction for the study of the RAP, the fact that an element of the real system is considered a resource often makes practical sense only if it intervenes, or may intervene, in situations of deadlock. For this it is necessary (though not sufficient) that the resource is held at some time while waiting for the allocation of a different resource so as that the system (or part thereof) can progress: a property known in the realm of operating systems as *"hold & wait"* [CES71]. Otherwise, for the sake of conciseness and manageability of the resulting RAS model, it is unnecessary to consider the item as a resource in it.

On the other hand, given that the steps of a process that are relevant to the RAP are somehow related to the set of resources involved, the aforementioned issue can also affect the size or structure of the processes (understood as abstract entities). Or, even, it might involve their disappearance from the model if no other resource interacts with them. In short, the concept of process and resource is heavily dependent not only on the application domain, but also on the particular conditions of the problem to be studied.

Given the foregoing, it is still possible to sketch an outline of the main steps that constitute the process of abstraction to obtain an RAS. Broadly speaking, the common features in the abstraction of systems of very diverse nature could be catalogued according to the following road map:

1. *Resource identification.* The Collins English Dictionary defines a resource as 'a supply or source of aid or support; something resorted to in time of need' or 'a means of doing something' [Col11]. A first important consideration is that the nature of this medium can be logical, and as such, there is no need for a visibly located physical embodiment of an element to be considered as a resource from an RAS point of view. A simple example of this in the field of manufacturing

systems is the consideration of storage space as a system resource. Indeed, the lack of storage space for a part in transit in the manufacturing system can be a cause of circular waits in the system and therefore of deadlocks. It is also possible to consider a group of physically dispersed elements as a single resource for the sake of the study of the RAP if all of them are always used in unison under the same circumstances. The possibilities are very diverse under the interpretation of various application domains.

Another relevant consideration concerns the way a resource is seen with respect to its interaction with the rest of the system. From an RAS point of view, resources are passive elements in the sense that their behavior can be summarised in two states (allocated or free) and that the change from one to another state can be fully explained from the evolution of the rest of the system. In fact, the latter behaviour should apply for all resources considered in the system.

Considering the above, two types of high-level operations can be identified: *assignment operations*, which wait until the resource is free and switch its status to assigned, and *release operations*, which switch the resource status from allocated to free. It is important to note that the assignment operation thus conceived is a blocking operation (i.e., if the resource is not free, then the system, or that part of the system, wait for it to be). Therefore, a parallel identification of operations that interact with the resources is important in order to decide what should be considered a resource in our high-level abstraction.

Another fundamental aspect to consider is whether the resources can be grouped into sets of resources that can be used interchangeably at all times. In that sense, returning to the previous example in the context of manufacturing systems, we could quantify the storage space in a given buffer content, where a single resource would be considered the gap that a piece would fill by being placed at the buffer. Obviously, all gaps should stand on equal footing for all system operations, so that any free resource of the set can be chosen whenever one of these is required. When we find such an scheme, we can reach a higher level of abstraction that simplifies the vision of the system: the existence of *resource types*.

2. *Process identification.* The Collins English Dictionary defines a process as 'a series of actions that produce a change or development' or 'a method of doing or producing something' [Col11]. Unlike resources, processes are the active elements of the system from an RAS point of view. Processes are patterns of behavior of the system, potentially repetitive and often sequential, that are observable from a local perspective. Note that the condition of locality here does not refer to a physically contiguous location for the operations that compose it, but to the existence of a certain order relation between them or between groups of them, so that their repetition is closely linked. Or, more appropriately, to

the possibility of discerning a control flow that relates them logically.

At this point, one may notice a new category of significant operations beyond those of resource allocation and release. These operations are those that alter the control flow of a particular process, either breaking or restoring their sequentiality (e.g., job splitting/merging operations in the context of FMSs) or by establishing alternative execution paths or reunifying them (e.g., chain branching). Such operations can also be coupled with those above, so that the alteration of the control flow is subject to the allocation of certain resource(s).

In line with this last point, it should be noted that, from the point of view of the processes, the allocation and release operations may require different resource types, or multiple instances of the same type, simultaneously. This is another question that must be evaluated in order to characterise the process of abstraction of the system and finally determine the type of model required to properly study the problem.

Finally, processes following the same behavioural pattern may concur in the system. Similar to the approach undertaken with resources, this observation leads to the introduction of a higher level abstraction that can result in more compact models: the existence of *process types* that may or may not have a certain capacity (i.e., maximum number of concurrent instances).

3. *Modular construction of the model.* A modular abstraction, and thus ending up with a modular model, is natural in such abstractions. This is because RASs consist of multiple concurrent active entities that, if they had enough resources to progress, could ignore each other. As resources are limited, the construction is done by overlapping entities via use of shared resources.

   Essentially, the modelling stage consists in the embodiment of the product of the abstraction process through a formal language. Thus, strictly speaking, modelling cannot be considered part of the process of abstraction. Nonetheless, it is still a process closely linked to the process of abstraction, to the extent that the latter is often consciously or unconsciously influenced by the type of target model. Note that in other abstractions modularity is not evident as a constructive principle, but that is not the case of RASs. In that sense, the choice of a formal model with restricted expressiveness can derail the study of the desired property. The versatility and modularity of Petri nets, however, seems to provide a natural setting which is suitable for the expression of the various entities involved in an RAS in due complexity. This issue is revisited in Sect. 1.3.

   In the case of Petri nets, the modeling methodology is based on three steps: characterisation of the processes, incorporation of the resources, and construc-

tion of the complete model by fusion of all shared resources. In that sense, the construction of the model is bottom-up, allowing the designer to focus on the understanding of the system in a disaggregated way. In the end, a model is reached that allows to study the problems of the system as a whole.

4. *Identification of the minimally admissible operating conditions* The establishment of some minimum operating conditions is often a prerequisite in the design of these systems, and therefore inherent in the definition of the resulting models. In fact, in the context of RAS modelling it is very natural to require initial states in which resources are not used (cold start). In such cases, we say that the system is in the *idle state*[1]. Also, it is often very natural to have enough resources in the system to grant the execution of each process path in isolation (where a process path is considered as a sequence of operations which occurrence represents the successful execution and completion of one single process). Often, these are minimum operating conditions that can and should be established from the beginning because the system would not work otherwise. Proceeding thus, it suffices to study and correct the problems caused by competition for shared resources.

Based on the above, one can extrapolate certain aspects that allow a rough classification of the RASs into various categories. These aspects largely determine the shape of the resulting model. In the case of modeling using Petri nets, they can impose syntactic constraints that may simplify the analysis and synthesis of the resulting models. Among these decisive aspects the next ones are specially remarkable: (i) Resource conservativity; (ii) Categorising of the resources into resource types within which any instance can be used interchangeably by the processes; (iii) Routing flexibility in the control flow of the processes; (iv) Internal execution cycles within the processes; (v) Sequentiality of the processes; (vi) Resource-dependent decisions/merges/forks/joins within the processes; (vii) Joint acquisition requests of different resources by a single process; (viii) Process types; (ix) Existence of minimally admissible operating conditions. These and other properties will be studied further in Subsection 1.3.1 from the perspective of the classification of Petri net models for RASs.

## 1.3 The resource allocation problem through Petri net models

The traditional usage of raw Petri nets for modelling RASs was essentially disrupted in 1995, when the publication of a seminal work [ECM95] triggered the integration of

[1] In Petri net terms, all tokens are in the idle places of the processes and in the resources; see Sect. 1.4.

methodological aspects in the construction of models. Conceptual objects of higher level than places and transitions like processes were incorporated. Also, a semantics connected to the physical system was introduced at the proper definition of the new models, addressing the interpretation of the own models, their objects and the analysis results. Finally, modularity was introduced as a design principle both in the definition and construction of the models. In the end, these new models belonged to novel subclasses from a syntactic point of view, being the definition of these subclasses directed by the application domain.

Precisely, one of the pillars on which rests the success of these Petri net-based approaches for the study of the RAP is the existence of a concise, natural and intuitive way of expressing the functional entities of a typical RAS abstraction (processes, resources, etc.) by means of Petri net artifacts, considered both from a fine-grained perspective (places, transitions) and from that of more structurally complex elements (p-components, circuits, state machines, etc.).

For instance, a resource type can be modelled using a place: the number of instances of it being modelled with tokens. Meanwhile, sequential processes are modelled with tokens progressing through state machines. Arcs from resource places to transitions (from transitions to resource places) represent the acquisition (return) of some resources by a process. In the end, the process state machines can be merged into a model of the whole system via fusion of the common resource places. In summary, Petri nets provide a natural formal framework for the modelling and analysis of RASs, besides benefiting from the goods of compositionality.

In this section we examine how the different classes of Petri nets for RASs respond to the modelling needs arising from the abstraction process.

### 1.3.1 An overview of model features

In Sect. 1.2, a catalogue of generic properties that can be found in the abstraction of an RAS was presented. These find their starting point in those observed for FMSs in a previous work [Col03]. Next, it is intended to introduce a list of (Petri net) model features that are closely related to those generic properties. To this end, it is necessary to introduce a few instrumental concepts allowing us to discuss the model structure. The first concept is that of process path. Process paths are used to discuss the structure of processes, and later the syntactical implications they have on the classes of Petri net models derived from such discussion.

A *process path* is a sequence of operations whose occurrence represents the successful execution and completion of one single process. Obviously, the sequence of operations belongs to a unique process type. In the context of FMSs, for instance, a process path is a production sequence that generates a finished product from some raw material(s) following a production plan. Meanwhile, in the context of multipro-

cessor interconnection networks with wormhole routing, a process path is a sequence of flit transmissions such that the sending of a whole single message from a source node to a (set of) target node(s) is successfully completed.

Moreover, a few more concepts must be defined prior to the main discussion. Given a Petri net model for RASs, two kinds of subnets can be clearly identified considering the main elements of an RAS, i.e., processes and resources. Those are the process subnets and the resource subnets.

A *process subnet* is a subnet of the given Petri net model for RASs which models all the possible evolution of a single process, i.e., all the process paths that it can execute. In general, each process subnet models a different process type. Therefore, process subnets are mutually disjoint. Under this interpretation, several process instances (modelled by tokens) can concurrently execute that process type by being concurrently moved through the same process subnet. Usually, well-known Petri net subclasses such as strongly connected state machines or marked graphs are used to restrict the syntax of the process subnet structure. These syntax restrictions are usually derived from the application domain, and are fundamental to confront the inherent complexity of the RAP in the most general case.

Meanwhile, a *resource subnet* is a subnet which models one resource type along with all the stages of process paths in which resources of this type are used (and the operations that allocate/release those resources). When the use of resources is conservative, the resource subnet is a strongly connected subnet in which the set of places is the support of a special (minimal) p-semiflow called the resource p-semiflow (which is defined below). In contrast to process subnets, resource subnets are not mutually disjoint in general.

Finally, a *resource p-semiflow* is a minimal p-semiflow which describes an invariant relation of use of instances from a certain (unique) resource type. In other words, this invariant relation rules how such resources are used by the processes. As explained above, there exists a tight relation between a resource p-semiflow and the corresponding resource subnet.

The model features (or properties) presented next are catalogued into three main categories. A subsubsection is devoted to each one of these categories.

#### Properties for the whole system

In the RAS abstraction of a system, there exist two fundamental types of participating entities: processes and resources. The following properties capture the support to the multiplicity and heterogeneity of the processes which concur in the system, as well as to those of the corresponding system resources.

- *Unique process subnet.* This property answers the question: do all the concurrent processes follow the same behavioural pattern? In other words, can all the

inactive processes eventually execute the same process paths? If that is not the case, then we say that there exist several process types. Each process type is usually modelled by a different process subnet.

- *Closed process subnets.* Instead of having one process subnet per process, RAS models are usually compacted by allowing several tokens within the same process subnet. This is often accomplished by establishing a structural, non-binary bound of the number of concurrent process instances of the corresponding process type. This bound is enforced by the initial marking of the so-called *idle place* of the process subnet. When such a place does not exist, we say that the process subnet is open [GV99] and the number of concurrent instances is only limited by the available resources, yet no deliberated restriction is imposed on the number of concurrent processes; i.e., it models an open system.

- *Binary resources.* As explained earlier, resource types are usually modelled using a place. This place is usually called the *resource place*. Usually, the initial marking of that place (assuming that the processes are in a idle state) establishes a capacity for the corresponding resource type. If that capacity is limited to one, then we speak of binary resources. Otherwise, a number of available instances of that resource type may be granted to any requesting process. These resources are used in equal footing, i.e., it is assumed that they can be used interchangeably.

**Structural properties for each process type**

Below are some properties that the structure of each process of an RAS model can verify *when considered in isolation*. In some cases, they are desirable system properties that most RAS models meet by construction. In other cases, they are more specific characteristics aimed to adapt the modeling capability to the needs of a particular application domain.

- *Reproducibility of t-semiflows.* Overall, this is a usually desirable property in the RAS field. Not in vain, this is related to the possibility of repeating the successful execution of a process, since the process paths in these models are usually captured by t-semiflows. Consequently, the property is supported by practically all Petri net classes for RASs. However, it should be pointed out that some t-semiflows do not capture any process path: that is the case when there exist internal cycles in the system (e.g., recirculating circuits in FMSs). On the other hand, the concept of reproducibility is usually captured by the concept of t-semiflow, although this is not a strict rule of thumb, not even when the net is consistent[2]. However, all t-semiflows are reproducible for well-

[2]In Chap. 2, Fig. 2.18 depicts a Petri net such that no minimal t-semiflow is reproducible

known Petri net classes such as marked strongly connected state machines or live strongly connected marked graphs. For instance, in $S^4PR$ nets, each process path is a minimal t-semiflow, and every minimal t-semiflow is reproducible since processes are marked strongly connected state machines. $S^4PR$ nets are thoroughly revisited in Sect. 1.4.

- *Consistency.* In conjunction with the above property, it seizes a usually desirable situation in the RAS context, since this relates to the repeatability of every process. This property answers the question: can a process type be fully explained as a composition of its process paths? In terms of Petri nets, the property is satisfied, as long as each process path is a t-semiflow, if the net is consistent. Again, practically all classes of Petri nets for RASs are consistent by construction.

- *Equivalence of t-semiflows.* This property refers to the fact of each t-semiflow being executable in isolation from the idle state. By the expression *idle state* we denote the system state in which no process is in an intermediate step of execution. This usually coincides, in general, with the initial state of the system. This is a common and often desirable property in Petri net models for RASs when every t-semiflow represents a process path, since it refers to the possibility of fully executing any process path without triggering the execution of other processes. In that sense, it is often accompanied by the two previous properties. However, it is not usually satisfied when there exist internal cycles within the control flow of processes, as those cycles are usually captured by t-semiflows which are only activated when the process reaches a certain intermediate point of execution. Typically, the problem of the existence of cyclic behavior inside processes severely complicates liveness analysis in the general case. However, some classes of models can syntactically address the problem in, at least, a partial way, as shall be seen in the next section. In particular, Chap. 2 shows that the context of the RAS vision of multithreaded software is a good example of a situation in which it is usually necessary to consider the existence of internal cycles within the processes (such as loops, for example) in order to successfully address the problem of occurrence of deadlocks.

- *Process termination.* This property is verified if, given a process in an intermediate execution state, its execution can always be concluded in such a way that the idle state is reached. In the side of Petri nets, this is held if the idle state is a home state for each process type. In most Petri net classes for RASs, the idle state is represented by the empty marking when the so-called idle place is omitted (the concept of idle place will be defined in Sect. 1.4). This property is therefore strongly related to that of the reproducibility of the empty mark-

ing [Lau02]. Again, this can be seen as a property of good behaviour of systems, and therefore it is usual among those classes.

- *Process liveness.* Another interesting property that most RAS models fulfill by construction is the liveness of all transitions that compose each process. Thereby, liveness problems only arise when adding the system resources. In other words, it is ensured that the analysis of liveness problems can be limited to those emerging from the order of allocation of shared resources used in mutual exclusion by the processes, and no other cause whatsoever. This property is satisfied if the idle state is a home state and all processes are consistent, which is a commonplace in the RAS context.

- *No internal decisions.* Essentially, this property refers to the existence of multiple (non-disjoint) process paths within a single process. More precisely, it points to the possibility of finding conflicts (i.e., decisions) within the control flow of the processes. In the context of FMSs, where such decisions appear we speak of 'routing flexibility'. If no conflicts exist, there can only exist non-disjoint process paths if path reunification is allowed. However, these reunified paths are independent from the very start of the process until they merge, and never split up again. In other words, in such kind of RASs the process paths are permanently confluent. This often derives in a considerable simplicity in the analysis of the RAP when the resources are aggregated. When neither internal decisions nor reunifications are allowed, it is said that the processes are linear [EGVC98].

- *Process sequentiality.* A severe source of problems when addressing the RAP from a structural perspective arises when some processes are eventually subdivided in concurrent subtasks (through fork-like operations), and/or reunite the latter (through join-like operations). In such cases, we usually speak of Non-Sequential RASs [Rev99] in reference to the non-sequentiality of some of the processes that comprise them. In case all the abstracted processes are conceived as sequential processes, the system is referred as a Sequential RAS.

**Properties regarding how processes use the resources**

- *Resource conservativity.* The conservative use of resources by the set of processes considered individually is a commonplace in the abstraction of RASs. This requirement must be met for every resource that each process interacts with during its execution. Then each of these resources belongs to the support of (at least) one minimal p-semiflow that defines the positive invariant relation that characterises the way the resource is used by that process: the resource p-semiflow. Furthermore, if every place of the subnet that models the process

belongs to some p-semiflow (and resources are used conservatively) then the system is structurally bounded.

- *Ordinary resource subnets.* This property essentially refers to the simplicity of the resource request/release operations. If such operations allow requesting/releasing multiple instances of the same resource type, then the corresponding arcs in the resource subnets are weighted (i.e., some subnets are non-ordinary). In general, the more complex the allocation operations, the more difficult the model is to analyse.

- *Binary resource p-semiflows.* Stricter than the preceding one, this model restriction is applicable when processes never request several instances of the same resource type, neither simultaneously nor in an successive requests. Instead, an allocated resource must be released prior to any other allocation request of a resource of the same type.

- *Orthogonal resource p-semiflows.* The orthogonality of the resource p-semiflows implies that a process can only simultaneously use resources of a single resource type. Prior to requesting resources from a different resource type, it must release all its allocated resources. This property often comes coupled with the preceding one [ECM95, EGVC98]. When both restrictions apply, at most one resource can be allocated to each process at a given state of execution.

- *Resource-independent internal conflicts.* This property holds when each conflict belonging to a single process path is fully non-deterministic with respect to the allocation state of resources. In other words, resource allocation operations are conceived as strictly blocking operations: whenever a process requests some resources, it cannot advance in whichever other way until that precise request is granted (and no other set of available resources can serve as an alternative).

- *Resource lending.* In some application domains, it may be necessary to consider the fact that a resource can be created by some process, shared with other processes, and finally destroyed by the same process that created it. This is done in such a way that there still exists an invariant relation that ensures that these new resource is serially reusable by the concurrent processes. Notice that under certain circumstances (namely, when the process subnets are open) this invariant relation may no longer be captured by the concept of resource p-semiflow, and certain specific p-flows are used instead. All of this is further discussed in Chap. 2 from the perspective of the modelling of multithreaded software systems. This phenomenon is labelled as *resource lending*, since some resources are somewhat lent by some 'server' processes to other 'client' processes.

- *First allocated, first released.* This property establishes a link between the allocation operations and the release operations within every single process path, in such a way that resources are released in the same order as they were allocated. In many cases (e.g., as that of multiprocessor interconnection networks) this is a natural restriction imposed by the application domain, as discussed in Subsection 1.3.3.

### 1.3.2 Petri net models for RASs

Despite the existence of other works based on formal models [FMMT97], Petri nets have proven to be an especially useful tool for the modelling, analysis and synthesis of RASs with serially reusable resources [LT79, ECM95, GV99, JX01, PR01, ETGVC02, Tri03, Col03, ER04, JXC04, TGVCE05, LDZ06].

With respect to the structure of the concurrent processes, most of the current work focuses on Sequential RASs, as opposed to Non-Sequential RASs, in which assembly and disassembly (fork / join) operations are allowed within the processes. However, some studies have attempted to approach Non-Sequential RASs using Petri nets as modelling paradigm. Classes such as Non-Sequential Resource Allocation Process (NS-RAP) [ER04], Extended Resource Control Net (ERCN) merged nets [XJ99], ERCN$^*$ merged nets [JXC04] or Process Nets with Resources (PNR) nets [JXP02] extend the capabilities of conventional models beyond Sequential RASs by way of lot splitting or merging operations. Unfortunately, finding effective solutions for these systems is, in general, much more complicated [XJ99].

As for Sequential RASs, which are the subject of study in this PhD thesis, it is worth mentioning the disparity of Petri net models that have emerged successively, often extending previous results and thus extending the subclass of systems that can be modelled and studied. The Cooperating Sequential Systems (CSS) class is one of the first classes designed to study the RAP in Sequential RASs [LT79]. In CSS, there is only one type of process, i.e., all processes share identical structure of stages or steps to execute. These processes may compete for multiple resource types, with possibly multiple instances of each type. In more recent works on Sequential RASs, the existence of multiple process types is supported, usually allowing different routes or alternative execution plans. In some of these models, however, the execution plan of a process is chosen at the launch of its execution, and remains fixed throughout the course of it [FMMT97]. Other works overcome this limitation, supporting routing decisions at runtime. In particular, so it is with the seminal S$^3$PR class [ECM95]. However, the processes of an S$^3$PR net can reserve at most one single resource in each stage or execution step. Also of note is the existence of subclasses of S$^3$PR (like Linear S$^3$PR (L-S$^3$PR) nets [EGVC98]) with interesting structural properties.

The restriction on the resources usage per process is overcome by the S$^4$PR

class [Tri03, TGVCE05], also named $S^3PGR^2$ [PR01], which is a superclass of $S^3PR$. This class allows runtime routing decisions, as well as the simultaneous reservation of multiple resources of different types by a single process. These resources can be released in an arbitrary order. Systems of this type are occasionally named Disjunctive-Conjunctive RASs [RLF97]. Particularly, the $S^4PR$ class has received special attention because it can deal with a fairly general class of Sequential RASs, and efficient characterisations exist for deadlock situations [Tri03]. Although most of these works emphasise the application in FMS, the use of a purely systemic approximation allows to apply these models, as well as the techniques developed for analysis and synthesis [ETGVC02, TGVCE05], to very different application domains.

Another interesting subclass of $S^4PR$ is that of $S^4PR$ with Ordered Allocation and Release of the Resources ($SOAR^2$) [Rov11]. In this case, the resources are managed in a "first allocated, first released" basis by the processes, and the model structure is restricted to adhere to this particularity. This characteristic is inherent to some application domains such as wormhole communication in multiprocessor interconnection networks. This is further discussed in Subsection 1.3.3.

At present, the most general class of Petri nets for Sequential RASs is the $S^*PR$ class [ETGVC02], in which processes are ordinary state machines with internal cycles. However, deadlocks in $S^*PR$ net models are not fully comprehended from a structural perspective. A more detailed study on the relationships between the different kinds of nets was published in a previous work [Col03]. It should be noted that for all the above classes except for the $S^*PR$ class, there exist liveness characterisations based on the absence of certain types of partially unmarked siphons. The structural nature of such characterisations opens a door to the efficient detection and correction of deadlocks, by implementing controllers (usually by adding new places) that restrict the behaviour of the net, preventing the reach of undesirable markings.

There exist, however, other studies that try to address the problem of processes with internal cycles, always starting from certain restrictions on the approach of the $S^*PR$ class regarding the use of resources within those cycles [JX01, JXC04, LDZ06]. One of these Petri net subclasses is that of Gadara [WLR+09]. Gadara nets are presented as a model for studying the RAP in multithreaded software systems sharing a set of binary locks. Gadara nets therefore feature binary resources and internal cycles are allowed within the structure of the processes. However, all conflicts in the process paths of a Gadara net are resource-independent. Unfortunately, it is not hard to find real-world systems in which the above restriction does not apply. This deficiency is addressed along this thesis in the framework of the study and exploitation of the RAP in the context of multithreaded software. In particular, Chap. 4 carefully approaches Gadara nets from a critical position.

Finally, the class of System of Processes Quarrelling over Resources (SPQR) [LGC06] notably diverges from the previous approaches in an attempt to

generalise the systems considered in the traditional modelling of FMSs from an RAS perspective. This generalisation is essentially driven in two directions. First, the process subnets are open systems. Second, resource lending is allowed. SPQR nets will be thoroughly studied in Chap. 2.

Table 1.1 relates the Petri net classes mentioned above with the model properties presented in Subsection 1.3.1. The table draws a more complete picture of the relationships between them from an essentially syntactic viewpoint.

### 1.3.3 Deployment of the RAS vision in different domains

Having reviewed the steps which are necessary to realise the process of abstraction, in this section we focus on the observation of some application domains in which the RAS approach has led to relevant results. In this course of action, we identify the basic functional entities which are abstracted in the process. Consequently, those aspects which were previously introduced from a conceptual perspective are here materialised.

#### The RAP in Flexible Manufacturing Systems

The problem of deadlocks due to an inappropriate resource allocation sequence is very well-known in the field of FMSs, where the RAS approach has a long, successful road [Col03].

In this area, an RAS would be a view of a manufacturing cell where resources are often machines or tools that are used to achieve the manufacturing process. Such tools can be fixed processing elements such as workstations, as well as transport mechanisms such as robotic arms or conveyor belts. As noted above, it is also possible to consider the space in containers or pallets as system resources used to hold parts in the process. Figure 1.1 depicts a production cell with two robotic arms and three workstations, as well as conveyor belts for the input and output of material.

On the other hand, the processes of the RAS would be the parts being processed in the system. Each of these parts follows a prefixed production plan which is divided into different stages where operations are executed. These operations can be classified into different types closely related to the various types of resources mentioned above (transformations, handling, storage, etc.). Such production plans would be our "process types". For example, a process type would be the production plan for the cell of Fig. 1.1 in which, first, parts from I1 are taken to M1 through R1. Second, the arm moves them, when processed, from M1 to M2. And finally, after the processing in M2 is completed, R1 moves them to O1 so as to leave the cell. Parts following different production plans but sharing a nonempty set of resources can concur in the system, which may be a source of deadlocks.

At this point, it is important to discern whether parts can be assembled and disassembled in the system. Assembly/dissassembly operations can have a dramatic impact

| | CSS [LT79] | L-S$^3$PR [EGVC98] | S$^3$PR [ECM95] | SOAR$^2$ [Rov11] | Gadara [WLR$^+$09] | S$^4$PR [TGVCE05] | S$^*$PR [ETGVC02] | PNR-nets [JXP02] | ERCN$^*$-merged [JXC04] | NS-RAP [ER04] | SPQR [LGC06] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Properties of the whole system | | | | | | | | | | | |
| Unique process subnet | ✓ | × | × | × | × | × | × | × | × | × | × |
| Closed process subnets | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × |
| Binary resources | × | × | × | ✓ | ✓ | × | × | × | × | × | × |
| Structural properties of the processes | | | | | | | | | | | |
| Reproducibility of t-semiflows | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | ✓ |
| Consistency | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Equivalence of t-semiflows | ✓ | ✓ | ✓ | ✓ | × | ✓ | × | ✓ | × | × | ✓ |
| Process termination | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | ✓ |
| Process liveness | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | ✓ |
| No internal decisions | ✓ | ✓ | × | × | × | × | × | × | × | × | × |
| Process sequentiality | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | × | × | ✓ |
| Properties about the use of resources | | | | | | | | | | | |
| Conservativeness | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Ordinary resource subnets | × | ✓ | ✓ | ✓ | ✓ | × | × | ✓ | ✓ | × | × |
| Binary resource p-semiflows | × | ✓ | ✓ | ✓ | ✓ | × | × | × | × | × | × |
| Orthogonal resource p-semiflows | × | ✓ | ✓ | × | × | × | × | × | × | × | × |
| Resource-independent conflicts | ✓ | ✓ | × | × | ✓ | × | × | × | × | × | × |
| No resource lending | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × |
| First allocated, first released | × | ✓ | ✓ | ✓ | × | × | × | × | × | × | × |

**Table 1.1**: Modelling capabilities of some well-known Petri net classes for RASs

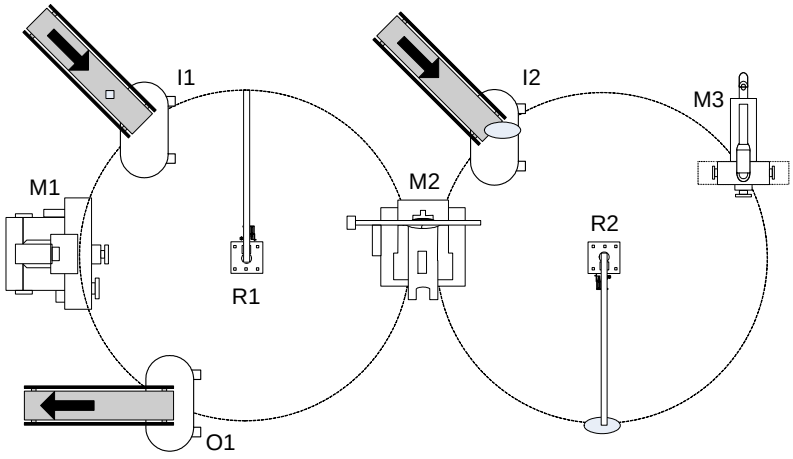



**Figure 1.1**: Layout of a manufacturing cell

on the complexity of the analysis of the problem, as discussed in Subsection 1.3.2.

Furthermore, it is important to note that in such systems the production plans often have a very specific structure, in which all operations, or large groups of them, are fully or partially ordered. Besides, it is unusual to find internal cycles, as recirculating circuits; or if any, there are usually few of them, having no major ramifications and being well located. In general, the predominant element regarding changes in the process flow is the routing of parts into different production branches. Again, this peculiar type of structure has been well exploited by the application of formal methods (and particularly Petri nets) to the study of the RAP.

Most of the Petri net models presented in Subsection 1.3.2 have been conceived with FMSs in mind as the target application domain.

From a modelling point of view, the problem of integrating assembly/dissassembly operations has been approached in several works. Classes such as NS-RAP [ER04], ERCN merged nets [XJ99], ERCN$^*$ merged nets [JXC04] or PNR nets [JXP02] were proposed in the context of approaching the RAP in FMSs. However, structural liveness enforcing approaches can be computationally demanding in this scenario, as evidenced for augmented marked graphs [XJ99]. An insight on the computational complexity of such approaches on the Sequential RAS context is driven in Chapt. 5.

In most recent works in the context of Sequential RAS, different process types with multiple concurrent instances are allowed, often allowing on-line routing decisions into different production branches. In particular, such kind of FMSs are dealt with the seminal S$^3$PR class [ECM95]. However, processes in a S$^3$PR net can use at most a single resource unit at a given state. The same applies for its subclass L-S$^3$PR [EGVC98], also conceived to approach certain FMSs. Although more restric-

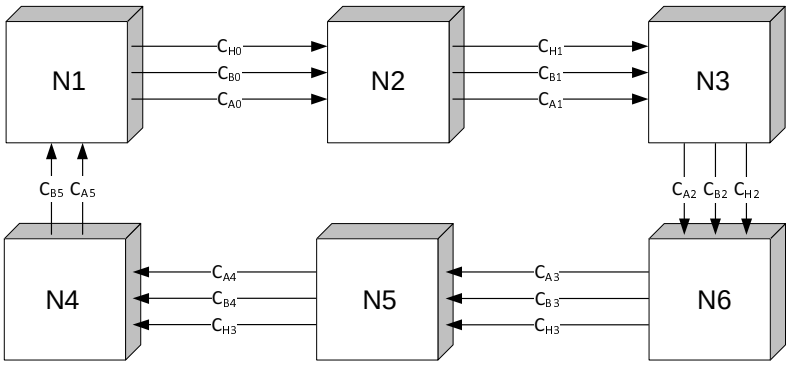



**Figure 1.2**: Layout of a multiprocessor interconnection network with an unidirectional ring topology

tive, L-$S^3$PR nets are very interesting from an analytic point of view.

The mentioned restriction over resources usage is eliminated by the (more general) $S^4$PR class [Tri03, TGVCE05]. This allows processes to simultaneously reserve several resources belonging to distinct types. Nowadays, this is the most general class that allows modeling FMSs while maintaining a complete correction methodology based on efficient structural analysis and synthesis techniques. However, a few more general net classes (SPQR, $S^*$PR) have been proposed in the realm of approaching the RAP in FMSs. This PhD thesis attempts to push the boundaries of those structural techniques into this essentially unexplored ground.

### The RAP in multiprocessor interconnection networks

In recent times, novel results have seen the light in the development of a methodology for the design of deadlock-free minimal adaptive routing algorithms for multicomputer interconnection networks from an RAS perspective [Rov11].

From this point of view, an RAS within the multicomputer interconnection network would be a potential view of it with messages in transit between nodes. Here, the resources are the channels available for communication between nodes, be those physical or virtual. Figure 1.2 shows a network of this type with six nodes, an unidirectional ring interconnection topology, and a variable number of channels between the interconnected nodes. In the particular case of routing in communication networks using wormhole flow control [DS86, Dua95], deadlocks can occur by the emergence of circular waits due to the reservation of channels for simultaneous communication of different messages, each of which is composed of a row of elementary flow units called flits. Such situations are difficult to detect when the routing algorithm is adaptive, i.e., routes are non-preset but dynamic depending on the traffic.

In networks with wormhole flow control, flits travel in a row through the interconnection network by establishing a virtual path. The first of these flits is called head and reserves the channels through which, in turn, the rest of flits will be orderly sent. The last of those flits, called tail, releases the channels reserved by the head, which are then made available for sending flits of other messages.

In this context, a process is the routing of a message. Each process consists of the various states that a message undergoes from source to destination through the interconnection network. Therefore, the state of a process in networks with wormhole flow control is determined by the location of the head and tail flits, as well as by the sequence of channels that are still reserved for the transmission of the message, and that the queue must release. Unlike manufacturing systems, we can see that the state of a process does not have a clear physical location (i.e., a part located on a stage of the production line using some resources) but is somehow scattered in space.

In general terms, the structure of a process is determined by the network topology, the routing algorithm and the flow control model. A particular feature of the networks with wormhole flow control stems from the fact that the reservation and release of channels (resources) by the processes is conducted in a unitary and sequentially ordered way following a policy that might be called 'first allocated, first released'. In the case of static routing, the structure of the processes shall also be linear, which greatly facilitates determining whether the system is deadlock-free. The use of adaptive routing algorithms (i.e., dynamic) marks the possibility of finding alternative paths (decisions) and merges in the structure of the processes. Furthermore, the minimality of the adaptive routing algorithm guarantees the absence of internal cycles in the structure of the processes. Assuming unicast communications, the types of processes would be determined by the various destinations.

Approaching the problem of minimal adaptive routing in multiprocessor interconnection networks from the perspective of RASs is a novel initiative. As such, the corpus of knowledge is essentially condensed throughout a single PhD thesis [Rov11] and external contributions in this area are yet to emerge. That thesis tackles the aforementioned problem in interconnection networks with wormhole flow control providing a particular subclass of $S^4PR$ to model and analyse this kind of systems: the $SOAR^2$ class. The definition of this class takes advantage of the peculiar physical constraints of the domain. Despite this limitations, the groundbreaking synthesis approach developed throughout this thesis is relevant for the study of the RAP in the multithreaded software domain. In essence, there exists a disruption here with the classical synthesis methods based on the addition of monitor places that restrict the system behaviour. This disruption came somewhat forced by the need for a truly, unavoidable 'distributed' control approach that can be obviated in the context of FMS, but which is much convenient, even sometimes indispensable, in the concurrent software domain. We delve into this issue throughout Chap. 3.

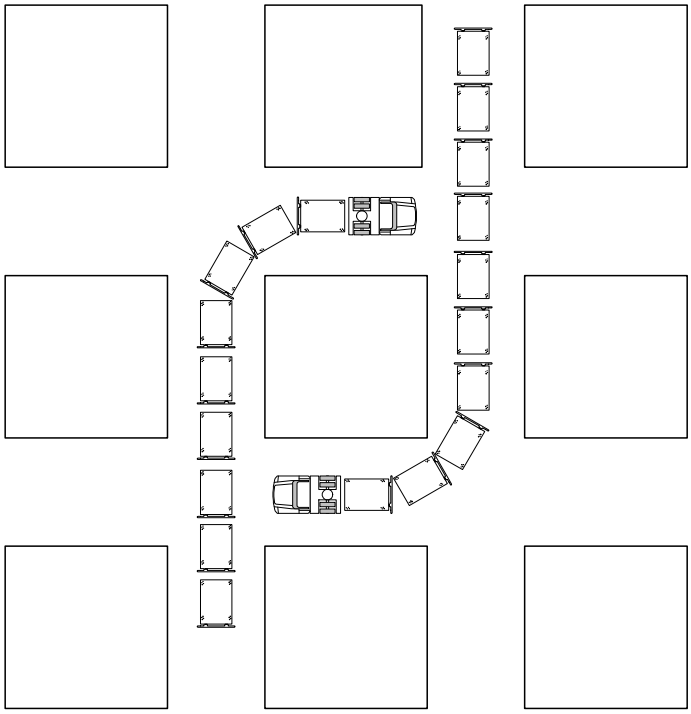


**Figure 1.3**: Example of deadlock in a tow type AGV transportation system

### The RAP in transportation systems

Viewed from an RAS perspective, there are significant parallels between Automated Guided Vehicle (AGV) transportation systems and multicomputer interconnection networks [Rov11]. In particular, in the case of tow type AGVs applied for in-plant material transport, strong resemblances can be traced between deadlocks caused by the blocking of path segments and those emerging in interconnection networks with wormhole flow control. Figure 1.3 illustrates a possible deadlock of this nature.

Those commonplaces have been studied in depth in order to establish a design methodology of deadlock-free minimal adaptive routing algorithms for such systems [Rov11]. In this case, the resources are the segments visited by the AGV. These segments are occupied by the AGV as soon as the tow tractor (the head) enters in them (that includes the decision points that delimit the segment) and are not released until the last of the carts (tail) leaves.

In turn, a process is the transport of goods from a source point (location / station) to a destination point. Following this scheme, it is easy to understand the aforementioned parallel with multicomputer interconnection networks with wormhole

flow control, again being the structure of the processes dependent on the plant topology and the routing algorithm proposed for the AGVs.

The observation previously stated on the synthesis techniques applicable to multiprocessor interconnection networks can be fully extrapolated to the context of the development of deadlock-free minimal adaptive routing algorithms in tow type AGV transportation systems. This parallelism is examined throughout the said PhD thesis, and the results of this work will be exploited and extended in Chap. 3. It should also be mentioned the existence of some other works that address the modeling and analysis from a RAS point of for other types of AGV transportation systems, although the techniques of analysis and correction, when presented, do not always address the problem from a structural standpoint [Rev00, Fan02, WZ05].

#### The RAP in multithreaded software systems

Although there exist a significant number of works that approach the modelling of concurrent programming through Petri nets [IA09], only a handful of them approach multithreaded software from the perspective of RASs [LGC06, WLR+09, Wan09, LGC12], often either with severe limitations in the class of software systems which are subject of study or lacking a complete methodology to correct the liveness problems. This PhD thesis tries to fill this gap in the literature. To this end, the abstraction and modelling of this kind of systems is tackled in Chapter 2. On the other hand, the limitations of the Gadara project, an alternate approach in the same vein [Wan09], are studied in depth in Chapter 4.

## 1.4 The class of $S^4PR$ net systems

In order to illustrate all the ideas previously presented about the different classes of Petri nets that are currently used to model RASs we recall here the definition of $S^4PR$. This class is fundamental because it represents the largest class for which the analysis and synthesis results allow to dispose a complete methodology to construct good RASs. We have syntactic constraints but they are general enough to cover many real applications as those previously presented.

Historically, the $S^4PR$ class appeared as a generalisation of the seminal $S^3PR$ class which was proposed to extend the existing techniques to a more general category of RASs. In this generalisation, the structure of the processes is left intact, and the novelty lies in the fact that several resources can be simultaneously allocated to a single process. Although, $S^4PR$ nets were originally proposed to deal with FMSs, they can be applied to other domains, as discussed in Sect. 1.3.

$S^4PR$ nets are modular models composed of state machines with no internal cycles plus shared resources. One of the most interesting features of this kind of models is

their composability. Two $S^4PR$ nets can be composed into a new $S^4PR$ model via fusion of the common resources. Since multiple resource reservation is allowed, $S^4PR$ nets can be non-ordinary, i.e., the weight of the arcs from the resources to the state machines (or vice versa) is not necessarily equal to one, in contrast to $S^3PR$ nets.

**Definition 1.1.** *[Tri03, TGVCE05] Let $I_{\mathcal{N}}$ be a finite set of indices. An $S^4PR$ net is a connected generalised pure Place/Transition (P/T) net $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ where:*

1. *$P = P_0 \cup P_S \cup P_R$ is a partition such that:*
   - *(a)* [idle places] $P_0 = \bigcup_{i \in I_{\mathcal{N}}} \{p_{0_i}\}$.
   - *(b)* [process places] $P_S = \bigcup_{i \in I_{\mathcal{N}}} P_i$, *where:* $\forall i \in I_{\mathcal{N}} : P_i \neq \emptyset$ *and* $\forall i, j \in I_{\mathcal{N}} : i \neq j,\ P_i \cap P_j = \emptyset$.
   - *(c)* [resource places] $P_R = \{r_1, ..., r_n\}, n > 0$.
2. *$T = \bigcup_{i \in I_{\mathcal{N}}} T_i$, where $\forall i \in I_{\mathcal{N}}, T_i \neq \emptyset$, and $\forall i, j \in I_{\mathcal{N}}, i \neq j, T_i \cap T_j = \emptyset$.*
3. *For each $i \in I_{\mathcal{N}}$ the subnet generated by $\{p_{0_i}\} \cup P_{S_i}$, $T_i$ is a strongly connected state machine such that every cycle contains $p_{0_i}$.*
4. *For each $r \in P_R$ there exists a unique minimal p-semiflow $\mathbf{y_r} \in \mathbb{N}^{|P|}$ such that $\{r\} = \|\mathbf{y_r}\| \cap P_R$, $\|\mathbf{y_r}\| \cap P_0 = \emptyset$, $\|\mathbf{y_r}\| \cap P_S \neq \emptyset$, and $\mathbf{y_r}[r] = 1$.*
5. *$P_S = \bigcup_{r \in P_R} (\|\mathbf{y_r}\| \setminus \{r\})$.*

Fig. 1.4 depicts a net system belonging to the $S^4PR$ class. Places R1 and R2 are the *resource places*. A resource place represents a resource type, and the number of tokens in it represents the quantity of free instances of that resource type. If we remove these places, we get two isolated state machines. These state machines represent the different patterns of resource reservation that a process can follow. In the context of FMSs, these two state machines model two different production plans.

Consequently, tokens in a state machine represent parts which are being processed in stages of the same production plan. At the initial state, the unique tokens in each machine are located at the so-called *idle place* (here: A0, B0). In general, the idle place can be seen as a mechanism which limits the maximum number of concurrent parts being processed in the same production plan. The rest of places model the various stages of the production plan as far as resource reservation is concerned.

Meanwhile, the transitions represent the acquisition or release of resources by the processes along their evolution through the production plan. Every time a transition fires, the total amount of resources available is altered while the part advances to the next stage. The weight of an arc connecting a resource with a transition models the number of instances which are allocated or released when a part advances.

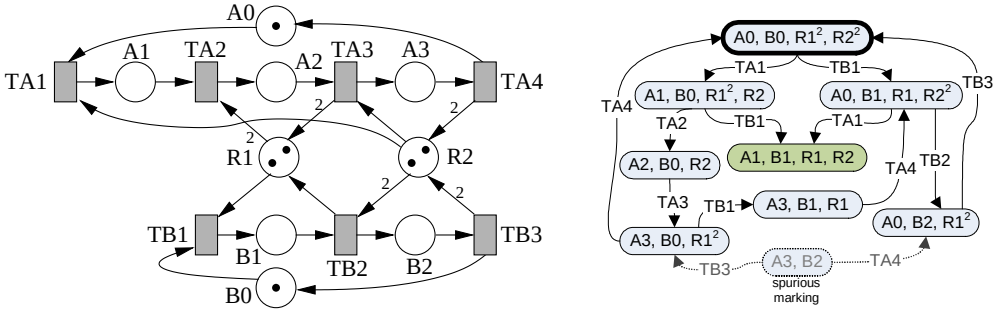



**Figure 1.4**: A $S^4PR$ net system. Despite being non-live, no minimal siphon is ever insufficiently marked

For instance, place R1 could model a set of free robotic arms used to process parts in the stage A2 of the first production plan (two arms are needed per each part processed there) as well as in the stage B1 of the second production plan (only one arm needed per part processed). Consequently, if transition TB1 is fired from the initial marking then one robotic arm will be allocated and one part will visit stage B1. Still, there will remain one robotic arm to be used freely by other processes.

Finally, it is worth noting that moving one isolated token of a state machine (by firing its transitions) until the token reaches back the idle state, leaves the resource places marking unaltered. Thus, the resource usage is conservative.

The next definition formalises the fact that there should exist enough free resource instances in the initial state so that every production plan is realisable:

**Definition 1.2.** *[TGVCE05] Let* $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ *be an* $S^4PR$ *net. An initial marking* $\mathbf{m_0}$ *is acceptable for* $\mathcal{N}$ *iff* $\|\mathbf{m_0}\| = P_0 \cup P_R$ *and* $\forall p \in P_S, r \in P_R : \mathbf{m_0}[r] \geq \mathbf{y_r}[p]$.

Traditionally, empty or insufficiently marked siphons have been a fruitful structural element for characterising non-live RASs. The more general the net class, however, the more complex the siphon-based characterisation is. In the case of $S^3PR$ liveness can be characterised through the existence of siphons that can eventually be emptied. However, this characterisation is sufficient, but not necessary, in general, for $S^4PR$ net systems: this fact is further examined throughout Chapt. 2. Hence, the liveness characterisation, and particularly the notion of empty siphon, had to be generalised. The following theorem presents a liveness characterisation for $S^4PR$ nets. This characterisation is fully behavioural. The structural causes for non-liveness are related to the existence of certain siphons and this will be presented later. An instrumental definition will be introduced first.

**Definition 1.3.** *Given a marking* $\mathbf{m}$ *in an* $S^4PR$ *net system, a transition* $t$ *is said to be:*

- $\mathbf{m}$*-process-enabled (*$\mathbf{m}$*-process-disabled) iff it has (not) a marked input process place, i.e.* $t \in (\|\mathbf{m}\| \cap P_S)^\bullet$ *(i.e.,* $t \notin (\|\mathbf{m}\| \cap P_S)^\bullet$*).*

- $\mathbf{m}$*-resource-enabled (*$\mathbf{m}$*-resource-disabled) iff its input resource places have (not) enough tokens to fire it, i.e.,* $\mathbf{m}[P_R, t] \geq \mathbf{Pre}[P_R, t]$ *(i.e.,* $\mathbf{m}[P_R, t] \ngeq \mathbf{Pre}[P_R, t]$*).*

**Theorem 1.4.** *[TGVCE05] Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be an* $S^4PR$ *net system with an acceptable initial marking.* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *is non-live iff* $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ *such that the set of* $\mathbf{m}$*-process-enabled transitions is non-empty and each one of these transitions is* $\mathbf{m}$*-resource-disabled.*

Theorem 1.4 relates non-liveness to the existence of a marking in which all active processes are blocked. Their output transitions need resources that are not available. These needed resources cannot be generated (released by the corresponding processes) by the system (the transitions are dead) because there exists a set of circular waits between the blocked processes.

This concept of circular waits is captured in the model by the existence of a siphon (in Petri net terms) whose resource places are the places preventing the firing of the process-enabled transitions. The following theorem characterises non-liveness in terms of siphons establishing the bridge between behavior and model structure.

**Theorem 1.5.** *[TGVCE05] Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be an* $S^4PR$ *net system with an acceptable initial marking.* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *is non-live iff* $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ *and a siphon* $D$ *such that:* i) *There exists at least one* $\mathbf{m}$*-process-enabled transition;* ii) *Every* $\mathbf{m}$*-process-enabled transition is* $\mathbf{m}$*-resource-disabled by resource places in* $D$*;* iii) *Process places in* $D$ *are empty at* $\mathbf{m}$*.*

Most analysis and control techniques in the literature are based on the computation of a structural element which characterises deadlocks in many RAS models: the so-called *bad siphon*. A bad siphon (often also called *strict siphon* in the literature [LZ09]) is a siphon which is not the support of a p-semiflow. If bad siphons become (sufficiently) emptied, their output transitions die since the resource places of the siphon cannot regain tokens anymore, thus revealing the *deadly embrace*. Control techniques thus rely on the insertion of monitor places [HZL09] , i.e. controllers in the real system, which limit the leakage of tokens from the bad siphons. Such a siphon is said to be *insufficiently marked at* $\mathbf{m}$, generalising the notion of empty siphon. Such kind of techniques are revisited throughout Chapt. 3.

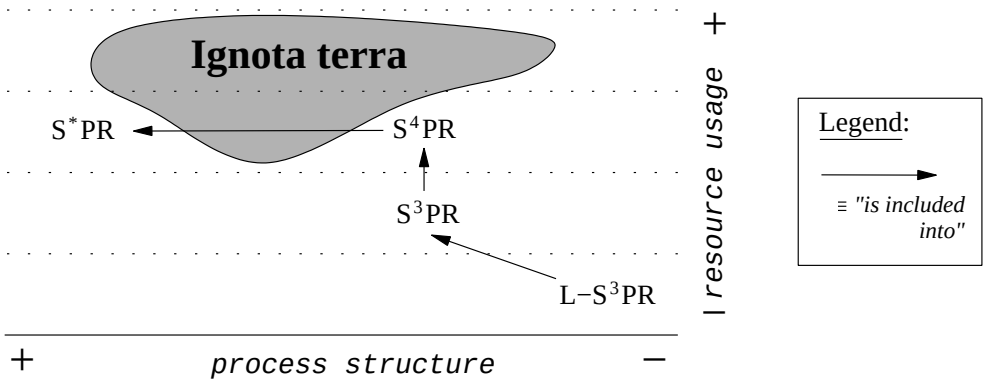



**Figure 1.5**: Inclusion relations between Petri net classes for RASs

## 1.5 Conclusions

In this chapter we have emphasised that an RAS view of a system is just a facet of the system that we consider the first one to be studied and corrected. This, in fact, is a basic principle of our methodological proposal for the construction of complex systems where the existence of concurrency and sharing of resources are two essential characteristics. For the construction of this facet of the system and translate it into a model aimed to study this aspect of the system, an abstraction process of the system is needed. This abstraction process must be able to capture the active entities of the system (processes) and the shared resources for which they are competing. We have proven that this abstraction process is difficult to formalise in a general setting because it depends on the application domains.

After the identification of processes and resources the translation into Petri nets is a more or less easy task. Nevertheless, we have observed that the most synthetic systems that engineers conceive share a significant set of properties. Therefore, the resulting Petri nets belong to a restricted number of subclasses where the variations come from the structure of the processes or the way the resources are used by the processes. Among these subclasses, $S^4PR$ represents an important subclass because it is the most general one for which everything works well. Beyond $S^4PR$, there are many difficulties for the analysis and the synthesis of well-behaved systems that this PhD thesis will point out in the following chapters.

# Chapter 2

# The resource allocation problem in software applications

## Summary

RASs were intensively studied in the last years for FMSs. The success of this research stems from the identification of subclasses of Petri nets that correspond to an RAS abstraction of these systems. In this chapter, a parallel road to that travelled through for FMSs is taken. It is revealed that the existing subclasses of Petri nets used to study this kind of deadlock problems are insufficient, even for very simple software systems. A new subclass of Petri nets that generalises the previously known RAS subclasses is proposed, but extending the kind of systems that can be tackled. These extensions allow to consider nested iterations within the processes, and to hold resources in the initial state. We will show that these extensions are very relevant from the real-world system point of view. Nevertheless, the behaviour of the resulting models is much more complex than those of the previous restricted models. A taxonomy of emerging anomalies is therefore presented in the context of software systems. These harden the extension of the existing liveness analysis and synthesis techniques from FMSs into the new scenario. Finally, an interesting subclass of nets named SPQR is presented, which was originally aimed to generalise the class of FMSs tackled with previous approaches. Due to its syntactic simplicity and the existence of powerful net transformation rules which retain the essential net properties, this subclass seems to facilitate the study of the liveness problem at its very core. Therefore, its basic properties will be studied and relations between both classes will be manifested.

## 2.1 Introduction

In Chap. 1, the fundamental concepts on the RAP are introduced, and a whole family of Petri net models to represent RASs which are commonplace in the literature are categorised. Such Petri net classes are frequently presented in the context of FMS modelling, and make sense as artifacts conceived for properly modelling significant physical aspects of this kind of systems. On the other hand, there exists a family of powerful results which allow applying liveness enforcement techniques over such FMS-oriented models, and ultimately, over the real system.

Although there exist obvious resemblances between the RAP in FMSs and that of parallel or concurrent software, previous attempts to bring these well-known RAS techniques into the field of software engineering have been, to the best of our knowledge, either too limiting or unsuccessful. Gadara nets [WLR+09] constitute the most recent attempt, yet they fall in the over-restrictive side in the way the resources can be used. Presumably, this is a consequence of being conceived with a primary focus on inheriting the powerful structural liveness results which were fruitful in the context of FMSs. Such a bias works against obtaining a model class capable of properly abstracting RASs in many multithreaded systems, as later discussed in Chap. 4. In this chapter, it is basically analysed why the net classes and results introduced in the context of FMSs can fail when brought to the field of concurrent programming.

Section 2.2 presents a motivating example and discusses the elements that an RAS net model should desirably feature in order to successfully explore the RAP within the software engineering discipline. Taking into account those considerations, Sect. 2.3 introduces a new Petri net class, called $\mathrm{PC}^2\mathrm{R}$. Section 2.4 relates the new class to those defined in previous works and forewarn us about new behavioural phenomena. Some of these anomalies highlight the fact that previous theoretical results in the context of FMSs are insufficient in the new framework. Finally, in Sect. 2.5 another Petri net class is introduced named System of Processes Quarrelling over Resources (SPQR). This is essentially a subclass of $\mathrm{PC}^2\mathrm{R}$ which facilitates the study of the liveness problem in multithreaded software systems. Some properties, relations and net transformations will be introduced to justify its definition.

Note that some additional figures are provided in Appendix B which complement some of the examples throughout the chapter with further information.

## 2.2 The RAS view of a software application

Example 2.1 presents a humorous variation of Dijkstra's classic problem of the dining philosophers which adopts the beautiful writing by C.A.R. Hoare [Hoa78].

**Example 2.1** *The postmodern dining philosophers.* "Five philosophers spend their lives thinking and eating. The philosophers share a common dining room where there

is a circular table surrounded by five chairs, each belonging to one philosopher. A microwave oven is also available. In the centre of the table there is a large bowl of spaghetti which is frequently refilled (so it cannot be emptied), and the table is laid with five forks. On feeling hungry, a philosopher enters the dining room, sits in his own chair, and picks up the fork on the left of his place. Then he touches the bowl to feel its temperature. If he feels the spaghetti got too cold, he leaves his fork and takes the bowl to the microwave. Once it is warm enough, he comes back to the table, sits on his chair and leaves the bowl on the table after recovering his left fork. Unfortunately, the spaghetti is so tangled that he needs to pick up and use the fork on his right as well. If he can do this before the bowl gets cold again, he serves himself and starts eating. When he has finished, he puts down both forks and leaves the room."

According to the classic RAS nomenclature, each philosopher is a sequential process, and the five forks plus the bowl are serially reusable resources which are shared among the five processes. From a software perspective, each philosopher can be a process or a thread executed concurrently.

Algorithm 2.1 introduces the code for each philosopher. Notationally, the acquisition / release of resources is modelled by way of the `wait()` / `signal()` atomic operations, respectively. Both of them have been generalised for the acquisition of multiple resources (separated by commas, inside the parentheses, when invoking the operation). Finally, `trywait()` models a non-blocking wait operation. If every resource is available at the time `trywait()` is invoked, then it acquires them and returns `TRUE`. Otherwise, `trywait()` will return `FALSE` without acquiring any resource. For the sake of simplicity, it is assumed that the conditions with two or more literals are also evaluated atomically.

Figure 2.1 depicts the net for Algorithm 2.1, with $i = 1$, after abstracting the relevant information from an RAS perspective. Figure 2.2 renders the composition of the five philosopher nets via fusion of the common shared resources. Note that if the dashed arcs from Fig. 2.2 are removed, then we can see five disjoint strongly connected state machines plus six isolated places.

Each state machine represents the control flow for a philosopher. Every state machine is composed of seven states (places). Tokens in a state machine represent concurrent processes/threads which share the same control flow. At the initial state, every philosopher is thinking (outside the room), i.e. the unique token in each machine is located at the so-called *idle place*. In general, the idle place can be seen as a mechanism which limits the number of concurrent *active threads*. Here, at most one philosopher of type $i$ can be inside the room, for each $i \in \{1, ..., 5\}$.

The six isolated places are called *resource places*. A resource place represents a resource type, and the number of tokens in it represents the quantity of free instances of that resource type. In this case, every resource place is monomarked. Thus, at

**Algorithm 2.1** - Code for Philosopher i (where $i \in \{1, 2, 3, 4, 5\}$)

```
var
      fork:  array [1..5] of semaphores; // shared resources
      bowl:  semaphore;                  // shared resource
begin
   do while (1)
      THINK;
      Enter the room;
(T1)  wait(fork[i]);                (T6)
      do while (not(trywait(bowl, fork[i mod 5 +1]))
                or the spaghetti is cold)
(T2)     if (trywait(bowl)
                and the spaghetti is cold) then
(T3)        signal(fork[i]);
            Go to the microwave;
            Heat up spaghetti;
            Go back to table;
(T4)        wait(fork[i]);
(T5)        signal(bowl);
         end if;
      loop;
      Serve spaghetti;
(T7)  signal(bowl);
      EAT;
(T8)  signal(fork[i], fork[i mod 5 +1]);
      Leave the room;
   loop;
```

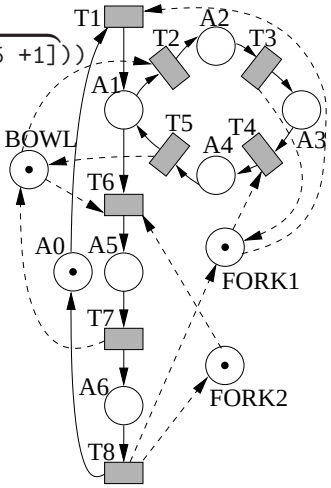



**Figure 2.1**: Philosopher 1

the initial state there is one fork of type $i$, for every $i \in \{1, ..., 5\}$, plus one bowl of spaghetti (modelled by the resource place at the centre of the figure).

Finally, the dashed arcs represent the acquisition or release of resources by the active threads when they change their execution state. Every time a transition fires, the total amount of resources available is altered. Please note, however, that moving one isolated token of a state machine (by firing its transitions) until the token reaches back the idle state, leaves the resource place markings unaltered. Thus, the resource usage is conservative.

At this point, some capabilities that an RAS model should have so as to support the modelling of concurrent programs are discussed.

State machines without internal cycles are rather versatile for modelling sequential processes in the context of FMSs. In FMSs, the process plans containing internal loops

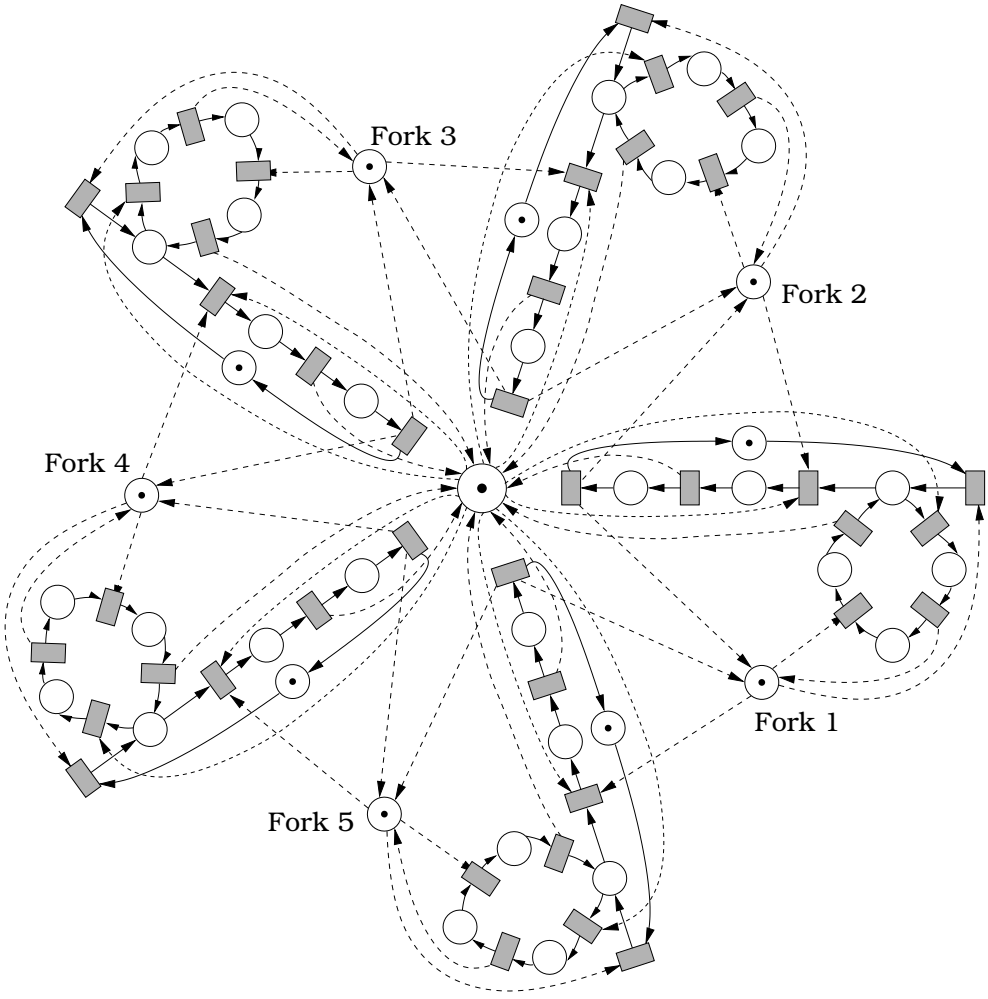



**Figure 2.2**: The dining philosophers are thinking. Arcs from/to $P_{\mathrm{R}}$ are dashed for clarity

exist. Nevertheless, the number of iterations is usually constant and independent of the raw parts to be processed. Consequently, from the point of view of modelling, loops are unrolled and models without internal loops are generated. The success of the $\mathrm{S}^3\mathrm{PR}$ and $\mathrm{S}^4\mathrm{PR}$ classes, in which every circuit in the state machines traverses the idle place, proves the claimed versatility of the approach.

Nevertheless, this is clearly too constraining even for very simple software systems. Considering Böhm and Jacopini's theorem [Har80], however, it can be assumed that every non-structured sequential program can be refactored into a structured one using `while-do` loops. In this case, many loops cannot be unrolled, because the number

of iterations depends on the value of some data to be processed during the process runtime.

Meanwhile, calls to procedures and functions can be substituted by inlining techniques. Let us also remind that `fork/join` operations can also be unfolded into isolated concurrent sequential processes, as some works evidence [ER04]. As a result, process models can be restricted to state machines in which decisions and iterations (in the form of `while-do` loops) are supported, but not necessarily every kind of unstructured branch.

Another significant difference between FMSs and software systems from an RAS perspective is that resources in the latter are not necessarily physical (e.g., a file) but can also be logical (e.g., a semaphore). This has strong implications in the degree of freedom in allocating those resources: this issue will be further inspected later.

In this domain, a resource is an object that is shared among concurrent processes/threads and must be used in mutual exclusion. Since the number of resources is limited, the processes compete for the resource and use it in a non-preemptive way. This particular allocation scheme can be imposed by the resource access primitives, which may be blocking. Otherwise, the resource can be protected by a binary semaphore/mutex/lock (if there is only one instance of that resource type) or by a counting semaphore (multiple instances). Note that this kind of resources can be of varied nature (e.g., shared memory locations, storage space, database table rows) but the required synchronisation scheme is inherently similar.

On the other hand, it is well-known that semaphores used in that context can also be seen as non-preemptive resources which are used in a conservative way. For example, a counting semaphore that limits the number of connections to a database can be interpreted in that way from an RAS point of view. Here processes wait for the semaphore when attempting to establish a database connection, and release it when they decide to close the aforementioned connection.

However, semaphores also perform a relevant role as an interprocess signalling facility, which can also be a source of deadlocks. In this chapter, the goal is the study of the RAP, so this mode of use is out of scope. Instead, it is proposed to fix deadlock problems due to resource allocation issues firstly, and later apply other techniques for amending those due to message passing.

Due to their versatility, semaphore primitives are interesting for studying how resources can be allocated by a process/thread. For instance, XSI semaphores (also known as System V semaphores) have a multiple wait primitive (`semop` with `sem_op<0`). An example of multiple resource allocation appears in Algorithm 2.1. Besides, an XSI semaphore can be decremented atomically in more than one unit. Both POSIX semaphores (through `sem_trywait`) and XSI semaphores (through `semop` with `sem_op<0` and `sem_flag=IPC_NOWAIT`) have a non-blocking wait primitive. Again, Algorithm 2.1 could serve as an example. Finally, XSI semaphores also feature inhibition

mechanisms (through `semop` with `sem_op=0`), i.e. processes can wait for a zero value of the semaphore.

All these implementations of the semaphore API use a waiting queue of pending processes for completing `wait` operations on 0-valued semaphores. In this thesis, this queue is ignored in order to consider the inherent non-determinism of the operating system scheduler when imposing an ordering to the processes attempting to enter in the queue of the semaphore.

As suggested earlier, the fact that resources in software engineering do not always have a physical counterpart is a peculiar characteristic with consequences. In this context, processes do not only consume resources but also can *create* them. A process will destroy the newly created resources before its termination. For instance, a process can create a shared memory variable (or a service!) which can be allocated to other processes/threads. Hence the resource allocation scheme is no longer *first-acquire-later-release*, but it can be the other way round too. Still, all the resources are used conservatively by the processes (either by a create-destroy sequence or by a wait-release sequence). As a side effect, and perhaps counter-intuitively, there may not be free resources during the system startup (as they still must be created), yet the system is live.

Nevertheless, the kind of software systems to be considered in this study will be limited. The following typical features that can appear in a program will not be considered for the moment:

- Dynamic creation of new types of functionality implemented throughout threads based on functions which were not present in the initial state.
- Function recursion.
- Reentrant routines outside the caller's address space. Coroutines.
- Software or hardware interrupts.
- Signalling facilities.

Besides, internal choices of the processes determined by the value of a data-dependent expression (i.e., not dependent on semaphores) will be represented as non-deterministic choices (i.e., free choices).

Summing up, for successfully modelling RASs in the context of software engineering, a Petri net model should at least fulfil the following requirements:

1. The control flow of the processes should be represented by state machines with support for decisions (`if-then-else` blocks) and nested internal cycles (`while-do` blocks).
2. There can be several resource types and multiple instances of each one.

3. State machines can have multiple tokens (representing concurrent threads) executing the same function.

4. Processes/threads use resources in a conservative way.

5. Acquisition/release arcs can have non-ordinary weights (e.g., a semaphore value can be atomically incremented/decremented in more than one unit).

6. Atomic multiple acquisition/release operations must be allowed.

7. Processes can have decisions dependent of the allocation state of resources (due to the non-blocking wait primitives, as in Fig. 2.2).

8. Processes can lend resources. As a side effect, there could exist processes that depend on resources which must be created/lent by other processes

With these characteristics in mind, the next section defines a class of Petri nets to cope with these requirements.

## 2.3 The PC$^2$R class

### 2.3.1 Functional entities. Representation

In this section, a new Petri net class is presented: the class of Processes Competing for Conservative Resources (PC$^2$R). This class fulfils the list of abstract requirements enunciated in Sect. 2.2 and generalises other subclasses of the S$^n$PR family while respecting the design philosophy on these. Hence, some previous results are still valid in the new framework. However, PC$^2$R nets can deal with more complex scenarios which were not yet addressed from the domain of S$^n$PR nets.

It should be remarked that no S$^n$PR net class fulfils all requirements presented at the end of Sect. 2.2. Requirements 2–6 are satisfied by the S$^4$PR class. Requirement 7 also is, but it is not fulfilled by Gadara nets. Requirement 1 is only verified by the Gadara and S$^*$PR classes, and fulfilment of Requirement 8 has never been addressed before, justifying the next definition.

Definition 2.2 presents a subclass of state machines used for modelling the control flow of the processes in isolation. Iterations are allowed, as well as decisions within internal cycles, in such a way that the control flow of structured programs can be fully supported, in fulfilment of Requirement 1.

**Definition 2.2.** *An* iterative state machine $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ *is a strongly connected state machine such that:*

1. *$P$ can be partitioned into three subsets: $\{p_k\}$, $P_1$ and $P_2$.*

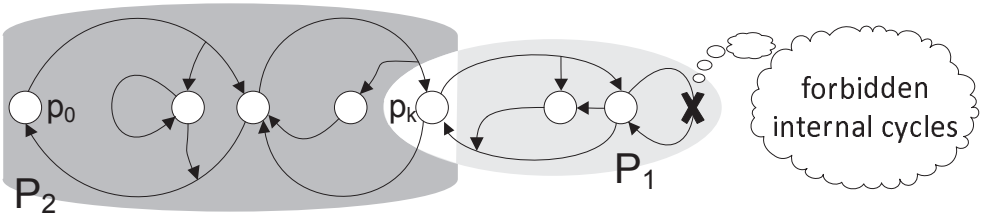



**Figure 2.3**: Schematic diagram of an iterative state machine

2. $P_1 \neq \emptyset$,

3. *The subnet generated by* $\{p_k\} \cup P_1, {}^\bullet P_1 \cup P_1{}^\bullet$ *is a strongly connected state machine in which every cycle contains* $p_k$,

4. *If* $P_2 \neq \emptyset$, *the subnet generated by* $\{p_k\} \cup P_2, {}^\bullet P_2 \cup P_2{}^\bullet$ *is an iterative state machine.*

As Fig. 2.3 shows, $P_1$ contains the set of places of an outermost[1] iteration block, while $P_2$ is the set of places of the rest of the state machine (the inner structure, which may contain multiple loops within). Consequently, the subnet generated by $\{p_\mathrm{k}\} \cup P_1, {}^\bullet P_1 \cup P_1{}^\bullet$ is a strongly connected state machine in which every cycle contains $p_\mathrm{k}$. Meanwhile, inner iteration blocks can be identified in the iterative state machine generated by $\{p_\mathrm{k}\} \cup P_2, {}^\bullet P_2 \cup P_2{}^\bullet$.

Finally, the place $p_0$ represents the place "$p_\mathrm{k}$" that we choose after removing every iteration block. In the following, such possible *last* $p_\mathrm{k}$ places will be generically called *idle places*. Note that by the definition of iterative state machine, at least one idle place must exist. Such term will serve as a bridge prior to the introduction of the concept of idle place of a process subnet in the context of the class of PC$^2$R nets. At this point, it must be remarked that the definition of iterative state machines is instrumental for introducing the class of PC$^2$R nets.

In the net of Fig. 2.1, if we remove the resource places FORK1, FORK2 and BOWL, we obtain an iterative state machine, with $P_1 = \{\mathrm{A2}, \mathrm{A3}, \mathrm{A4}\}$, $P_2 = \{\mathrm{A0}, \mathrm{A5}, \mathrm{A6}\}$ and $p_\mathrm{k} = \mathrm{A1}$. Note that, according to the previous description, A0 can be considered the idle place.

In the following, we assume that every state machine has one fixed idle place. Obviously, this idle place can be any place of the iterative state machine, but it is

[1] In the context of this thesis, we always use the adjectives *outer*/*inner*/etc. from the perspective of the structure of the iterative state machine: where the outermost iteration blocks are those farthest from the idle place. Please note that in the context of imperative programming languages is exactly the other way round: the innermost loops in the program structure are those which are modelled with the outermost iteration blocks in the iterative state machine.

assumed being identified from the beginning. Considering the application domain (multithreaded software engineering) this makes perfect sense, since an iterative state machine models the possible execution paths of a thread. In this sense, it must be remarked that the point of entry (thread startup) can be unequivocally identified by the `begin` statement.

An elementary iteration block of an iterative state machine is a maximal strongly connected state machine embedded in the iterative state machine where all circuits contain a distinguished idle place named the idle place of the elementary iteration block. Algorithm 2.2 computes the set of these elementary iteration blocks and, in the answer, each elementary block is characterised by the idle place, $p_{\mathrm{k}}$, the set of places, $P_{\mathrm{SM}}$, and the set of transitions, $T_{\mathrm{SM}}$. Obviously, the flow relation between places and transitions of an elementary iteration block is the flow relation of the iterative state machine constrained to the objects of the elementary iteration block.

**Definition 2.3.** *Let $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$ be an iterative state machine, and $p_0 \in P$ be its idle place. The set of* elementary iteration blocks *of $\mathcal{N}$, $\mathcal{S}_{\mathcal{N}}$, is the set generated by Algorithm 2.2.*

Note that the computation of a minimal circuit which covers a certain place $p$ is a polynomial complexity problem that can be accomplished, for example, through the computation of a basic feasible solution via the SIMPLEX algorithm on the set of constraints $\mathbf{C} \cdot \mathbf{x} = \mathbf{0}$, $\mathbf{x}[p^\bullet] \neq \mathbf{0}$, $\mathbf{x} \geq \mathbf{0}$ (basic feasible solutions are of minimal support and are therefore minimal t-semiflows of the net, i.e., elementary circuits). An analogous approach can be used to compute a minimal circuit containing a given transition. In the same vein, D. B. Johnson [Joh75] proposes an algorithm for the computation of minimal circuits of a digraph which starts by computing a first elementary circuit in polynomial time. Therefore Algorithm 2.2 exhibits a polynomial run-time growth rate on the size of the net in the worst-case scenario.

Figure 2.4 depicts the elementary iteration blocks obtained for a sample iterative state machine using Algorithm 2.2, assuming that P0 is the idle place of the iterative state machine. In order to illustrate how it works, a trace of the first iterations of an execution of the algorithm using the net in Fig. 2.4 as input follows:

*idle_pairs* $= \{(\mathrm{P0}, [\mathrm{P0\ T1\ P1\ T3\ P3\ T6\ P5\ T8}])\}$;
**Iteration 1** (idle place P0)
$p_k = \mathrm{P0}$; $\pi = [\mathrm{P0\ T1\ P1\ T3\ P3\ T6\ P5\ T8}]$;
$\rho = \{\mathrm{P0}, \mathrm{P1}, \mathrm{P3}, \mathrm{P5}\}$; $P_{\mathrm{SM}} = \emptyset$; $T_{\mathrm{SM}} = \{\mathrm{T1}, \mathrm{T3}, \mathrm{T6}, \mathrm{T8}\}$;
*idle_pairs* $= \emptyset$;
**Iteration 1.1** (place $\mathrm{P0} \in \rho$)
$\rho = \{\mathrm{P1}, \mathrm{P3}, \mathrm{P5}\}$; $p_1 = \mathrm{P0}$;
$\tau = \mathrm{P0}^\bullet \setminus T_{\mathrm{SM}} = \{\mathrm{T1}, \mathrm{T2}\} \setminus T_{\mathrm{SM}} = \{\mathrm{T2}\}$;

**Algorithm 2.2** Splitting of an iterative state machine into its set of elementary iteration blocks

**Input:** An iterative state machine $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ and the idle place $p_0 \in P$ of $\mathcal{N}$.

**Output:** A set of strongly connected state machines of $\mathcal{N}$, *iteration_blocks*, each one with a distinguished idle place, and all circuits containing the detected idle place.

**Note 1:** The function Min_Circuit($\mathcal{N}, x$), $x \in P \cup T$, returns a minimal circuit in $\mathcal{N}$ traversing $x$.

**Note 2:** Given a circuit $\pi$, the functions $\mathrm{f_P}(\pi)$ and $\mathrm{f_T}(\pi)$ return the set of places and transitions, respectively, that occur at least once in $\pi$.

**begin**

1: *iteration_blocks* $\leftarrow \emptyset$; *visited_trans* $\leftarrow \emptyset$; *idle_pairs* $\leftarrow \{(p_0, \text{Min\_Circuit}(\mathcal{N}, p_0))\}$;
2: **while** *idle_pairs* $\neq \emptyset$ **do**
3: $(p_\mathrm{k}, \pi) \leftarrow$ Extract_A_Pair(*idle_pairs*);
4: $\rho \leftarrow \mathrm{f_P}(\pi)$; $P_\mathrm{SM} \leftarrow \emptyset$; $T_\mathrm{SM} \leftarrow \mathrm{f_T}(\pi)$;
5: **while** $\rho \neq \emptyset$ **do**
6: $p_1 \leftarrow$ Extract_An_Element($\rho$);
7: $\tau \leftarrow p_1^\bullet \setminus (T_\mathrm{SM} \cup$ *visited_trans*$)$; $P_\mathrm{SM} \leftarrow P_\mathrm{SM} \cup \{p_1\}$;
8: **while** $\tau \neq \emptyset$ **do**
9: $t \leftarrow$ Extract_An_Element($\tau$);
10: $\pi' \leftarrow$ Min_Circuit($\mathcal{N}, t$);
11: **if** $p_k \notin f_\mathrm{P}(\pi')$ **then**
12: *idle_pairs* $\leftarrow$ *idle_pairs* $\cup \{(p_1, \pi')\}$;
13: **else**
14: $\rho \leftarrow (\rho \cup f_\mathrm{P}(\pi')) \setminus P_\mathrm{SM}$; $T_\mathrm{SM} \leftarrow T_\mathrm{SM} \cup f_\mathrm{T}(\pi')$;
15: **end if**
16: **end while**
17: **end while**
18: *iteration_blocks* $\leftarrow$ *iteration_blocks* $\cup \{(p_\mathrm{k}, P_\mathrm{SM}, T_\mathrm{SM})\}$;
19: *visited_trans* $\leftarrow$ *visited_trans* $\cup\, T_\mathrm{SM}$;
20: **end while**

$P_\mathrm{SM} = P_\mathrm{SM} \cup \{\mathrm{P0}\} = \{\mathrm{P0}\}$;
**Iteration 1.1.1** (transition T2 $\in \tau$)
$t = \mathrm{T2}$; $\tau = \emptyset$; $\pi' = [\mathrm{P0\ T2\ P2\ T5\ P4\ T7\ P5\ T8}]$;
$p_\mathrm{k} \in f_P(\pi') \implies$
$\rho = \{\mathrm{P1}, \mathrm{P2}, \mathrm{P3}, \mathrm{P4}, \mathrm{P5}\}$;
$T_\mathrm{SM} = \{\mathrm{T1}, \mathrm{T2}, \mathrm{T3}, \mathrm{T5}, \mathrm{T6}, \mathrm{T7}, \mathrm{T8}\}$;
**Iteration 1.2** (place P1 $\in \rho$)
$\rho = \{\mathrm{P2}, \mathrm{P3}, \mathrm{P4}, \mathrm{P5}\}$; $p_1 = \mathrm{P1}$;

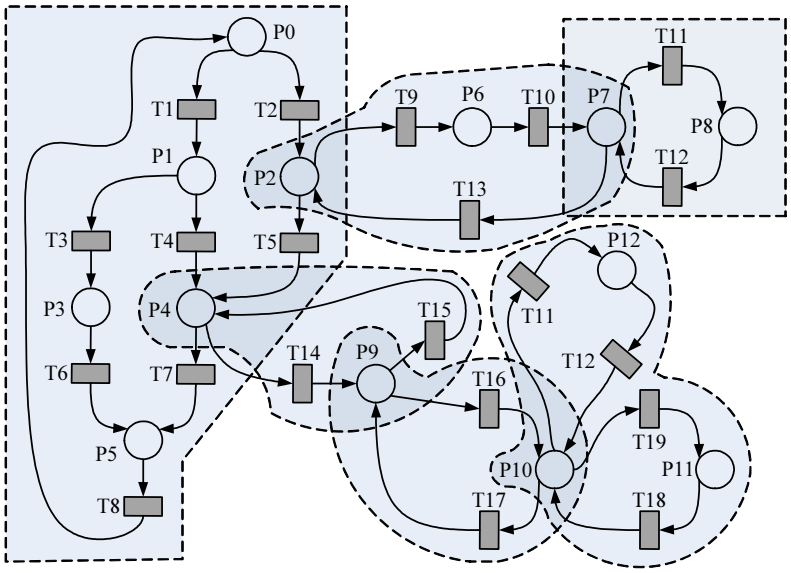



**Figure 2.4**: Elementary iteration blocks of an iterative state machine

$\tau = \mathrm{P1}^\bullet \setminus T_{\mathrm{SM}} = \{\mathrm{T3}, \mathrm{T4}\} \setminus T_{\mathrm{SM}} = \{\mathrm{T4}\}$;
$P_{\mathrm{SM}} = \{\mathrm{P0}, \mathrm{P1}\}$;
**Iteration 1.2.1** (transition $\mathrm{T4} \in \tau$)
$t = \mathrm{T4}$; $\tau = \emptyset$; $\pi' = [\mathrm{P0\ T1\ P1\ T4\ P4\ T7\ P5\ T8}]$;
$p_{\mathrm{k}} \in f_P(\pi') \implies$
$\rho = \{\mathrm{P2}, \mathrm{P3}, \mathrm{P4}, \mathrm{P5}\}$;
$T_{\mathrm{SM}} = \{\mathrm{T1}, \mathrm{T2}, \mathrm{T3}, \mathrm{T4}, \mathrm{T5}, \mathrm{T6}, \mathrm{T7}, \mathrm{T8}\}$;
**Iteration 1.3** (place $\mathrm{P2} \in \rho$)
$\rho = \{\mathrm{P3}, \mathrm{P4}, \mathrm{P5}\}$; $p_1 = \mathrm{P2}$;
$\tau = \mathrm{P2}^\bullet \setminus T_{\mathrm{SM}} = \{\mathrm{T5}, \mathrm{T9}\} \setminus T_{\mathrm{SM}} = \{\mathrm{T9}\}$;
$P_{\mathrm{SM}} = \{\mathrm{P0}, \mathrm{P1}, \mathrm{P2}\}$;
**Iteration 1.3.1** (transition $\mathrm{T9} \in \tau$)
$t = \mathrm{T9}$; $\tau = \emptyset$; $\pi' = [\mathrm{P2\ T9\ P6\ T10\ P7\ T13}]$;
$p_{\mathrm{k}} \notin f_P(\pi') \implies$
$\mathit{idle\_pairs} = \emptyset \cup \{(\mathrm{P2}, [\mathrm{P2\ T9\ P6\ T10\ P7\ T13}])\}$;
**Iteration 1.4** (place $\mathrm{P3} \in \rho$)
$\rho = \{\mathrm{P4}, \mathrm{P5}\}$; $p_1 = \mathrm{P3}$;
$\tau = \mathrm{P3}^\bullet \setminus T_{\mathrm{SM}} = \{\mathrm{T6}\} \setminus T_{\mathrm{SM}} = \emptyset$;
$P_{\mathrm{SM}} = \{\mathrm{P0}, \mathrm{P1}, \mathrm{P2}, \mathrm{P3}\}$;
**Iteration 1.5** (place $\mathrm{P4} \in \rho$)
$\rho = \{\mathrm{P5}\}$; $p_1 = \mathrm{P4}$;

$\tau = \mathrm{P4}^\bullet \setminus T_{\mathrm{SM}} = \{\mathrm{T7}, \mathrm{T14}\} \setminus T_{\mathrm{SM}} = \{\mathrm{T14}\}$;
$P_{\mathrm{SM}} = \{\mathrm{P0}, \mathrm{P1}, \mathrm{P2}, \mathrm{P3}, \mathrm{P4}\}$;
**Iteration 1.5.1** (transition $\mathrm{T14} \in \tau$)
$t = \mathrm{T14}$; $\tau = \emptyset$; $\pi' = [\mathrm{P4\ T14\ P9\ T15}]$;
$p_{\mathrm{k}} \notin f_P(\pi') \implies$
*idle_pairs* $= \{(\mathrm{P2}, [\mathrm{P2\ T9\ P6\ T10\ P7\ T13}]),$
$(\mathrm{P4}, [\mathrm{P4\ T14\ P9\ T15}])\}$;
**Iteration 1.6** (place $\mathrm{P5} \in \rho$)
$\rho = \emptyset$; $p_1 = \mathrm{P5}$;
$\tau = \mathrm{P5}^\bullet \setminus T_{\mathrm{SM}} = \{\mathrm{T8}\} \setminus T_{\mathrm{SM}} = \emptyset$;
$P_{\mathrm{SM}} = \{\mathrm{P0}, \mathrm{P1}, \mathrm{P2}, \mathrm{P3}, \mathrm{P4}, \mathrm{P5}\}$;
*iteration_blocks* $= \{(\mathrm{P0}, \{\mathrm{P0}, \mathrm{P1}, \mathrm{P2}, \mathrm{P3}, \mathrm{P4}, \mathrm{P5}\},$
$\{\mathrm{T1}, \mathrm{T2}, \mathrm{T3}, \mathrm{T4}, \mathrm{T5}, \mathrm{T6}, \mathrm{T7}, \mathrm{T8}\})\}$;
*visited_trans* $= \{\mathrm{T1}, \mathrm{T2}, \mathrm{T3}, \mathrm{T4}, \mathrm{T5}, \mathrm{T6}, \mathrm{T7}, \mathrm{T8}\}$;
**Iteration 2** (idle place P2)
$p_k = \mathrm{P2}$; $\pi = [\mathrm{P2\ T9\ P6\ T10\ P7\ T13}]$;
$\rho = \{\mathrm{P2}, \mathrm{P6}, \mathrm{P7}\}$; $P_{\mathrm{SM}} = \emptyset$; $T_{\mathrm{SM}} = \{\mathrm{T9}, \mathrm{T10}, \mathrm{T13}\}$;
*idle_pairs* $= \{(\mathrm{P4}, [\mathrm{P4\ T14\ P9\ T15}])\}$;
**Iteration 2.1** (place $\mathrm{P2} \in \rho$)
$\rho = \{\mathrm{P6}, \mathrm{P7}\}$; $p_1 = \mathrm{P2}$;
$\tau = \mathrm{P2}^\bullet \setminus (T_{\mathrm{SM}} \cup \mathit{visited\_trans}) =$
$\{\mathrm{T5}, \mathrm{T9}\} \setminus (\{\mathrm{T9}, \mathrm{T10}, \mathrm{T13}\} \cup \{\mathrm{T1} - \mathrm{T8}\}) = \emptyset$;
$P_{\mathrm{SM}} = \{\mathrm{P2}\}$;
**Iteration 2.2** (place $\mathrm{P6} \in \rho$)
$\rho = \{\mathrm{P7}\}$; $p_1 = \mathrm{P6}$;
$\tau = \mathrm{P6}^\bullet \setminus (T_{\mathrm{SM}} \cup \mathit{visited\_trans}) =$
$\{\mathrm{T10}\} \setminus (\{\mathrm{T9}, \mathrm{T10}, \mathrm{T13}\} \cup \{\mathrm{T1} - \mathrm{T8}\}) = \emptyset$;
$P_{\mathrm{SM}} = \{\mathrm{P2}, \mathrm{P6}\}$;
**Iteration 2.3** (place $\mathrm{P7} \in \rho$)
$\rho = \emptyset$; $p_1 = \mathrm{P7}$;
$\tau = \mathrm{P7}^\bullet \setminus (T_{\mathrm{SM}} \cup \mathit{visited\_trans}) =$
$\{\mathrm{T11}, \mathrm{T13}\} \setminus (\{\mathrm{T9}, \mathrm{T10}, \mathrm{T13}\} \cup \{\mathrm{T1} - \mathrm{T8}\}) = \{\mathrm{T11}\}$;
$P_{\mathrm{SM}} = \{\mathrm{P2}, \mathrm{P6}, \mathrm{P7}\}$;
**Iteration 2.3.1** (transition $\mathrm{T11} \in \tau$)
$t = \mathrm{T11}$; $\tau = \emptyset$; $\pi' = [\mathrm{P7\ T11\ P8\ T12}]$;
$p_{\mathrm{k}} \notin f_P(\pi') \implies$
*idle_pairs* $= \{(\mathrm{P4}, [\mathrm{P4\ T14\ P9\ T15}]),$
$(\mathrm{P7}, [\mathrm{P7\ T11\ P8\ T12}])\}$;
*iteration_blocks* $= \{(\mathrm{P0}, \{\mathrm{P0}, \mathrm{P1}, \mathrm{P2}, \mathrm{P3}, \mathrm{P4}, \mathrm{P5}\},$
$\{\mathrm{T1}, \mathrm{T2}, \mathrm{T3}, \mathrm{T4}, \mathrm{T5}, \mathrm{T6}, \mathrm{T7}, \mathrm{T8}\}),$

$(\mathrm{P2}, \{\mathrm{P2}, \mathrm{P6}, \mathrm{P7}\}, \{\mathrm{T9}, \mathrm{T10}, \mathrm{T13}\})\}$;

*visited_trans* $= \{\mathrm{T1}, \mathrm{T2}, \mathrm{T3}, \mathrm{T4}, \mathrm{T5}, \mathrm{T6}, \mathrm{T7}, \mathrm{T8}, \mathrm{T9}, \mathrm{T10}, \mathrm{T13}\}$;

**Iteration 3** (idle place P4)

$p_k = \mathrm{P4}$; $\pi = [\mathrm{P4\ T14\ P9\ T15}]$;

$\rho = \{\mathrm{P4}, \mathrm{P9}\}$; $P_{\mathrm{SM}} = \emptyset$; $T_{\mathrm{SM}} = \{\mathrm{T14}, \mathrm{T15}\}$;

*idle_pairs* $= \{(\mathrm{P7}, [\mathrm{P7\ T11\ P8\ T12}])\}$;

**Iteration 3.1** (place P4 $\in \rho$)

$\rho = \{\mathrm{P9}\}$; $p_1 = \mathrm{P4}$;

$\tau = \mathrm{P4}^{\bullet} \setminus (T_{\mathrm{SM}} \cup \mathit{visited\_trans}) =$

$\{\mathrm{T7}, \mathrm{T14}\} \setminus (\{\mathrm{T14}, \mathrm{T15}\} \cup \{\mathrm{T1} - \mathrm{T10}, \mathrm{T13}\}) = \emptyset$;

$P_{\mathrm{SM}} = \{\mathrm{P4}\}$;

**Iteration 3.2** (place P9 $\in \rho$)

$\rho = \emptyset$; $p_1 = \mathrm{P9}$;

$\tau = \mathrm{P9}^{\bullet} \setminus (T_{\mathrm{SM}} \cup \mathit{visited\_trans}) =$

$\{\mathrm{T15}, \mathrm{T16}\} \setminus (\{\mathrm{T14}, \mathrm{T15}\} \cup \{\mathrm{T1} - \mathrm{T10}, \mathrm{T13}\}) = \{\mathrm{T16}\}$;

$P_{\mathrm{SM}} = \{\mathrm{P4}, \mathrm{P9}\}$;

**Iteration 3.2.1** (transition T16 $\in \tau$)

$t = \mathrm{T16}$; $\tau = \emptyset$; $\pi' = [\mathrm{P9\ T16\ P10\ T17}]$;

$p_{\mathrm{k}} \notin f_P(\pi') \implies$

*idle_pairs* $= \{(\mathrm{P7}, [\mathrm{P7\ T11\ P8\ T12}]),$

$(\mathrm{P9}, [\mathrm{P9\ T16\ P10\ T17}])\}$;

*iteration_blocks* $= \{(\mathrm{P0}, \{\mathrm{P0}, \mathrm{P1}, \mathrm{P2}, \mathrm{P3}, \mathrm{P4}, \mathrm{P5}\},$

$\{\mathrm{T1}, \mathrm{T2}, \mathrm{T3}, \mathrm{T4}, \mathrm{T5}, \mathrm{T6}, \mathrm{T7}, \mathrm{T8}\}),$

$(\mathrm{P2}, \{\mathrm{P2}, \mathrm{P6}, \mathrm{P7}\}, \{\mathrm{T9}, \mathrm{T10}, \mathrm{T13}\}),$

$(\mathrm{P4}, \{\mathrm{P4}, \mathrm{P9}\}, \{\mathrm{T14}, \mathrm{T15}\})\}$;

*visited_trans* $= \{\mathrm{T1} - \mathrm{T10}, \mathrm{T13} - \mathrm{T15}\}$;

Although, for the sake of concision, the computation of the three last elementary iteration blocks (i.e., Iterations 4-6) has been omitted, the reader can easily check that the final result of the computation fits the one depicted in Fig. 2.4.

In the following, a number of properties concerning the set of elementary blocks computed by Algorithm 2.2 are proven. These are later used to reveal the particular morphology of the metastructure that synthesises the connection between the elementary blocks of an iterative state machine. This morphology is ultimately symbolised through a metamodel called the *shrinking graph* of the iterative state machine.

**Lemma 2.4.** *Let $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ be an iterative state machine, $p_0 \in P$ be its idle place, and $\mathcal{B}_{\mathcal{N}}$ be the set of elementary iteration blocks of $\mathcal{N}$. There exists one and only one elementary block $B_0 \in \mathcal{B}_{\mathcal{N}}$ such that $p_0 \in B_0$.*

*Proof.* Before the main loop of Algorithm 2.2, the variable *idle_pairs* is initialised with the set $\{(p_0, \mathrm{Min\_Circuit}(\mathcal{N}, p_0))\}$. This set contains a unique pair, where place

$p_0$ indicates the idle place for the construction of the elementary iteration block. To accomplish that we will use as seed a minimal circuit of $\mathcal{N}$ containing $p_0$. From this circuit, the algorithm adds all circuits containing place $p_0$. This means that no further step will be able to add a strongly connected state machine containing $p_0$ because none of the places identified with the computed state machine will be visited in future searches. □

Observe that, from Definition 2.2, two different elementary iteration blocks obtained from a same iterative state machine can share, at most, a place that will be the idle place of one of the two elementary iteration blocks, and the sets of transitions of the two elementary iteration blocks are disjoint. This property is formalised in the following Property.

**Property 2.5.** *Let $B_i$ and $B_j$ be two distinct elementary iteration blocks obtained from an iterative state machine, where $B_i = (p_i, P_{\mathrm{SM}_i}, T_{\mathrm{SM}_i})$ and $B_j = (p_j, P_{\mathrm{SM}_j}, T_{\mathrm{SM}_j})$:*

**a)** $P_{\mathrm{SM}_i} \cap P_{\mathrm{SM}_j} = \{p_i\}$ *or* $P_{\mathrm{SM}_i} \cap P_{\mathrm{SM}_j} = \{p_j\}$ *or* $P_{\mathrm{SM}_i} \cap P_{\mathrm{SM}_j} = \emptyset$.

**b)** $T_{\mathrm{SM}_i} \cap T_{\mathrm{SM}_j} = \emptyset$.

*Proof.* According to Definition 2.2, one of the two elementary iteration blocks will be the first iteration block generated by the partition of Definition 2.2. Without loss of generality, we can suppose that this first elementary iteration block is $B_j$. The only shared place that can be shared with $B_i$ is the place $p_j$. This is the case if the places of $B_i$ play the role of the set $P_2 \cup \{p_{\mathrm{k}}\}$ of Definition 2.2. Otherwise, $P_{\mathrm{SM}_i} \cap P_{\mathrm{SM}_j} = \emptyset$. The set of transitions is disjoint because they are strongly connected state machines that only share one place at most. □

Another interesting property of elementary iteration blocks concerns the fact that all minimal circuits of the iterative state machine are local to a unique iteration block. This obviously matches the intuitive idea of iterative programming, in which all paths of execution within an iteration block are confined to the very own iteration block.

**Lemma 2.6.** *A minimal t-semiflow of an iterative state machine is contained in one and only one elementary iteration block of the iterative state machine.*

*Proof.* In strongly connected state machines, minimal t-semiflows are the minimal circuits of the iterative state machine. Minimal circuits are contained in an elementary iteration block, because Algorithm 2.2 proceeds in the definition of one elementary iteration block by adding minimal circuits containing the idle place of the block. □

From Property 2.5 we can define a partial order relation between the elementary iteration blocks.

**Definition 2.7.** *Let $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$ be an iterative state machine, $p_0 \in P$ be the idle place of $\mathcal{N}$, and $\mathcal{B}_\mathcal{N}$ be the set of elementary iteration blocks of $\mathcal{N}$. We define $\prec$ as a binary relation in $\mathcal{B}_\mathcal{N}$ such that, for every pair of elements $B_i, B_j$ of $\mathcal{B}_\mathcal{N}$, where $B_i = (p_i, P_{\mathrm{SM}_i}, T_{\mathrm{SM}_i})$ and $B_j = (p_j, P_{\mathrm{SM}_j}, T_{\mathrm{SM}_j})$:*

$$B_i \prec B_j \text{ iff } \begin{cases} P_{\mathrm{SM}_i} \cap P_{\mathrm{SM}_j} = \{p_j\}, \\ \qquad or \\ \exists B_k = (p_k, P_{\mathrm{SM}_k}, T_{\mathrm{SM}_k}) \in \mathcal{B}_\mathcal{N} \text{ such that} \\ \qquad B_i \prec B_k \prec B_j \text{ and } B_i \neq B_k \neq B_j. \end{cases}$$

**Lemma 2.8.** *The binary relation $\prec$ defined on the set $\mathcal{B}_\mathcal{N}$ of the elementary iteration blocks of an iterative state machine is a strict partial order relation.*

*Proof.* The relation is irreflexive since (i) $P_{\mathrm{SM}_i} \cap P_{\mathrm{SM}_i} \neq \{p_i\}$ because every iterative state machine contains at least two places by Definition 2.2, and so does every elementary iteration block by Algorithm 2.2, and (ii) $\nexists B_k \in \mathcal{B}_\mathcal{N}, B_k \neq B_i : B_i \prec B_k \prec B_i$. Otherwise, following a recursive reasoning, there should exist a sequence of elementary iteration blocks $B_i$, ..., $B_n$, with $n > k > i$, such that $B_i \prec ... \prec B_k \prec ... \prec B_n$, where $B_n = B_i$ and every elementary iteration block $B_j$, with $j \in [i, n)$, shares (only) the idle place $p_{j+1}$ with the next block in the sequence, $B_{j+1}$. Now let $B_\alpha \prec B_{\alpha+1} \prec ... \prec B_\omega$, where $n \geq \omega > \alpha \geq i$, be the shortest subsequence contained in it such that all its elements are distinct except for $B_\alpha = B_\omega$. Obviously, one such subsequence with two or more elements must exist since $B_k \neq B_i$. Then for every elementary iteration block $B_j$, with $j \in [\alpha, \omega)$, an elementary path $\pi_j$ can be constructed from the idle place $p_j$ to the next idle place $p_{j+1}$. By concatenating the set of paths, $\pi_\alpha ... \pi_\omega$ we obtain a minimal circuit which contains transitions from at least two different elementary iteration blocks, which is inconsistent with Lemma 2.6, reaching a contradiction.

On the other hand, the relation is trivially transitive according to Definition 2.7, since it explicitly states that $B_i \prec B_k$ and $B_k \prec B_j$, with $B_i \neq B_k \neq B_j$, implies that $B_i \prec B_j$. Note that, since the relation is irreflexive and transitive, it must be asymmetric too. □

**Definition 2.9.** *Let $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$ be an iterative state machine, $p_0 \in P$ be the idle place of $\mathcal{N}$, and $\mathcal{B}_\mathcal{N}$ be the set of elementary iteration blocks of $\mathcal{N}$. The shrinking graph associated to $\mathcal{N}$, $\mathcal{G}_\mathcal{N}$, is a labelled (directed) graph $\mathcal{G}_\mathcal{N} = (V, E, L)$, where:*

1. $V = \mathcal{B}_\mathcal{N}$,
2. $E \subseteq V \times V : (B_i, B_j) \in E$ *iff* $B_i \prec B_j$ *and* $P_{\mathrm{SM}_i} \cap P_{\mathrm{SM}_j} = \{p_j\}$,
3. $L : E \to P : L((B_i, B_j)) = P_{\mathrm{SM}_i} \cap P_{\mathrm{SM}_j} = \{p_j\}$.

Figure 2.5 illustrates the shrinking graph associated to the net in Fig. 2.4.

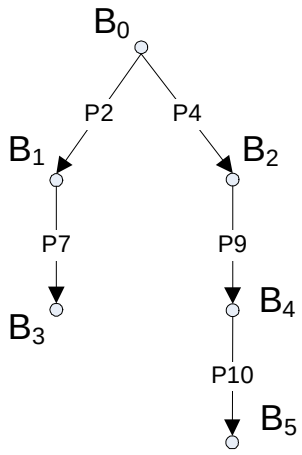



**Figure 2.5**: Shrinking graph of the iterative state machine in Fig. 2.4

**Lemma 2.10.** *Let $\mathcal{N}$ be an iterative state machine, $p_0$ the idle place of $\mathcal{N}$ and $\mathcal{G}_\mathcal{N}$ the shrinking graph associated to $\mathcal{N}$. $\mathcal{G}_\mathcal{N}$ is a rooted tree and the root node is the elementary iteration block containing the idle place $p_0$.*

*Proof.* First, Lemma 2.4 determines that there exists one and only one elementary iteration block $B_0 \in \mathcal{B}_\mathcal{N}$ containing the idle place $p_0$. Since, by the way Algorithm 2.2 proceeds, every minimal circuit in $\mathcal{N}$ containing $p_0$ belongs to $B_0$ then $\nexists B_i \in \mathcal{B}_\mathcal{N} : B_i \prec B_0$, and therefore $B_0$ has no input arcs in the shrinking graph.

Second, the shrinking graph is an acyclic graph, since there exists an arc between two nodes $B_i$ and $B_j$ only if $B_i \prec B_j$, and the binary relation $\prec$ is an strict partial order relation (Lemma 2.8).

Third, considering the way Algorithm 2.2 works, the shrinking graph must be connected. In there, the construction of a new block is started only if there exists a pair in *idle_pairs*. A pair in *idle_pairs* is only introduced when inspecting another block which shares a special place with it: this place will be the idle place of the new block. Therefore, an arc must exist between both blocks.

Fourth and last, each node in the shrinking graph can only have one parent node. If it were not so, all parent blocks would share a common place: the idle place of the child block. Since two blocks share a place only if this is the idle place of one of the two (Property 2.5), then the idle place of the child node should be the idle place of (at least) one of its parents too. This is impossible considering the way Algorithm 2.2 works: no pair $(p_1, \pi')$ can ever be added to *idle_pairs* such that $p_1 = p_k$. □

The last lemma is straightforward considering that an elementary iteration block is a strongly connected state machine:

**Lemma 2.11.** *Each elementary iteration block is conservative and consistent.*

So far, it has been established how the models of the processes that comprise the system are constructed. In the next subsection, we will study how these models are composed together with the resources to build a global model of the system that meets the list of requirements introduced in Sect. 2.2. This is materialised through the class of models PC$^2$R.

### 2.3.2 Definition

PC$^2$R nets are modular models composed by iterative state machines and shared resources. Two PC$^2$R nets can be composed into a new PC$^2$R model via fusion of the common resources. Note that a PC$^2$R net can simply be one process modelled by an iterative state machine along with the set of resources it uses.

The class supports iterative processes, multiple resource acquisitions, non-blocking wait operations and resource lending. Inhibition mechanisms (such as processes that can wait for a zero value of a semaphore) are not natively supported in general, although some cases can still be modelled with PC$^2$R nets.

**Definition 2.12.** *Let $I_{\mathcal{N}}$ be a finite set of indices. A net of Processes Competing for Conservative Resources (PC$^2$R net) is a connected generalised self-loop free P/T net $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ where:*

1. *$P = P_0 \cup P_{\mathrm{S}} \cup P_{\mathrm{R}}$ is a partition such that:*
   - *(a)* [idle places] $P_0 = \bigcup_{i \in I_{\mathcal{N}}} \{p_{0_i}\}$.
   - *(b)* [process places] $P_{\mathrm{S}} = \bigcup_{i \in I_{\mathcal{N}}} P_i$, *where:* $\forall i \in I_{\mathcal{N}} : P_i \neq \emptyset$ *and* $\forall i, j \in I_{\mathcal{N}} : i \neq j,\ P_i \cap P_j = \emptyset$.
   - *(c)* [resource places] $P_{\mathrm{R}} = \{r_1, ..., r_n\}, n > 0$.
2. *$T = \bigcup_{i \in I_{\mathcal{N}}} T_i$, where $\forall i \in I_{\mathcal{N}}, T_i \neq \emptyset$, and $\forall i, j \in I_{\mathcal{N}}, i \neq j, T_i \cap T_j = \emptyset$.*
3. *For all $i \in I_{\mathcal{N}}$ the subnet generated by $\{p_{0_i}\} \cup P_i, T_i$ is an iterative state machine. This is called the $i$-th* process subnet.
4. *For each $r \in P_{\mathrm{R}}$, there exists a unique minimal p-semiflow $\mathbf{y_r} \in \mathbb{N}^{|P|}$ such that $\{r\} = \|\mathbf{y_r}\| \cap P_{\mathrm{R}}$, $\|\mathbf{y_r}\| \cap (P_0 \cup P_{\mathrm{S}}) \neq \emptyset$, and $\mathbf{y_r}[r] = 1$.*
5. *$P_{\mathrm{S}} \subseteq \bigcup_{r \in P_{\mathrm{R}}} (\|\mathbf{y_r}\| \setminus \{r\})$.*

In fulfilment of Requirement 8, the support of the $\mathbf{y_r}$ p-semiflows (point 4 of Definition 2.12) may include $P_0$: this is new with respect to S$^4$PR nets. For such a resource place $r$, there exists at least a process which creates (*lends*) instances of $r$.

As a consequence, there might exist additional minimal p-semiflows containing more than one resource place. This is also new and will be discussed in Subsection 2.3.3.

In the following, $\mathcal{B}_{\mathcal{N}}$ denotes the set of elementary iteration blocks of $\mathcal{N}$, i.e., the set of elementary iteration blocks of each process subnet of $\mathcal{N}$. Since all process subnets are disjoint, the shrinking graph associated to $\mathcal{N}$, $\mathcal{G}_{\mathcal{N}}$, is a forest of $|I_{\mathcal{N}}|$ rooted trees holding the conditions of Lemma 2.10.

The next definition is strongly related to the notion of acceptable initial marking introduced for the $S^4PR$ class. In software systems all processes/threads are initially inactive and start from the same point (the `begin` statement). Hence, all of the corresponding tokens are in the idle place at the initial marking (the process places being therefore empty). The definition takes this into account and establishes a lower bound for the marking of the resource places.

**Definition 2.13.** *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ be a $PC^2R$ net. An initial marking $\mathbf{m_0}$ is* acceptable at level 0 (0-acceptable) *for $\mathcal{N}$ iff $\|\mathbf{m_0}\| \setminus P_R = P_0$, and $\forall p \in P_S, r \in P_R$ :*

$$\mathbf{y_r}^{\mathrm{T}} \cdot \mathbf{m_0} = \mathbf{m_0}[r] + \sum_{p_0 \in P_0 \cap \|\mathbf{y_r}\|} \mathbf{y_r}[p_0] \cdot \mathbf{m_0}[p_0] \geq \mathbf{y_r}[p]$$

.

Note that Definition 2.13 traces a lower bound for the initial marking of each resource place $r \in P_R$ (namely, $\forall p \in P_S : \mathbf{m_0}[r] \geq \mathbf{y_r}[p] - \sum_{p_0 \in P_0 \cap \|\mathbf{y_r}\|} \mathbf{y_r}[p_0] \cdot \mathbf{m_0}[p_0]$). If the marking of some resource place is below that bound, then there exists at least one dead transition at the initial marking, as later proved in Subsection 2.4.1. But having a marking which is above that bound does not guarantee, in the general case, that there do not exist dead transitions. Again, all of this is more profoundly discussed in Subsection 2.4.1.

Nevertheless, it is worth stressing at this point that an $S^4PR$ net system with an acceptable initial marking cannot have any dead transition at the initial marking (since every minimal t-semiflow is firable in isolation from it). Hence the use of the suffix *at level 0* in Definition 2.13, in spite of introducing a concept which collapses with the definition of acceptable initial marking for the $S^4PR$ subclass, as proved a bit later in Subsection 2.3.3. By refining this definition, the fact that they may induce different behavioural patterns in the general case is stressed. Note that in earlier works [LGC12], this refining is not used. However, it is introduced here for the sake of clarity.

Note that in the following, when the context does not allow any ambiguity, the term *acceptable initial marking* may be used to refer to 0-acceptable initial markings in a more concise way.

Finally, note that resource places whose support includes some idle place may be empty for a 0-acceptable initial marking. Figure 2.2 shows a $PC^2R$ net with a 0-acceptable initial marking. This net does not belong to the $S^4PR$ subclass.

### 2.3.3 Hierarchy of classes and p-semiflows

In the following definition we stress the main differences between the well-known classes of the $S^nPR$ family. Observe that these differences are essentially associated to the places belonging to the support of the p-semiflows of the resources.

**Definition 2.14.** *Previous classes of the $S^nPR$ family are defined as follows:*

- *An* $S^5PR$ net *[LGC06] is a $PC^2R$ net where* $\forall r \in P_R : \|\mathbf{y_r}\| \cap P_0 = \emptyset$.
- *An* $S^4PR$ net *[Tri03, TGVCE05] is an $S^5PR$ net where* $\forall i \in I_{\mathcal{N}}$ *the subnet generated by* $\{p_{0_i}\} \cup P_i, T_i$ *is a strongly connected state machine in which every cycle contains* $p_{0_i}$ *(i.e., a iterative state machine with no internal cycles).*
- *An* $S^3PR$ net *[ECM95] is an $S^4PR$ net where* $\forall p \in P_S : |{}^{\bullet\bullet}p \cap P_R| = 1$ *and* ${}^{\bullet\bullet}p \cap P_R = p^{\bullet\bullet} \cap P_R$.
- *An* L-$S^3PR$ net *[EGVC98] is an $S^3PR$ net where* $\forall p \in P_S : |{}^{\bullet}p| = |p^{\bullet}| = 1$.

**Remark 2.15.** *L-$S^3PR \subseteq S^3PR \subseteq S^4PR \subseteq S^5PR \subseteq PC^2R$.*

The preceding remark is straightforward from Definition 2.14. It is worth noting that Definition 2.13 collapses with the definition of acceptable initial markings respectively provided for those subclasses [ECM95, EGVC98, TGVCE05]. That for the $S^4PR$ subclass is provided in Definition 1.2. In a similar vein, we can define the concept of acceptable initial marking for the $S^5PR$ subclass:

**Definition 2.16.** *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$ be an $S^5PR$ net. An initial marking $\mathbf{m_0}$ is acceptable for $\mathcal{N}$ iff $\|\mathbf{m_0}\| \setminus P_R = P_0$, and $\forall p \in P_S, r \in P_R : \mathbf{m_0}[r] \geq \mathbf{y_r}[p]$.*

The latter also collapses with Definition 2.13, since $\mathbf{y_r}^T \cdot \mathbf{m_0} = \mathbf{y_r}[r] \cdot \mathbf{m_0}[r] + \sum_{r' \in P_R \setminus \{r\}} (\mathbf{y_r}[r'] \cdot \mathbf{m_0}[r']) + \sum_{p \in P_S} (\mathbf{y_r}[p] \cdot \mathbf{m_0}[p]) + \sum_{p_0 \in P_0} (\mathbf{y_r}[p_0] \cdot \mathbf{m_0}[p_0]) = \mathbf{m_0}[r]$ (mind that, by Definition 2.12, $\mathbf{y_r}[r] = 1, \|\mathbf{y_r}\| \cap P_R = \{r\}$ and $\mathbf{m_0}[P_S] = \mathbf{0}$ and, by Definition 2.14, $\|\mathbf{y_r}\| \cap P_0 = \emptyset$).

Besides, there exists another class for Sequential RASs, called SPQR [LGC06], which is dealt with in Sect. 2.5. This class does not strictly contain or is contained by the $PC^2R$ class. Yet, there exist transformation rules to travel between $PC^2R$ and Structurally Bounded (SB) SPQR nets. Note that, by construction, $PC^2R$ nets are conservative, and hence SB, but this is not true for SPQR nets. The SPQR class seems interesting from an analytical point of view thanks to its syntactic simplicity, as discussed in Subsection 2.5.

One perspective for inspecting the differences between the subclasses of $PC^2R$ is that of the form and number of minimal p-semiflows. All subclasses are conservative by definition. Let $\mathbf{y_{S_i}}$ denote the unique minimal p-semiflow induced by the iterative state machine generated by restricting $\mathcal{N}$ to $\langle \{p_{0_i}\} \cup P_i, T_i \rangle$.

**Lemma 2.17.** *Let $\mathcal{N} = \langle P, T, C\rangle$ be a PC²R net. A basis of the left null space of the incidence matrix $\mathbf{C}$ contains $|P_{\mathrm{R}}| + |I_{\mathcal{N}}|$ vectors.*

*Proof.* For every $r \in P_{\mathrm{R}}$, let $\mathbf{y}_{\boldsymbol{r}}^{-} = \mathbf{y}_{\boldsymbol{r}} - \sum_{i \in I_{\mathcal{N}}} \mathbf{y}_{\boldsymbol{r}}[p_{0_i}] \cdot \mathbf{y}_{\mathbf{S}_{\boldsymbol{i}}}$. The set of vectors $A = \{\mathbf{y}_{\mathbf{S}_{\boldsymbol{i}}} \mid i \in I_{\mathcal{N}}\} \cup \{\mathbf{y}_{\boldsymbol{r}}^{-} \mid r \in P_{\mathrm{R}}\}$ contains $|P_{\mathrm{R}}| + |I_{\mathcal{N}}|$ vectors and they are linearly independent, because for each $p \in P_{\mathrm{R}} \cup P_0$ there exists one and only one distinct vector $\mathbf{y} \in A$ such that $\mathbf{y}[p] = 1$ and $\forall \mathbf{y}' \in A$, $\mathbf{y}' \neq \mathbf{y}$, $\mathbf{y}'[p] = 0$. Moreover, $A$ is a basis because there is no other vector linearly independent with the vectors of $A$. This last statement is proved by contradiction. Let us suppose that there exists $\mathbf{y}$ being a left annuller of $\mathbf{C}$ and $\mathbf{y}$ cannot be generated from vectors of $A$. Construct the vector $\mathbf{y}' = \mathbf{y} - \sum_{i \in I_{\mathcal{N}}} \mathbf{y}[p_{0_i}] \cdot \mathbf{y}_{\mathbf{S}_{\boldsymbol{i}}} - \sum_{r \in P_{\mathrm{R}}} \mathbf{y}[r] \cdot \mathbf{y}_{\boldsymbol{r}}^{-}$. Obviously, $\mathbf{y}'$ is a left annuller of $\mathcal{N}$ because so are the vectors of $A$ and $\mathbf{y}$, but $\|\mathbf{y}'\| \subseteq P_{\mathrm{S}}$, and this is not possible because there are no annullers of the iterative state machines without their idle places. □

Now let $\mathbf{B}$ be a matrix of dimensions $(|P_{\mathrm{R}}| + |I_{\mathcal{N}}|) \times |P|$ of integers such that the rows of $\mathbf{B}$ are the set of vectors $A$ defined in the previous proof.

**Lemma 2.18.** *If $\mathcal{N}$ is an S⁵PR net, $\mathbf{B}$ is a non-negative canonical basis of p-semiflows.*

*Proof.* By reordering columns in $\mathbf{B}$ so that the first ones correspond to $P_{\mathrm{R}} \cup P_0$, and subsequently reordering the rows of $\mathbf{B}$, a matrix of the form $[\mathbf{I} \mid \mathbf{B}']$ is obtained, where $\mathbf{I}$ is the identity matrix of dimension $|P_{\mathrm{R}}| + |I_{\mathcal{N}}|$, and $\mathbf{B}'$ is a matrix of dimensions $(|P_{\mathrm{R}}| + |I_{\mathcal{N}}|) \times |P_{\mathrm{S}}|$ of non-negative integers, since in S⁵PR nets, $\forall r \in P_{\mathrm{R}}$, $\mathbf{y}_{\boldsymbol{r}}[P_0] = 0$ and $\mathbf{y}_{\boldsymbol{r}}^{-} = \mathbf{y}_{\boldsymbol{r}}$. □

**Corollary 2.19.** *If $\mathcal{N}$ is an S³PR net, then every row in $\mathbf{B}$ belongs to $\{0,1\}^{|P|}$.*

However, nets belonging to the S⁴PR class may have non-binary minimal p-semiflows. Furthermore, PC²R nets feature a new kind of minimal p-semiflows:

**Lemma 2.20.** *If $\mathcal{N}$ is a PC²R net but not an S⁵PR net, there exists at least one minimal p-semiflow whose support contains more than one resource place.*

*Proof.* Since $\mathcal{N}$ is not an S⁵PR net, there exists $r \in P_{\mathrm{R}}$, $i \in I_{\mathcal{N}}$ such that $p_{0_i} \in \|\mathbf{y}_{\boldsymbol{r}}\|$. In that case, there exists at least one $p \in P_i$ such that $p \notin \|\mathbf{y}_{\boldsymbol{r}}\|$ (otherwise, $\mathbf{y}_{\boldsymbol{r}} - \mathbf{y}_{\mathbf{S}_{\boldsymbol{i}}} \geq \mathbf{0}$ and $\mathbf{y}_{\boldsymbol{r}}$ is not minimal). Let $A$ be the minimal set of minimal p-semiflows, $\{\mathbf{y}_{\boldsymbol{u}} \mid u \in P_{\mathrm{R}}\}$, that are essential to cover every $p \in P_i$ such that $\mathbf{y}_{\boldsymbol{r}}[p] = 0$. This means that $\mathbf{y}_{\boldsymbol{u}} \in A$ iff $\exists p \in P_i$, $\mathbf{y}_{\boldsymbol{r}}[p] = 0$, $\mathbf{y}_{\boldsymbol{u}}[p] \neq 0$ and $\forall \mathbf{y}_{\boldsymbol{v}} \in A$, $\mathbf{y}_{\boldsymbol{v}} \neq \mathbf{y}_{\boldsymbol{u}} : \mathbf{y}_{\boldsymbol{v}}[p] = 0$. By Definition 2.12.5, this set $A$ exists. We construct $\mathbf{y}'_{\boldsymbol{r}} = \mathbf{y}_{\boldsymbol{r}} + \sum_{\mathbf{y}_{\boldsymbol{u}} \in A} \mathbf{y}_{\boldsymbol{u}}$, that obviously contains, at least, the p-semiflow $\mathbf{y}_{\mathbf{S}_{\boldsymbol{i}}}$. Therefore, $\mathbf{y}_{\boldsymbol{r}}^{-} = \mathbf{y}'_{\boldsymbol{r}} - \mathrm{k} \cdot \mathbf{y}_{\mathbf{S}_{\boldsymbol{i}}}$, $\mathrm{k} = \min_{p \in \|\mathbf{y}_{\mathbf{S}_{\boldsymbol{i}}}\|} \{\mathbf{y}'_{\boldsymbol{r}}[p]\}$, is a p-semiflow by construction, and $\mathbf{y}_{\boldsymbol{r}}^{-}[P_{\mathrm{R}}] = \mathbf{y}'_{\boldsymbol{r}}[P_{\mathrm{R}}]$. In the same way, we detract other $\mathbf{y}_{\mathbf{S}_{\boldsymbol{j}}}$ that can be contained in $\mathbf{y}_{\boldsymbol{r}}^{-}$.

If $\mathbf{y}_{\boldsymbol{r}}^{-}$ is a minimal p-semiflow then the lemma is true, since the support of $\mathbf{y}_{\boldsymbol{r}}^{-}$ contains more than one resource place. Otherwise, three facts must be considered:

1. By construction, $\mathbf{y}_{\boldsymbol{r}}^{-}$ cannot contain any $\mathbf{y}_{\mathbf{S}_j}$, $\forall j \in I_{\mathcal{N}}$, since all those have been maximally detracted.

2. The p-semiflow $\mathbf{y}_{\boldsymbol{r}}^{-}$ does not contain any minimal p-semiflow $\mathbf{y}_{\boldsymbol{v}}$ covering one single resource place. By contradiction. Let us suppose that $\mathbf{y}_{\boldsymbol{r}}^{-}$ contains $\mathbf{y}_{\boldsymbol{v}}$. The resource $v$ must be either $r$ or a resource for which its minimal p-semiflow $\mathbf{y}_{\boldsymbol{v}}$ is in $A$. If $v$ is $r$ then $\mathbf{y}_{\boldsymbol{v}}[r] = \mathbf{y}_{\boldsymbol{r}}^{-}[r]$, and $(\mathbf{y}_{\boldsymbol{r}}^{-} - \mathbf{y}_{\boldsymbol{v}})[p_{0_i}] < 0$. Therefore, a contradiction is reached. If $v$ holds $\mathbf{y}_{\boldsymbol{v}} \in A$ then $\mathbf{y}_{\boldsymbol{v}}[v] = \mathbf{y}_{\boldsymbol{r}}^{-}[v]$ because the weight $\mathbf{y}_{\boldsymbol{r}}^{-}[u]$ is not modified by the previous operations. And $(\mathbf{y}_{\boldsymbol{r}}^{-} - \mathbf{y}_{\boldsymbol{v}})[p] < 0$ for some $p \in P_i$ for which $\mathbf{y}_{\boldsymbol{v}}$ becomes essential in the set $A$, since $\mathbf{y}_{\boldsymbol{r}}^{-}[p] \leq \mathbf{y}_{\boldsymbol{v}}[p] - k$.

3. Last, there must exist one minimal p-semiflow $\mathbf{y}_{\boldsymbol{r}}^{--}$ such that $\|\mathbf{y}_{\boldsymbol{r}}^{--}\| \subset \|\mathbf{y}_{\boldsymbol{r}}^{-}\|$ (otherwise, $\mathbf{y}_{\boldsymbol{r}}^{-}$ is minimal). Considering the two previous facts, the support of $\mathbf{y}_{\boldsymbol{r}}^{--}$ contains more than one resource place.

□

In other words, the above result reveals that the set of minimal p-semiflows contains strictly a basis of the left null space of the incidence matrix $\mathbf{C}$.

### 2.3.4 Basic structural properties

Trivially, the next lemma follows:

**Lemma 2.21.** *Every PC²R net is conservative.*

On the other hand, each PC²R net is consistent, and so is every net belonging to the rest of subclasses.

**Lemma 2.22.** *Every PC²R net is consistent.*

*Proof.* Let $\mathcal{N}$ be a PC²R net. The process subnets of $\mathcal{N}$ are strongly connected state machines (indeed, they are iterative state machines) and therefore each one is consistent, i.e., every transition $t$ of $\mathcal{N}$ is covered by at least a t-semiflow of the iterative state machine containing $t$. It will be proved that these t-semiflows are also t-semiflows of the net $\mathcal{N}$. Indeed, if $\mathbf{x}$ is a t-semiflow of $\mathcal{N}$ without resources it is enough to prove that $\forall r \in P_{\mathrm{R}} : \mathbf{C}[r, T] \cdot \mathbf{x} = 0$. Taking into account Definition 2.12, point 4, $\mathbf{C}[r, T] = -\sum_{p \in \|\mathbf{y}_{\boldsymbol{r}}\| \setminus \{r\}} \mathbf{y}_{\boldsymbol{r}}[p] \cdot \mathbf{C}[p, T]$, and therefore:

$$\mathbf{C}[r, T] \cdot \mathbf{x} = -\left( \sum_{p \in \|\mathbf{y}_{\boldsymbol{r}}\| \setminus \{r\}} \mathbf{y}_{\boldsymbol{r}}[p] \cdot \mathbf{C}[p, T] \right) \cdot \mathbf{x} = -\sum_{p \in \|\mathbf{y}_{\boldsymbol{r}}\| \setminus \{r\}} \mathbf{y}_{\boldsymbol{r}}[p] \cdot \mathbf{C}[p, T] \cdot \mathbf{x} = 0.$$

Hence, $\mathcal{N}$ is consistent. □

Observe that conservative and consistent nets are called well-formed nets [Ter04] because this is a necessary condition for structural liveness and structural boundedness.

The next lemma is rather obvious and a well-known result in the Petri net literature but, for practical reasons, it is nonetheless stated for our restricted class of nets since it will often be used in proofs of this and some other forthcoming chapters.

**Lemma 2.23.** *Let $\mathcal{N}$ be a conservative P/T net, and $\mathbf{y} \in \mathbb{N}^{|P|}$ be a p-semiflow of $\mathcal{N}$. The support of $\mathbf{y}$, $\|\mathbf{y}\|$, is both a siphon and a trap of the net.*

*Proof.* Since $\mathbf{y}$ is a p-semiflow, each column of the incidence matrix must be annulled by $\mathbf{y}$, i.e., $\mathbf{y} \cdot \mathbf{C}[P,t] = 0$, for every $t \in T$. In order to annul each column for its corresponding transition $t$ in the set ${}^\bullet(\|\mathbf{y}\|) \cup (\|\mathbf{y}\|)^\bullet$ at least one input and one output place of $t$ is needed. Therefore ${}^\bullet(\|\mathbf{y}\|) = (\|\mathbf{y}\|)^\bullet$, i.e., $\|\mathbf{y}\|$ is both a siphon and a trap. □

Now an analogous lemma can be stated for the dual case of t-semiflows. This lemma plays an instrumental role in subsequent proofs.

**Lemma 2.24.** *Let $\mathcal{N}$ be a consistent P/T net. Every t-semiflow $\mathbf{x}$ of $\mathcal{N}$ holds ${}^\bullet\|\mathbf{x}\| = \|\mathbf{x}\|^\bullet$.*

*Proof.* The dual net of a consistent net is a conservative net. By Lemma 2.23, the dual p-semiflow of $\mathbf{x}$ is the support of both a siphon and a trap. Then ${}^\bullet\|\mathbf{x}\| = \|\mathbf{x}\|^\bullet$. □

The proof of Lemma 2.22 shows that a minimal circuit is always the support of a t-semiflow (and vice versa). On the other hand, Lemma 2.6 proves that each minimal circuit is local to a unique iteration block. Therefore, the next lemma is closely related to the conservativeness in the use of resources within the circuits of an elementary iteration block. The conservativeness for the whole elementary iteration block is finally proved by Lemma 2.26.

**Lemma 2.25.** *Let $\mathcal{N}$ be a well-formed P/T net, and $\mathbf{x}$ be a t-semiflow of $\mathcal{N}$. The subnet induced by $\mathbf{x}$, i.e., the subnet generated by ${}^\bullet\|\mathbf{x}\|$, $\|\mathbf{x}\|$, is conservative.*

*Proof.* $\mathcal{N}$ is conservative, i.e., $\exists \mathbf{y} \in \mathbb{N}^{|P|}$, $\mathbf{y} > \mathbf{0} : \mathbf{y}^{\mathrm{T}} \cdot \mathbf{C} = \mathbf{0}$. Therefore, $\forall t \in \|\mathbf{x}\| : \mathbf{y}^{\mathrm{T}} \cdot \mathbf{C}[t] = \mathbf{0}$. Since, by Lemma 2.24, ${}^\bullet\|\mathbf{x}\| = \|\mathbf{x}\|^\bullet$, then $\mathbf{C}[P \setminus {}^\bullet\|\mathbf{x}\|, \|\mathbf{x}\|] = \mathbf{0}$. Therefore, $\mathbf{y}^{\mathrm{T}}[{}^\bullet\|\mathbf{x}\|] \cdot \mathbf{C}[{}^\bullet\|\mathbf{x}\|, \|\mathbf{x}\|] = \mathbf{0}$ and the lemma holds. □

**Lemma 2.26.** *Let $\mathcal{N}$ be a $PC^2R$ net, and $B = (p, P_{SM}, T_{SM})$ be an elementary iteration block of a process subnet of $\mathcal{N}$, $B \in \mathcal{B}_{\mathcal{N}}$. The use of resources in $B$ is consistent and conservative (i.e., the subnet generated by ${}^\bullet T_{SM}$, $T_{SM}$ is consistent and conservative).*

*Proof.* Lemma 2.11 proves that every elementary iteration block is consistent in isolation. Since $\mathcal{N}$ is a state machine, the t-semiflow that covers all the transitions in the block is also a t-semiflow of $\mathcal{N}$. On the other hand, Lemma 2.25 proves that the subnet induced by a t-semiflow in $\mathcal{N}$ is conservative. Therefore, the use of resources in $B$ is conservative. □

The above result states that not only a PC$^2$R net is globally well-formed, but also that so is every elementary iteration block augmented with the resource places it interacts with, considered from a local perspective.

Finally, we want to formalise that PC$^2$R nets correspond to an RAS view of systems since the so-named *resource* places have a behaviour according to the actual experience or intuition that software engineers have about conservative resources. This means that if we have enough copies of a type of resource then it is not a constraint to the system of processes and can be removed from the analysis of the RAP. This intuitive idea is captured in Petri nets by means of the concept of Structurally Implicit Place (SIP). An SIP is a place whose row in the incidence matrix can be obtained as a non-negative linear combination of other rows of the incidence matrix. This property, essentially a structural property, leads to that if we have the freedom to select the initial marking for these places, we can make them implicit places and then they can be removed from the net maintaining the same set of occurrence sequences of transitions. The next result points out this structural property.

**Lemma 2.27.** *Let $\mathcal{N}$ be a PC$^2$R net and $r$ a resource place of $\mathcal{N}$. The place $r$ is structurally implicit and*

$$\mathbf{C}[r,T] = -\sum_{p\in\|\mathbf{y_r}\|\setminus\{r\}} \mathbf{y_r}[p]\cdot\mathbf{C}[p,T] + \sum_{\forall i\,:\,\|\mathbf{y_{S_i}}\|\cap\|\mathbf{y_r}\|\neq\emptyset} \mathrm{K}_i\cdot\mathbf{y_{S_i}},$$

*where* $\mathrm{K}_i = \max\{\mathbf{y_r}[p] \mid p\in\|\mathbf{y_{S_i}}\|\}$.

*Proof.* The result is trivially true because from the definition of PC$^2$R net, there exists a unique p-semiflow such that $\{r\} = \|\mathbf{y_r}\|\cap P_{\mathrm{R}}$, $(P_0\cup P_{\mathrm{S}})\cap\|\mathbf{y_r}\|\neq\emptyset$ and $\mathbf{y_r}[r]=1$. Therefore, pre-multiplying the incidence matrix by the vector $\mathbf{y_r}$, we can construct the following equation:

$$\mathbf{C}[r,T] = -\sum_{p\in\|\mathbf{y_r}\|\setminus\{r\}} \mathbf{y_r}[p]\cdot\mathbf{C}[p,T]$$

Obviously, this is a non-negative linear combination of rows of the incidence matrix. But we know that, for each iterative state machine $i$, there exists a minimal p-semiflow $\mathbf{y_{S_i}}\in\{0,1\}^{|P|}$ that we can add to the previous equation because $\mathbf{y_{S_i}}^{\mathrm{T}}\cdot\mathbf{C}=0$, but weighted by the greatest coefficient $\mathbf{y_r}[p]$ of those places $p$ belonging to the $i$-th

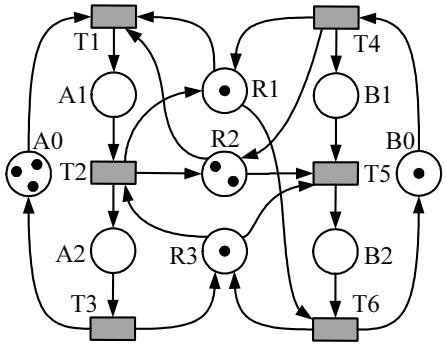



**Figure 2.6**: A rather simple $PC^2R$ net

iterative state machine. Applying this procedure to all iterative state machines visited by the p-semiflow of $r$ we obtain a non-negative linear combination of rows of the incidence matrix as follows:

$$\mathbf{C}[r,T] = -\sum_{p\in\|\mathbf{y_r}\|\setminus\{r\}} \mathbf{y_r}[p]\cdot\mathbf{C}[p,T] + \sum_{\forall i\,:\,\|\mathbf{y_{S_i}}\|\cap\|\mathbf{y_r}\|\neq\emptyset} \mathrm{K}_i\cdot\mathbf{y_{S_i}}$$

where $\mathrm{K}_i = \max\{\mathbf{y_r}[p] \mid p\in\|\mathbf{y_{S_i}}\|\}$. That is, $r$ is a structurally implicit place. □

Observe that this linear combination of process places and possibly some idle places is unique. Nevertheless, we can have other linear combinations including not only process places or idle places, but also other resource places present in the net.

For example, in the net depicted in Fig. 2.6, if we consider the resource place R2, we have two different equations to obtain the row of R2 in the incidence matrix:

$$\mathbf{C}[\mathrm{R2},\cdot\,] = \mathbf{C}[\mathrm{A0},\cdot\,] + \mathbf{C}[\mathrm{A2},\cdot\,] + \mathbf{C}[\mathrm{B1},\cdot\,]$$

$$\mathbf{C}[\mathrm{R2},\cdot\,] = \mathbf{C}[\mathrm{R1},\cdot\,] + \mathbf{C}[\mathrm{R3},\cdot\,] + \mathbf{C}[\mathrm{A2},\cdot\,]$$

Observe that, in the first equation, the implying places are only process places, A2 and B1, as well as the idle place A0. Nevertheless, the second equation contains as implying places the resources R1 and R3.

According to the theory of structurally implicit places [GVC99], if we are considering SIPs in structurally bounded nets then we can compute a finite initial marking for a SIP making it an implicit place. For example, we can compute a bound of the minimal initial marking for the place R2 in the net of Fig. 2.6 to make it implicit. This is accomplished by resolving a linear programming problem [GVC99]. The obtained result is 2, that is, the minimum initial marking for the place R2 to be implicit is equal to 2. Therefore, R2 in the figure is implicit.

This small value for this initial marking of the place R2 is due to the second previous equation because the more restrictive places to the firing of the output transitions of place R2 are the resource places and not the process and idle places of the first equation.

This apparently instrumental result allows us to obtain interesting conclusions. The first one says that we must analyse if the resources of a $PC^2R$ net system are implicit or not, because if some of them are implicit, we can remove them and then we can simplify further analysis (less siphons, for example) or even we can fall in a subclass where we can find stronger results that can be applied. The second conclusion says that we can conclude that these nets are structurally live and structurally bounded.

**Lemma 2.28.** *Every $PC^2R$ net is structurally live and structurally bounded*

*Proof.* The net is conservative, therefore it is structurally bounded. All resource places are structurally implicit places, therefore if we consider an initial marking where: (i) All resource places have an initial marking making each one implicit; and (ii) All idle places contain at least one token; then the net is live. In effect, if the resource places are implicit then we can remove them and the language of occurrence sequences remains unchanged. Therefore, the resulting net is composed by a set of strongly connected state machines, each one containing at least one token, and therefore, each one is live. If our net is live, is structurally live. □

That is, structural implicitness of resources, and the acceptable initial markings guarantees structural liveness and structural boundedness in $PC^2R$.

## 2.4 A cross-sectional view on the liveness analysis problem

So far, it has been introduced a Petri net class (the $PC^2R$ class) which fulfils the list of basic model requirements introduced in Sect. 2.2. In this section, it is proved that finding a structural characterisation of the liveness problem reveals itself much harder than was for previous (sub-)classes used in the FMS context.

The discussion is divided into three parts. In Subsection 2.4.1, the hard problem of establishing an acceptable initial marking in general $PC^2R$ nets is investigated. This problem was trivial for earlier net subclasses in the $S^nPR$ family. In Subsection 2.4.2, it is reminded how siphons structurally capture the problem of liveness in those subclasses, and evidenced that this kind of siphons is no longer sufficient to characterise non-liveness in the multithreaded software domain. Finally, in Subsection 2.4.3, some other properties strongly related to liveness analysis are inspected.

### 2.4.1 Acceptability of the initial marking: The 0-1 zone

In Subsection 2.3.2 it was introduced the notion of 0-acceptable initial marking. As explained there, it is a generalisation of the notion of acceptable initial marking which was introduced for the $\mathrm{S}^n\mathrm{PR}$ subclasses (see, e.g., Definition 1.2 for $\mathrm{S}^4\mathrm{PR}$, and Definition 2.16 for $\mathrm{S}^5\mathrm{PR}$).

One common factor in those subclasses is that any initial marking which is not acceptable just because of a lack of tokens in the resource places (i.e., it is not acceptable in spite of having $\|\mathbf{m_0}\| \setminus P_\mathrm{R} = P_0$) guarantees that there is a dead transition at the initial marking (and therefore the net system is non-live). Thus, this kind of markings can be discarded from the very beginning.

This is a property which can be extended to $\mathrm{PC}^2\mathrm{R}$ nets and 0-acceptable initial markings. The next theorem proves that. Although it can be trivially extended for a more general class of conservative Petri nets, it is specialised here for $\mathrm{PC}^2\mathrm{R}$ nets for the sake of clarity:

**Theorem 2.29.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a $PC^2R$ net system with an initial marking such that $\|\mathbf{m_0}\| \setminus P_\mathrm{R} = P_0$ but $\exists p \in P_\mathrm{S}, \mathbf{y} \in \mathbb{N}^{|P|}$ such that $\mathbf{y}^\mathrm{T} \cdot \mathbf{C} = \mathbf{0}$ and $\mathbf{y}^\mathrm{T} \cdot \mathbf{m_0} < \mathbf{y}[p]$. Then at least one transition is dead at $\mathbf{m_0}$.*

*Proof.* By multiplying both sides of the net state equation by $\mathbf{y}^\mathrm{T}$ we get: $\mathbf{y}^\mathrm{T} \cdot \mathbf{m} = \mathbf{y}^\mathrm{T} \cdot \mathbf{m_0} + \mathbf{y}^\mathrm{T} \cdot \mathbf{C} \cdot \boldsymbol{\sigma} = \mathbf{y}^\mathrm{T} \cdot \mathbf{m_0}$. Thus, for every $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) : 0 \leq \mathbf{y}^\mathrm{T} \cdot \mathbf{m} = \mathbf{y}^\mathrm{T} \cdot \mathbf{m_0} < \mathbf{y}[p]$. Since $\mathbf{m}, \mathbf{y} \geq \mathbf{0}$ then $\mathbf{y}[p] \cdot \mathbf{m}[p] \leq \mathbf{y}^\mathrm{T} \cdot \mathbf{m} < \mathbf{y}[p]$, and therefore $\mathbf{m}[p] = 0$. Thus $p$ is empty for every reachable marking, so all its input and output transitions must be dead. □

Nevertheless, another common factor for the $\mathrm{S}^5\mathrm{PR}$ and simpler subclasses is that having an acceptable initial marking guarantees that every minimal t-semiflow is eventually realisable from the initial marking. This ultimately implies that every transition is firable an arbitrary number of times since these nets are consistent by Lemma 2.22. This is later discussed in Subsection 2.4.3 and proved in Theorem 2.40.

On the contrary, there exist $\mathrm{PC}^2\mathrm{R}$ net systems with 0-acceptable initial markings such that there exist dead transitions at those initial markings. In some cases, the observation of a set of minimal p-semiflows provides enough information to infer the deadlock. In those cases, at least one of the minimal p-semiflows must have two or more resource places in its support (i.e., the set cannot be a subset of $A$, as defined in Subsection 2.3.3). As a result, such nets cannot belong to the $\mathrm{S}^5\mathrm{PR}$ subclass, since in that case every p-semiflow belongs to the set $A$ by Lemma 2.18.

One of such nets is depicted in Fig. 2.7. The net also proves that the reverse of Theorem 2.29 is false in general for the $\mathrm{PC}^2\mathrm{R}$ class: in this case, no (minimal) p-semiflow provides enough information when observed isolately. This is straightforward

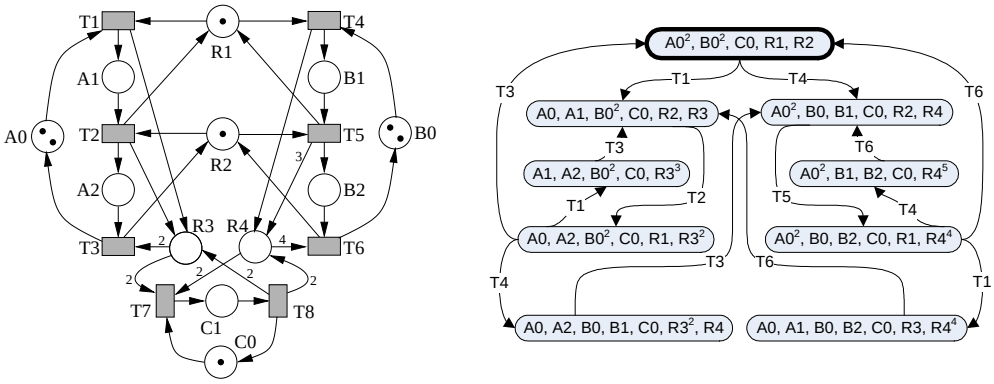



**Figure 2.7**: Non-live PC$^2$R net system with a 0-acceptable initial marking. No (minimal) p-semiflow reveals, when *considered in isolation*, that T7 and T8 are dead at $\mathbf{m_0}$

from the complete set of minimal p-semiflows, represented by the following place invariants:

$$\begin{aligned}
\mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{A1}] + \mathbf{m}[\mathrm{A2}] &= 2\\
\mathbf{m}[\mathrm{B0}] + \mathbf{m}[\mathrm{B1}] + \mathbf{m}[\mathrm{B2}] &= 2\\
\mathbf{m}[\mathrm{C0}] + \mathbf{m}[\mathrm{C1}] &= 1\\
\mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{A1}] + \mathbf{m}[\mathrm{B1}] &= 1\\
\mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{B2}] &= 1\\
\mathbf{m}[\mathrm{R3}] + 2\cdot\mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{A1}] + 2\cdot\mathbf{m}[\mathrm{C1}] &= 4\\
\mathbf{m}[\mathrm{R4}] + 4\cdot\mathbf{m}[\mathrm{B0}] + 3\cdot\mathbf{m}[\mathrm{B1}] + 2\cdot\mathbf{m}[\mathrm{C1}] &= 8\\
2\cdot\mathbf{m}[\mathrm{R1}] + 4\cdot\mathbf{m}[\mathrm{R2}] + 2\cdot\mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + \mathbf{m}[\mathrm{B1}] + 6\cdot\mathbf{m}[\mathrm{C1}] &= 6\\
\mathbf{m}[\mathrm{R1}] + 4\cdot\mathbf{m}[\mathrm{R2}] + 3\cdot\mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + 2\cdot\mathbf{m}[\mathrm{A0}] + 8\cdot\mathbf{m}[\mathrm{C1}] &= 9\\
\mathbf{m}[\mathrm{R1}] + 4\cdot\mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + 2\cdot\mathbf{m}[\mathrm{A2}] + 4\cdot\mathbf{m}[\mathrm{C1}] &= 5\\
\mathbf{m}[\mathrm{R1}] + 4\cdot\mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R4}] + \mathbf{m}[\mathrm{A1}] + 4\cdot\mathbf{m}[\mathrm{A2}] + 2\cdot\mathbf{m}[\mathrm{C1}] &= 5\\
3\cdot\mathbf{m}[\mathrm{R2}] + 3\cdot\mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + 3\cdot\mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{B0}] + 8\cdot\mathbf{m}[\mathrm{C1}] &= 11\\
3\cdot\mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R4}] + 3\cdot\mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{B0}] + 2\cdot\mathbf{m}[\mathrm{C1}] &= 5\\
\mathbf{m}[\mathrm{R1}] + 2\cdot\mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{B1}] + 2\cdot\mathbf{m}[\mathrm{B2}] + 2\cdot\mathbf{m}[\mathrm{C1}] &= 3\\
\mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{B2}] + 2\cdot\mathbf{m}[\mathrm{C1}] &= 3
\end{aligned}$$

Nevertheless, the fact that there exist dead transitions can be determined from the

marking of a subset of minimal p-semiflows. From the invariant $\mathbf{m}[\mathrm{R1}] + 4 \cdot \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + 2 \cdot \mathbf{m}[\mathrm{A2}] + 4 \cdot \mathbf{m}[\mathrm{C1}] = 5$ it can be inferred that $\mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] \geq 1$, since 5 is an odd number. If we combine this result with the invariant $2 \cdot \mathbf{m}[\mathrm{R1}] + 4 \cdot \mathbf{m}[\mathrm{R2}] + 2 \cdot \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + \mathbf{m}[\mathrm{B1}] + 6 \cdot \mathbf{m}[\mathrm{C1}] = 6$ then we conclude that $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) : \mathbf{m}[\mathrm{C1}] = 0$. Since C1 is already empty at $\mathbf{m_0}$, both its input (T7) and output transitions (T8) are dead.

Note that the above system has a solution in the field of the real numbers $[\mathrm{A0}^{0.5}, \mathrm{A1}, \mathrm{A2}^{0.5}, \mathrm{B0}^{1.5}, \mathrm{B2}^{0.5}, \mathrm{R3}^2, \mathrm{R4}^2]$ such that T7 is enabled, in spite of not having any analogous solution in the non-negative integer field. This is due to the convexity of the solution space of this net system. That real number solution is the midpoint of the segment between the reachable markings $[\mathrm{A1}, \mathrm{A2}, \mathrm{B0}^2, \mathrm{R3}^3]$ and $[\mathrm{A0}, \mathrm{A1}, \mathrm{B0}, \mathrm{B2}, \mathrm{R3}^3, \mathrm{R4}^4]$. Therefore, in order to 'cut' this solution from the state space (in other words: to explain that this solution does not belong to it) it is not enough with a unique straight line (i.e., linear equation) but several of them (in this case, two).

In some other cases, however, the death of a transition cannot be detected by taking into account (solely) the set of (minimal) p-semiflows, as proved by the net of Fig. 2.8. In this case, the *non-linear* invariant $(\mathbf{m}[\mathrm{R1}] < 1) \vee (\mathbf{m}[\mathrm{R2}] < 1) \vee (\mathbf{m}[\mathrm{R3}] < 1)$ cannot be inferred from the p-semiflows, since the marking $[\mathrm{A0}, \mathrm{B6}, \mathrm{C0}, \mathrm{R1}, \mathrm{R2}, \mathrm{R3}]$ is a solution of the system which is not a reachable marking. A non-negative canonical basis of p-semiflows follows:

$$
\begin{aligned}
\mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{A1}] + \mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{A3}] + \mathbf{m}[\mathrm{A4}] + \mathbf{m}[\mathrm{A5}] + \mathbf{m}[\mathrm{A6}] &= 1 \\
\mathbf{m}[\mathrm{B0}] + \mathbf{m}[\mathrm{B1}] + \mathbf{m}[\mathrm{B2}] + \mathbf{m}[\mathrm{B3}] + \mathbf{m}[\mathrm{B4}] + \mathbf{m}[\mathrm{B5}] + \mathbf{m}[\mathrm{B6}] &= 1 \\
\mathbf{m}[\mathrm{C0}] + \mathbf{m}[\mathrm{C1}] &= 1 \\
\mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{A1}] + \mathbf{m}[\mathrm{A4}] + \mathbf{m}[\mathrm{B0}] + \mathbf{m}[\mathrm{B3}] + \mathbf{m}[\mathrm{C1}] &= 1 \\
\mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{A5}] + \mathbf{m}[\mathrm{B1}] + \mathbf{m}[\mathrm{B4}] + \mathbf{m}[\mathrm{C1}] &= 1 \\
\mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{A3}] + \mathbf{m}[\mathrm{A6}] + \mathbf{m}[\mathrm{B2}] + \mathbf{m}[\mathrm{B5}] + \mathbf{m}[\mathrm{C1}] &= 1 \\
\mathbf{m}[\mathrm{R4}] + \mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{B3}] + \mathbf{m}[\mathrm{B5}] &= 1 \\
\mathbf{m}[\mathrm{R5}] + \mathbf{m}[\mathrm{A4}] + \mathbf{m}[\mathrm{A6}] + \mathbf{m}[\mathrm{B6}] &= 1 \\
\mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + \mathbf{m}[\mathrm{B0}] + \mathbf{m}[\mathrm{B1}] + \mathbf{m}[\mathrm{B2}] + 2 \cdot \mathbf{m}[\mathrm{B3}] + \\
\mathbf{m}[\mathrm{B4}] + 2 \cdot \mathbf{m}[\mathrm{B5}] + 3 \cdot \mathbf{m}[\mathrm{C1}] &= 3 \\
\mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R5}] + \mathbf{m}[\mathrm{A1}] + \mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{A3}] + 2 \cdot \mathbf{m}[\mathrm{A4}] + \\
\mathbf{m}[\mathrm{A5}] + 2 \cdot \mathbf{m}[\mathrm{A6}] + 3 \cdot \mathbf{m}[\mathrm{C1}] &= 3 \\
\mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + \mathbf{m}[\mathrm{R5}] + \mathbf{m}[\mathrm{A4}] + \mathbf{m}[\mathrm{A6}] + \mathbf{m}[\mathrm{B3}] + \\
\mathbf{m}[\mathrm{B5}] + 3 \cdot \mathbf{m}[\mathrm{C1}] &= 3
\end{aligned}
$$

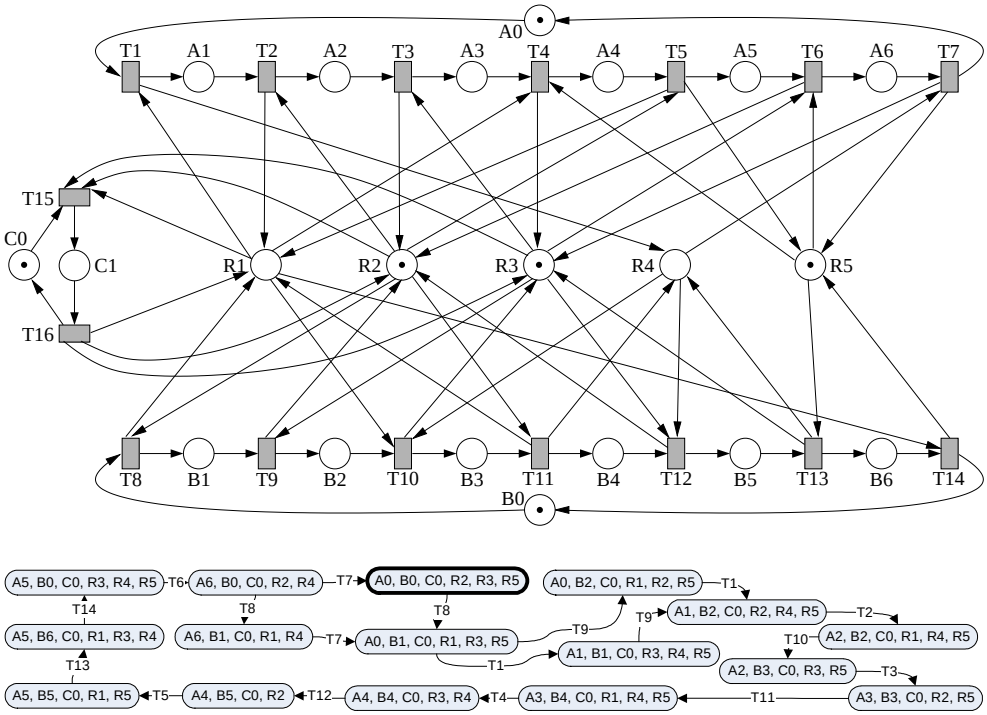



**Figure 2.8**: Non-live but reversible $PC^2R$ net system with a 0-acceptable initial marking. T15 and T16 are dead at $\mathbf{m_0}$. No minimal t-semiflow is ever realisable

Unfortunately, the problem of removing all the spurious solutions of a net system is a very hard one, even for very simple net classes. In Chap. 5 it is proved that the problem of deciding if a net state equation solution is spurious from the net structure is co-NP-complete for the $S^4PR$ subclass. This implies that, in the case of $PC^2R$ net systems, the problem is co-NP-hard.

The above discussion suggests the convenience of establishing a second frontier of initial markings from which the eventual firability of every transition is guaranteed. For coherence with the classical definition of acceptable initial marking, we establish a second frontier from which every minimal t-semiflow is eventually realisable. Note that this is a stronger condition: indeed, there exist live $PC^2R$ net systems in which no minimal t-semiflow is ever realisable, as the net in Fig. 2.9 proves. Observe that the two minimal t-semiflows correspond to the two process paths in the net. Nevertheless, every cycle in the reachability graph of this net system is labelled with the transitions of both t-semiflows. That is, the minimal t-semiflows are not firable in isolation.

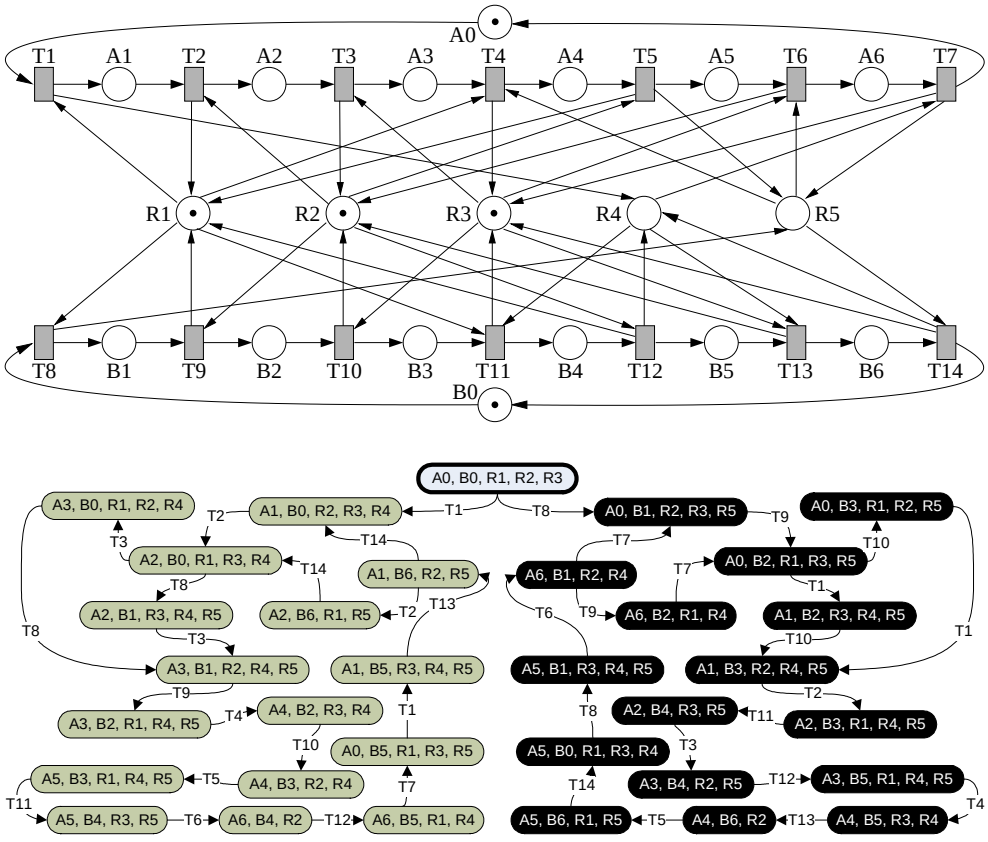



**Figure 2.9**: Live $PC^2R$ net system with a 0-acceptable initial marking for which no minimal t-semiflow is ever realisable

**Definition 2.30.** *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ be a $PC^2R$ net. An initial marking $\mathbf{m_0}$ is* acceptable at level 1 (1-acceptable) *for $\mathcal{N}$ iff $\|\mathbf{m_0}\| \setminus P_R = P_0$, and $\forall p \in P_S, r \in P_R : \mathbf{m_0}[r] \geq \mathbf{y_r}[p] - \mathbf{y_r}[p_0]$, where $p_0$ is the idle place of the process subnet which $p$ belongs to.*

The initial marking established by Definition 2.30 is greater, in general, than the minimum initial marking required by the definition of 0-acceptable initial marking (Definition 2.13), because in this new definition we diminish only with the initial marking of the idle place of the state machine of $p$. 1-acceptable initial markings can make much sense in the context of developing good practices in multithreaded programming. What is essentially requiring is that a single thread can be entirely executed in isolation, without requiring the intervention of any other process as far as the allocation of resources refers.

Observe that, once again, the definition of this kind of initial markings collapses with the definition of acceptable initial marking provided for subclasses such as $S^3PR$ or $S^4PR$, since no idle place belongs to any p-semiflow of resources in those cases.

**Theorem 2.31.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a $PC^2R$ net system with a 1-acceptable initial marking. Then every (minimal) t-semiflow is eventually realisable.*

*Proof.* It will be proved that a single token can be extracted from any idle place at $\mathbf{m_0}$ and be freely moved in isolation through its corresponding iterative state machine. Let $M_1$ be the subset of reachable markings such that one and only one process place is (mono-)marked, i.e., $M_1 = \{\mathbf{m} \in RS(\mathcal{N}, \mathbf{m_0}) \mid \exists |p \in P_S : \mathbf{m}[p] = 1, \|\mathbf{m}\| \cap P_S = \{p\}\}$.

First, every $t \in P_0{}^\bullet$ is enabled at $\mathbf{m_0}$ since ${}^\bullet t \subseteq P_0 \cup P_R$ and, by the definition of 1-acceptable initial marking, $P_0 \subset \|\mathbf{m_0}\|$ and $\forall r \in P_R : \mathbf{m_0}[r] \geq \mathbf{y_r}[q] - \mathbf{y_r}[p_0]$, where $q = t^\bullet \cap P_S$ and $p_0$ is the idle place of the process subnet which $q$ belongs to. Since $\mathbf{Pre}[r,t] = \max(0, \mathbf{y_r}[q] - \mathbf{y_r}[p_0])$ and $\mathbf{m_0}[r] \geq 0$ then $\mathbf{m_0}[r] \geq \mathbf{Pre}[r,t]$. By firing $t$ a marking of $M_1$ is reached.

Without loss of generality, let $\mathbf{m} \in M_1$. It will be proved that every transition $t \in (\|\mathbf{m}\| \cap P_S)^\bullet$ is enabled. Let $\{p\} = {}^\bullet t \cap P_S$, and $p_0$ be the idle place of the process subnet which $p$ belongs to. For every $r \in P_R$ the following invariant holds: $\mathbf{y_r} \cdot \mathbf{m} = \mathbf{y_r} \cdot \mathbf{m_0}$. This can be rewritten as follows: $\mathbf{y_r}[r] \cdot \mathbf{m}[r] + \mathbf{y_r}[p_0] \cdot \mathbf{m}[p_0] + \sum_{s \in P_0 \setminus \{p_0\}} (\mathbf{y_r}[s] \cdot \mathbf{m}[s]) + \mathbf{y_r}[p] \cdot \mathbf{m}[p] = \mathbf{y_r}[r] \cdot \mathbf{m_0}[r] + \mathbf{y_r}[p_0] \cdot \mathbf{m_0}[p_0] + \sum_{s \in P_0 \setminus \{p_0\}} (\mathbf{y_r}[s] \cdot \mathbf{m_0}[s])$. Since $\mathbf{m}[p] = 1$, $\mathbf{m}[p_0] = \mathbf{m_0}[p_0] - 1$, $\mathbf{y_r}[r] = 1$ and $\forall s \in P_0 \setminus \{p_0\} : \mathbf{m}[s] = \mathbf{m_0}[s]$ then it is derived: $\mathbf{m}[r] - \mathbf{y_r}[p_0] + \mathbf{y_r}[p] = \mathbf{m_0}[r]$. And, finally, $\mathbf{m}[r] = \mathbf{m_0}[r] + \mathbf{y_r}[p_0] - \mathbf{y_r}[p]$. It must be proved that $\mathbf{m}[r] \geq \mathbf{Pre}[r,t]$. Now two cases are distinguished:

- If $t^\bullet \cap P_S = \emptyset$ then $\mathbf{Pre}[r,t] = \max(0, \mathbf{y_r}[p_0] - \mathbf{y_r}[p])$. Since $\mathbf{m}[r] \geq 0$ then it only must be proved that $\mathbf{m}[r] \geq \mathbf{y_r}[p_0] - \mathbf{y_r}[p]$. Since $\mathbf{m}[r] = \mathbf{m_0}[r] + \mathbf{y_r}[p_0] - \mathbf{y_r}[p]$ and $\mathbf{m_0}[r] \geq 0$ then $\forall r \in P_R : \mathbf{m}[r] \geq \mathbf{Pre}[r,t]$ and $\mathbf{m} \xrightarrow{t} \mathbf{m_0}$.
- Otherwise, $\{q\} = t^\bullet \cap P_S$ and $\mathbf{Pre}[r,t] = \max(\mathbf{y_r}[q] - \mathbf{y_r}[p], 0)$. Since $\mathbf{m}[r] \geq 0$ then it only must be proved that $\mathbf{m}[r] \geq \mathbf{y_r}[q] - \mathbf{y_r}[p]$. By Definition 2.30, $\mathbf{m_0}[r] \geq \mathbf{y_r}[q] - \mathbf{y_r}[p_0]$ Then $\mathbf{m}[r] = \mathbf{m_0}[r] + \mathbf{y_r}[p_0] - \mathbf{y_r}[p] \geq \mathbf{y_r}[q] - \mathbf{y_r}[p_0] + \mathbf{y_r}[p_0] - \mathbf{y_r}[p] = \mathbf{y_r}[q] - \mathbf{y_r}[p]$. Thus, $\forall r \in P_R : \mathbf{m}[r] \geq \mathbf{Pre}[r,t]$; i.e. $\mathbf{m} \xrightarrow{t} \mathbf{m'}$, $\mathbf{m'} \in M_1$.

It has been proven that an isolated token can be carried from $\mathbf{m_0}[P_0]$ to any arbitrary $p \in P_S$. If $p$ belongs to a circuit, we can take that token and make it travel around the circuit. Since every t-semiflow corresponds to a circuit in a state machine (as proven for the dual case of circuits of marked graphs and p-semiflows [Mur89]), and $PC^2R$ nets are consistent by Lemma 2.22, the theorem holds. □

A marking which is 1-acceptable (Definition 2.30) is also 0-acceptable (Definition 2.13), although the opposite is not true in general, as proved by Fig. 2.9. Both

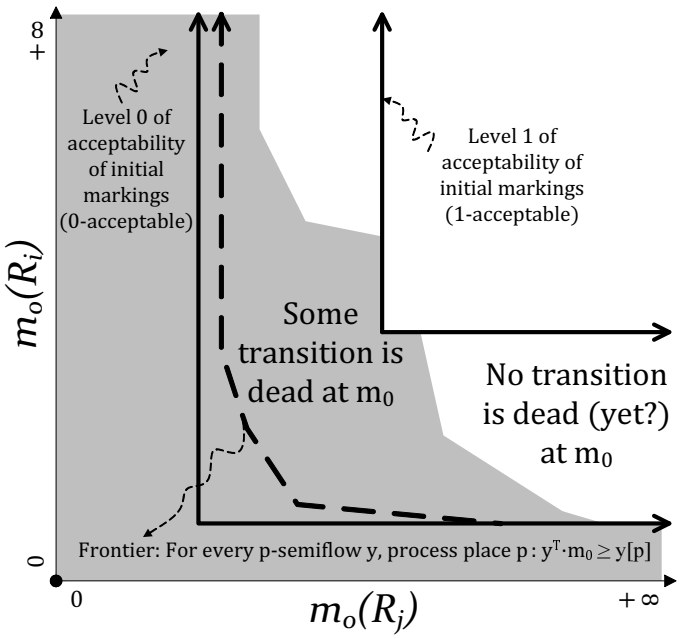



**Figure 2.10**: Schematic of the 0-1 zone of acceptable initial markings for general $PC^2R$ nets. It is assumed that $\|\mathbf{m_0}\| \setminus P_R = P_0$ and that $\mathbf{m_0}[P_0 \cup P_S]$ remains fixed

frontiers (that of 0-acceptable initial markings, and that of 1-acceptable ones) collapse into the definition of acceptable initial markings for the $S^5PR$ and lower subclasses. Unfortunately, for general $PC^2R$ nets out of those subclasses, there exists a 'dark region' in which it may be very difficult to determine if there exist dead transitions: we name this the *0-1 zone*. Figure 2.10 depicts how such 0-1 zone might look like.

At this 0-1 zone, it is worth noting the monotonicity (with respect to the marking of the resource places) of properties such as the absence of transitions which are already dead at $m_0$ or the eventual firability of a t-semiflow. This is due to the fact that increasing the initial marking of a resource place preserves every firing sequence, although some new firing sequences may be added. However, liveness is not monotonic there. Indeed, there may exist live systems in the 0-1 zone, but increasing the marking of resource places can make them non-live due to the addition of firing sequences which lead to deadlocks, while still staying at the 0-1 zone. The net in Fig. 2.11 illustrates this. As depicted, the net system is live. But if the initial marking of R2 is increased from 1 to 2 tokens, then a new reachable marking appears which is a deadlock ([A2, B2]).

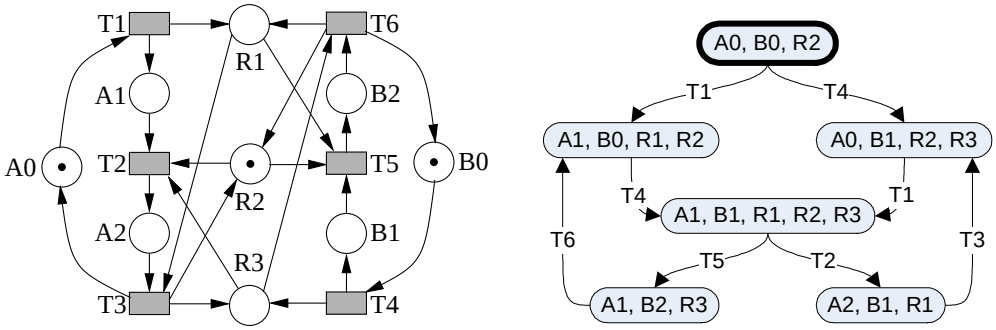



**Figure 2.11**: Live PC$^2$R net system with a 0-acceptable initial marking. Increasing the initial marking of R2 makes the system non-live while staying at the 0-1 zone

### 2.4.2 Liveness characterisation and siphons

Traditionally, empty or insufficiently marked siphons have been a fruitful structural element for characterising non-live RASs. The more general the net class, however, the more complex the siphon-based characterisation is. The following results can be easily obtained from previously published works. The originality here is to point out the strict conditions that the siphons must fulfil.

**Theorem 2.32.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an $S^3PR$ net system with an acceptable initial marking. $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live iff $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ and a minimal siphon $D : \mathbf{m}[D] = \mathbf{0}$.*

*Proof.* It has been proved that $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live iff $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ and an empty siphon $D$ at $\mathbf{m}$, i.e. $\mathbf{m}[D] = \mathbf{0}$ [ECM95]. Hence, the sufficient part is straightforward. Now suppose that the empty siphon $D$ is not minimal. Then there must exist a minimal siphon $D' \subset D$. Since $\mathbf{m}[D'] = \mathbf{0}$, an empty minimal siphon exists. □

For instance, the S$^3$PR net system in Fig. 2.12 is non-live with $\mathrm{K}_0 = \mathrm{K}_1 = 1$, $\mathrm{K}_3 = 2$. From this acceptable initial marking, the marking $[\mathrm{A4}, \mathrm{B4}, \mathrm{R2}, \mathrm{R3}^2]$ can be reached by firing $\sigma =$ TB1 TA1 TB2 TA2 TB3 TA3 TB4 TA4. This firing sequence empties the siphon $\{\mathrm{A1}, \mathrm{B1}, \mathrm{A5}, \mathrm{B5}, \mathrm{R1}, \mathrm{R4}\}$.

However, this characterisation is sufficient, but not necessary, in general, for S$^4$PR net systems. Hence, the concept of *empty siphon* had to be generalised. Note that the following theorem was already stated in Sect. 1.4 (see Definition 1.3 plus Theorem 1.4), although it is reminded here for the sake of self-containment of the section.

**Theorem 2.33.** *[TGVCE05] Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an $S^4PR$ net system with an acceptable initial marking. $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live iff $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ and a siphon $D$ such that:* i) *There exists at least one $\mathbf{m}$-process-enabled transition;* ii) *Every $\mathbf{m}$-process-enabled*

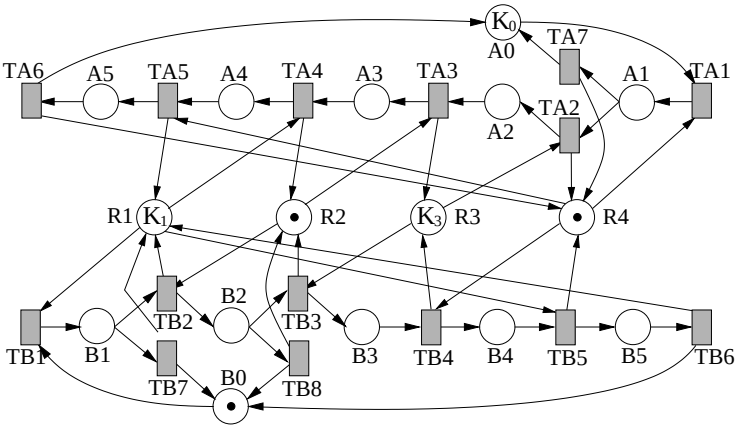



**Figure 2.12**: $S^3PR$ net which is non-live iff $(K_0 \geq K_1,\ K_3 \geq 2) \vee (K_0 \cdot K_1 \cdot K_3 = 0)$. Note that $\mathbf{m_0}$ is an acceptable initial marking iff $(K_0 \cdot K_1 \cdot K_3 \neq 0)$

*transition is* $\mathbf{m}$*-resource-disabled by resource places in* $D$*;* iii) *Process places in* $D$ *are empty at* $\mathbf{m}$.

Such a siphon $D$ is said to be insufficiently marked at $\mathbf{m}$. In Theorems 2.32 and 2.33, the siphon captures the concept of circular wait, revealing it from the underlying net structure. In contrast to the $S^3PR$ class, it is worth noting the following fact about *minimal* siphons in $S^4PR$ net systems, which emerges because of their minimal p-semiflows not being strictly binary.

**Property 2.34.** *There exists an $S^4PR$ net system with an acceptable initial marking which is non-live but every siphon pointing out the non-liveness is non-minimal, i.e., minimal siphons are non-sufficient to characterise non-liveness.*

For instance, the $S^4PR$ net system in Fig. 1.4 is non-live, but there is no minimal siphon containing both resource places R1 and R2. Note that the siphon $D = \{R1, R2, A3, B2\}$ becomes insufficiently marked at $\mathbf{m}$, where $\mathbf{m} \equiv [A1, B1, R1, R2]$, but it contains the minimal siphon $D' = \{R2, A3, B2\}$. $D'$ is not insufficiently marked for any reachable marking. The problem is that the unique resource place of $D'$ (i.e., R2) is not disabling every $\mathbf{m}$-process-enabled transition. Thus the resource place R1 is needed to decide that all $\mathbf{m}$-process-enabled transitions are $\mathbf{m}$-resource-disabled. Another interesting peculiarity of this net system is that no siphon is ever emptied.

Thus non-minimal siphons must be considered in order to deal with deadlocks in systems more complex than $S^3PR$.

On the other hand, insufficiently marked siphons (even considering those non-minimal) are not enough for characterising liveness for more complex systems such as

$S^5PR$ models. This means that siphon-based control techniques for RASs do not work in general for concurrent software, even in the 'good' case in which every `wait`-like operation precedes its complementary `signal`-like operation.

**Property 2.35.** *There exists an $S^5PR$ net system with an acceptable initial marking $\langle \mathcal{N}, \mathbf{m_0} \rangle$ which is non-live but insufficiently marked siphons do not point out non-liveness (dead markings).*

The $S^5PR$ net system in Fig. 2.13 evidences the claim stated above. The figure depicts a non-live system with three possibly bad minimal siphons. These siphons are $D_1 = \{A2, A3, A4, A5, A6, B2, B4, B5, B6, FORK2, BOWL\}$, $D_2 = \{A2, A4, A5, A6, B2, B3, B4, B5, B6, FORK1, BOWL\}$ and $D_3 = \{A2, A4, A5, A6, B2, B4, B5, B6, FORK1, FORK2, BOWL\}$. Besides, every transition in the set $\Omega = \{TA2, TA3, TA4, TA5, TB2, TB3, TB4, TB5\}$ is an output transition of $D_1$, $D_2$ and $D_3$. After firing transitions TA1 and TB1 starting from $\mathbf{m_0}$, the state [A1, B1, BOWL] is reached. This marking belongs to a livelock with other six markings. The reader can check that (i) there exists a firable transition in $\Omega$ for every marking in the livelock, (ii) the rest of transitions cannot be fired anymore (the net is non-live), and (iii) in any case there is no insufficiently marked siphon.

Revisiting Example 2.1 and its associated Algorithm 2.1, it is not difficult to see that, if every philosopher enters the room, sits down and picks up the fork on the left of himself, the philosophers will be trapped in a livelock. Any philosopher can eventually take the bowl of spaghetti and heat it up in the microwave. This pattern can be repeated infinitely often, but it is completely useless, since no philosopher will ever be able to have dinner.

This behaviour is obviously reflected in the corresponding net representation at Fig. 2.2. Let us construct a firing sequence $\sigma$ containing only the first transition of each state machine (i.e., the output transition of its idle place). The firing order of these transitions is irrelevant. Now let us fire such a sequence, and the net falls in a livelock. The internal cycles are still firable in isolation, but no idle place can ever be marked again. Unfortunately, the net has several bad siphons, but none of them is empty or insufficiently marked in the livelock. In other words, for every reachable marking in the livelock, there exist output transitions of the siphons which are firable. As a result, the siphon-based non-liveness characterisation for earlier net classes (such as $S^4PR$ [TGVCE05]) is not sufficient in the new framework.

### 2.4.3 Deadlock-freeness, liveness, reversibility and livelocks

By carefully observing the net in Fig. 2.13, it might seem that the difficulty in finding a liveness characterisation for $PC^2R$ net systems lies in the appearance of certain types of livelocks. In general, livelocks with dead transitions are not a new phenomenon in

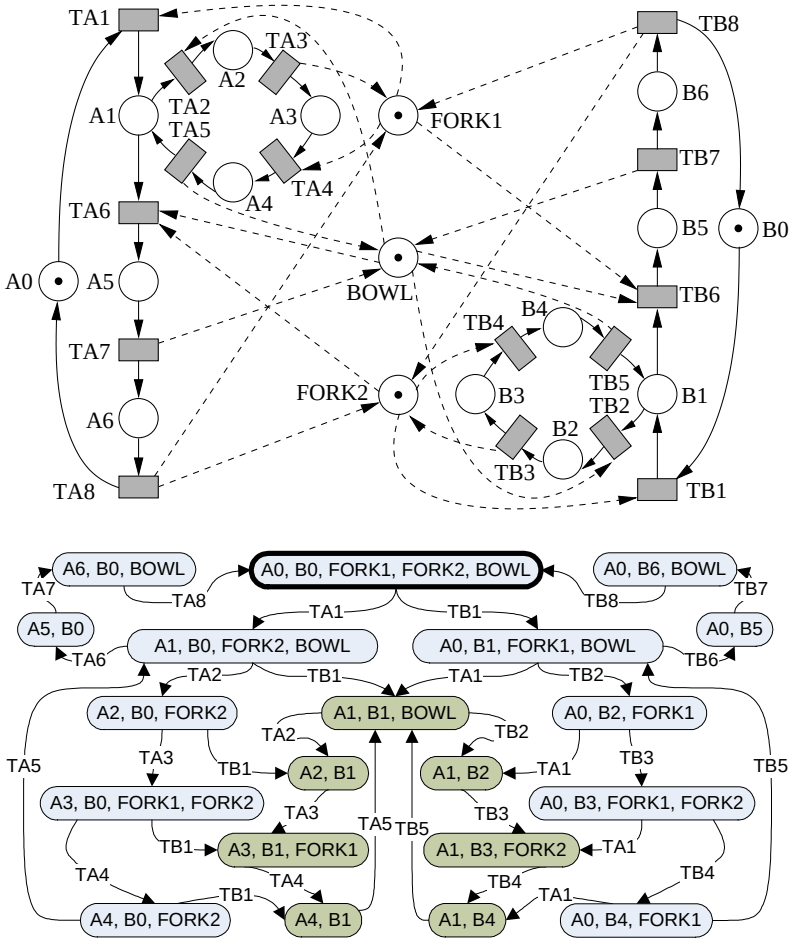



**Figure 2.13**: Non-live S$^5$PR net system modelling two postmodern dining philosophers

the context of Petri net models for RASs. Figure 2.14 shows that, even for L-S$^3$PR nets, deadlock-freeness does not imply liveness.

**Property 2.36.** *There exists a L-S$^3$PR net system with an acceptable initial marking such that it is deadlock-free but not live.*

This L-S$^3$PR net system has no deadlock but two reachable livelocks:

$$\text{Livelock1} \equiv \{[\text{A0}, \text{B2}, \text{C0}, \text{D1}, \text{R1}], [\text{A1}, \text{B2}, \text{C0}, \text{D1}]\}$$

$$\text{Livelock2} \equiv \{[\text{A0}, \text{B1}, \text{C0}, \text{D2}, \text{R3}], [\text{A0}, \text{B1}, \text{C1}, \text{D2}]\}$$

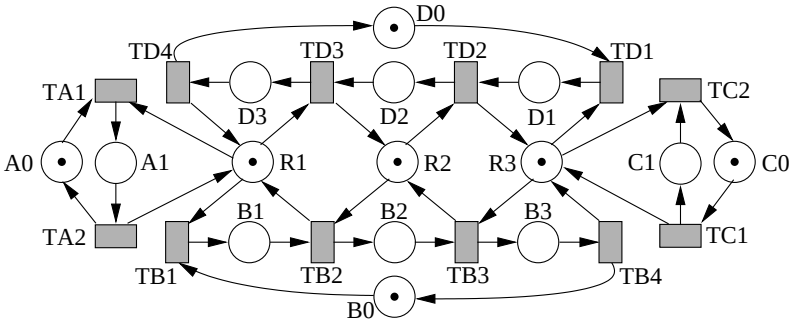



**Figure 2.14**: Non-live but deadlock-free L-S$^3$PR net system

Nevertheless, these livelocks are captured by insufficiently marked siphons. Unfortunately, this no longer holds for some kind of livelocks in S$^5$PR or more complex systems. Indeed, PC$^2$R nets feature some complex properties which complicate the finding of a liveness characterisation.

Another relevant property for studying liveness is its monotonicity. In spite of the seeming simplicity of S$^3$PR nets, the following negative result regarding liveness monotonicity applies:

**Property 2.37.** *There exists an $S^3PR$ net such that liveness is not monotonic, either with respect to the marking of the idle/process places, or that of the resource places, i.e., liveness is not always preserved when those are increased.*

The net depicted in Fig. 2.12 illustrates this fact:

- With respect to $P_R$: The system in Fig. 2.12 is live with $K_0 = K_1 = K_3 = 1$ and non-live with $K_0 = K_1 = 1$, $K_3 = 2$ (however, it becomes live again if the marking of R1, R2 and R4 is increased enough so as to make every resource place an implicit place).
- With respect to $P_0$: The system in Fig. 2.12 is live with $K_0 = 1$, $K_1 = K_3 = 2$ and non-live with $K_0 = K_1 = K_3 = 2$.

Note that liveness is monotonic with respect to the marking of the resource places for every net belonging to the L-S$^3$PR class [GV99]. But, from S$^3$PR nets upwards, there is a discontinuity zone (i.e., a range of initial markings where the property is fluctuating) between the point where the resource places are empty enough so that every transition is dead (also held for lower markings), and the point where every resource place is implicit (liveness is preserved if their marking is increased). Markings within these bounds fluctuate between liveness and non-liveness. The location of those points also depends on the marking of the idle/process places: the more tokens in them, the farther the saturation point (i.e., the upper bound).

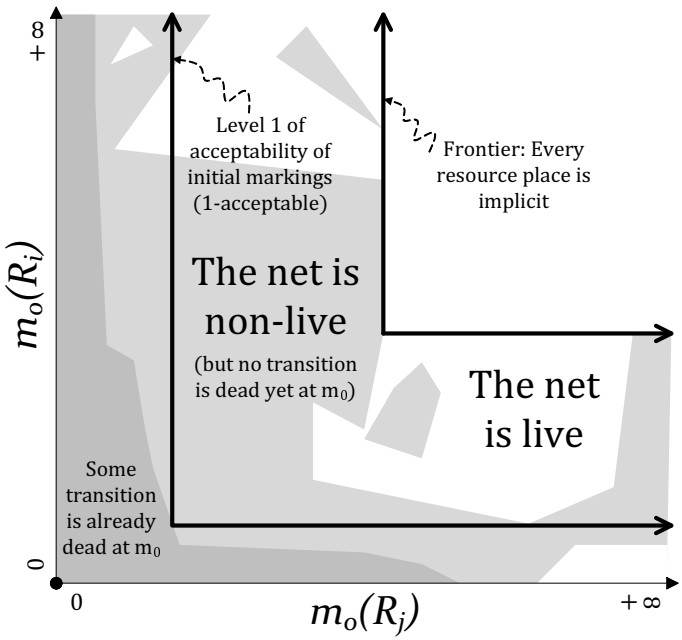



**Figure 2.15**: Schematic of the liveness discontinuity zone for marked $PC^2R$ nets. It is assumed that $\|\mathbf{m_0}\| \setminus P_R = P_0$ and that $\mathbf{m_0}[P_0 \cup P_S]$ remains fixed. Note that there may exist $\mathbf{m_0}$ making the system non-live at the 0-1 zone (i.e., below the '1-acceptable' frontier)

Note that, as discussed in Subsection 2.4.1, for $PC^2R$ nets liveness is also neither monotonic for those markings in the 0-1 zone (more exactly, above the frontier of markings which separates those systems with dead transitions and those without). Figure 2.15 depicts this.

Nevertheless, an interesting property of $S^4PR$ net systems is that liveness is a necessary and sufficient condition for reversibility. This, along with the fact that the idle place does not belong to any p-semiflow $\mathbf{y_r}$, is a powerful feature. If every token in a process net can be moved to the idle place, then the net is not dead (yet).

**Theorem 2.38.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an $S^4PR$ net system with an acceptable initial marking. $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is live iff $\mathbf{m_0}$ is a home state (i.e., the system is reversible).*

*Proof.* $\Longrightarrow$) Let us suppose that $\mathbf{m_0}$ is not a home state, i.e. $\exists \mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that $\mathbf{m_0} \notin \mathrm{RS}(\mathcal{N}, \mathbf{m}')$. Let $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m}')$ obtained by moving forward all the active processes (firing transitions $T \setminus P_0{}^\bullet$) until no process enabled transition can be fired. Since $\mathbf{m_0} \notin \mathrm{RS}(\mathcal{N}, \mathbf{m}')$, $\mathbf{m} \neq \mathbf{m_0}$, and the set of $\mathbf{m}$-process-enabled transitions is non-empty, and each one of these transitions is $\mathbf{m}$-resource-disabled. Hence, by Theorem 2.33, $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live.

$\Longleftarrow$) Let $\mathbf{m}$ be a reachable marking from $\mathbf{m_0}$, $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, and $t$ a transition of the net. It will be proved that there exists a successor marking of $\mathbf{m}$, $\mathbf{m}'$, from which $t$ is firable. Since $\mathbf{m_0}$ is a home state, there exists a firable sequence from $\mathbf{m}$ such that $\mathbf{m} \xrightarrow{\sigma} \mathbf{m_0}$. Taking into account that every t-semiflow can be fired in isolation from an acceptable initial marking of an $\mathrm{S^4PR}$ net [Tri03], then there exists a sequence $\sigma'$ whose characteristic vector $\boldsymbol{\sigma'}$ is equal to the t-semiflow containing $t$ (this t-semiflow exists because the net is consistent by Lemma 2.22). Let $\sigma' = \sigma'_1\, t\, \sigma'_2$ be a decomposition of the firing sequence $\sigma'$ to point out the first firing of $t$. Therefore $\sigma\, \sigma'_1$ is firable leading to a marking $\mathbf{m}'$ from which $t$ is firable. Because $\mathbf{m}$ and $t$ have been selected without constraints, the net is live.

□

However, Theorem 2.38 is false in general for $\mathrm{S^5PR}$ nets. In fact, the directedness property [BV84] does not even hold. This implies that an $\mathrm{S^5PR}$ net system may not have a home state, even being live.

**Property 2.39.** *There exists an $S^5PR$ net system with an acceptable initial marking $\langle \mathcal{N}, \mathbf{m_0} \rangle$ such that the system is live but there is no home state.*

The net system in Fig. 2.16 is a $\mathrm{S^5PR}$ which has no home state in spite of being live. It is worth noting that this net is ordinary. The reachability graph is illustrated in Fig. 2.17.

Having said that, $\mathrm{S^5PR}$ nets still retain an interesting property: its minimal t-semiflows are eventually realisable from an acceptable initial marking.

**Theorem 2.40.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an $S^5PR$ net system with an acceptable initial marking. For every (minimal) t-semiflow $\mathbf{x}$, there exists a reachable marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that $\mathbf{x}$ is realisable from $\mathbf{m}$, i.e. $\exists \sigma$ such that $\mathbf{m} \xrightarrow{\sigma}$, $\boldsymbol{\sigma} = \mathbf{x}$.*

*Proof.* This is straightforward from Theorem 2.31, since an acceptable initial marking for an $\mathrm{S^5PR}$ net is also a 1-acceptable initial marking (both definitions collapse for this subclass).

□

However, for $\mathrm{PC^2R}$ net systems there may not exist minimal t-semiflows being eventually realisable; even for live systems.

**Property 2.41.** *There exists a $PC^2R$ net system with a 0-acceptable initial marking $\langle \mathcal{N}, \mathbf{m_0} \rangle$ such that the system is live and there exists a minimal t-semiflow $\mathbf{x}$ such that $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, $\nexists \sigma$ such that $\mathbf{m} \xrightarrow{\sigma}$ and $\boldsymbol{\sigma} = \mathbf{x}$, i.e. $\mathbf{x}$ is not realisable from any $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$.*

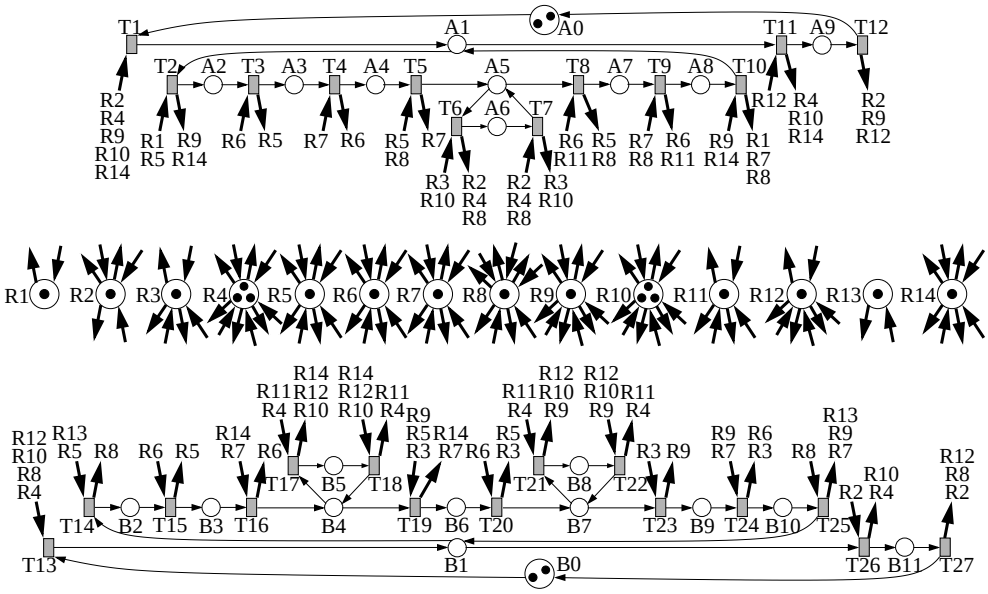



**Figure 2.16**: Live $S^5PR$ net system which has no home state. The arcs from/to $P_R$ are omitted for clarity. Instead, the set of input and output resource places are listed next to each transition

The reader can check that the $PC^2R$ net system in Fig. 2.9 has no home state in spite of being live. Depending on which transition is fired first (either T1 or T8) a different livelock is reached. Besides, for every reachable marking, there is no minimal t-semiflow such that it is realisable, i.e. firable in isolation. Instead, both state machines need each other to progress in an interleaved way from the very beginning.

T-semiflow realisability is, however, guaranteed when the initial marking is a 1-acceptable initial marking, by Theorem 2.31. Obviously, the initial marking in the net of Fig. 2.9 does not hold this, in spite of being a 0-acceptable initial marking.

Counterintuitively, the impossibility of realising every t-semiflow in a live $PC^2R$ net system cannot be directly linked to non-reversibility. The net system in Fig. 2.9 has no home state. However, the net system in Fig. 2.18 is reversible, live, but no minimal t-semiflow is realisable.

In fact, these two properties (reversibility and t-semiflow realisability) are usually strongly linked to the property of liveness for many Petri net classes. Particularly, reversibility is powerful since its fulfilment implies that the net is live iff there are no dead transitions at the initial marking. This last property is often easy to check (although that statement is not true in general for $PC^2R$ net systems, as discussed in


**Figure 2.17**: Reachability graph of the net system in Fig. 2.16. The net has two livelocks. Markings are coloured depending on the strongly connected component they belong to. The initial marking is stressed with a thicker border around it

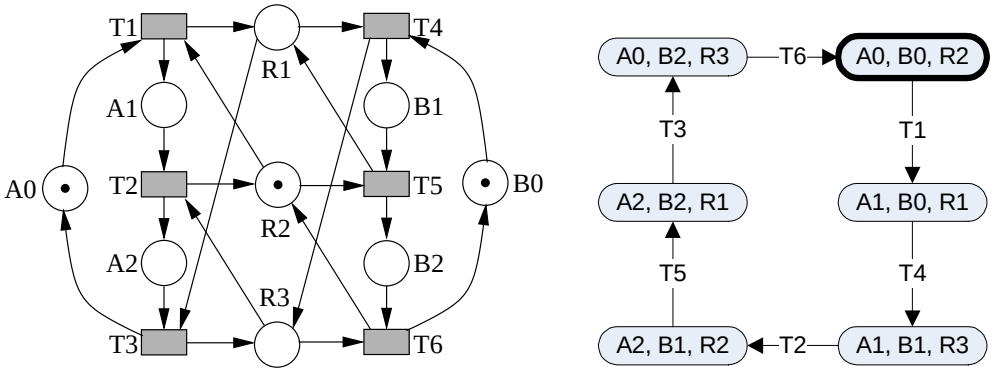



**Figure 2.18**: Live and reversible $PC^2R$ net system with a 0-acceptable initial marking such that no minimal t-semiflow is ever realisable

Subsection 2.4.1). Both properties (reversibility and t-semiflow realisability) together imply that the net is live, as the next theorem states.

**Theorem 2.42.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a $PC^2R$ net system with a 0-acceptable initial marking. If the net system is reversible and every (minimal) t-semiflow $\mathbf{x}$ is eventually realisable (i.e., there exist $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}), \sigma$ such that $\mathbf{m} \xrightarrow{\sigma}$, $\boldsymbol{\sigma} = \mathbf{x}$) then the net is live.*

*Proof.* By reduction to absurd: assume that the net is non-live. Then there exists $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ and a transition $t$ such that $t$ is dead at $\mathbf{m}$. But since the net is reversible, there exists a sequence $\sigma$ such that $\mathbf{m} \xrightarrow{\sigma} \mathbf{m_0}$. Since the net is consistent by Lemma 2.22, there exists a t-semiflow $\mathbf{x}$ that contains $t$, and $\mathbf{x}$ is eventually realisable, so $t$ is eventually firable from a successor marking of $\mathbf{m}$. But then $t$ cannot be dead, reaching a contradiction. □

Nevertheless, it is notorious that those properties are almost unlinked for general $PC^2R$ net systems, with the exception of the rule established by Theorem 2.42. Table 2.1 illustrates in a concise way the relation between these three properties (liveness, reversibility, and eventual firability of all t-semiflows) in the context of general $PC^2R$ net systems with 0-acceptable initial markings. The table highlights the fact that those properties are not totally independent because of $PC^2R$ nets being consistent, as proved by Theorem 2.42. It also reveals that the simpler the subclass, the less combinations of the three properties are possible (up to the point that liveness is a necessary and sufficient condition for reversibility for $S^4PR$ and simpler subclasses). Figure 2.21, which has not been introduced before, is used to complete the table.

| | | |
|---|---|---|
| $\boxed{\mathrm{L\,R\,T}}$<br>From L-S$^3$PR upwards<br>Fig. 2.12 with $K_0 = K_1 = K_3 = 1$ | $\boxed{\mathrm{L\,R\,\overline{T}}}$<br>PC$^2$R only<br>Fig. 2.18 | $\boxed{\mathrm{L\,\overline{R}\,T}}$<br>From S$^5$PR upwards<br>Figs. 2.16 and 2.17 |
| $\boxed{\mathrm{L\,\overline{R}\,\overline{T}}}$<br>PC$^2$R only<br>Fig. 2.9 | $\boxed{\mathrm{\overline{L}\,R\,T}}$<br>IMPOSSIBLE<br>Theorem 2.42 | $\boxed{\mathrm{\overline{L}\,R\,\overline{T}}}$<br>PC$^2$R only<br>Fig. 2.8 |
| $\boxed{\mathrm{\overline{L}\,\overline{R}\,T}}$<br>From L-S$^3$PR upwards<br>Fig. 2.12 with $K_0 = K_1 = 1$ and $K_3 = 2$ | | $\boxed{\mathrm{\overline{L}\,\overline{R}\,\overline{T}}}$<br>PC$^2$R only<br>Fig. 2.21 |

**Table 2.1**: Summary of possible combinations of liveness (L), reversibility (R), and eventual realisability of every t-semiflow (T) for PC$^2$R net systems with a *0-acceptable initial marking*. For each cell, the first line indicates which (sub)class such properties combination is possible from. The second line references a proof of such behaviour

Since, by Theorem 2.31, 1-acceptable initial markings grant the firability of every t-semiflow, a new corollary can be extracted from Theorem 2.42:

**Corollary 2.43.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a PC$^2$R net system with an 1-acceptable initial marking. By Theorem 2.31, if the net system is reversible then it is also live.*

Once more, it should be stressed that the reverse of Corollary 2.43 does not hold in general, as Figs. 2.16 and 2.17 prove. The same thing applies for the PC$^2$R net in Fig. 2.19. Note that this net only has one single elementary iteration block per process subnet (i.e., no internal loops). The reachability graph is depicted in Fig. 2.20.

Table 2.2 synthesises the relation between liveness and reversibility in general PC$^2$R net systems with 1-acceptable initial markings.

## 2.5 An insight on the problem of RASs with lender processes

Finally, it is worth bringing to attention the existence of yet another class of Petri net models for RASs, named SPQR [LGC06]. This class is interesting from the point of view of RAS analysis and synthesis in the domain of multithreaded programming. In the following subsections, this is made obvious since it is tightly related to the PC$^2$R class.

As a class on its own, SPQR nets feature an appealing syntactic simplicity and

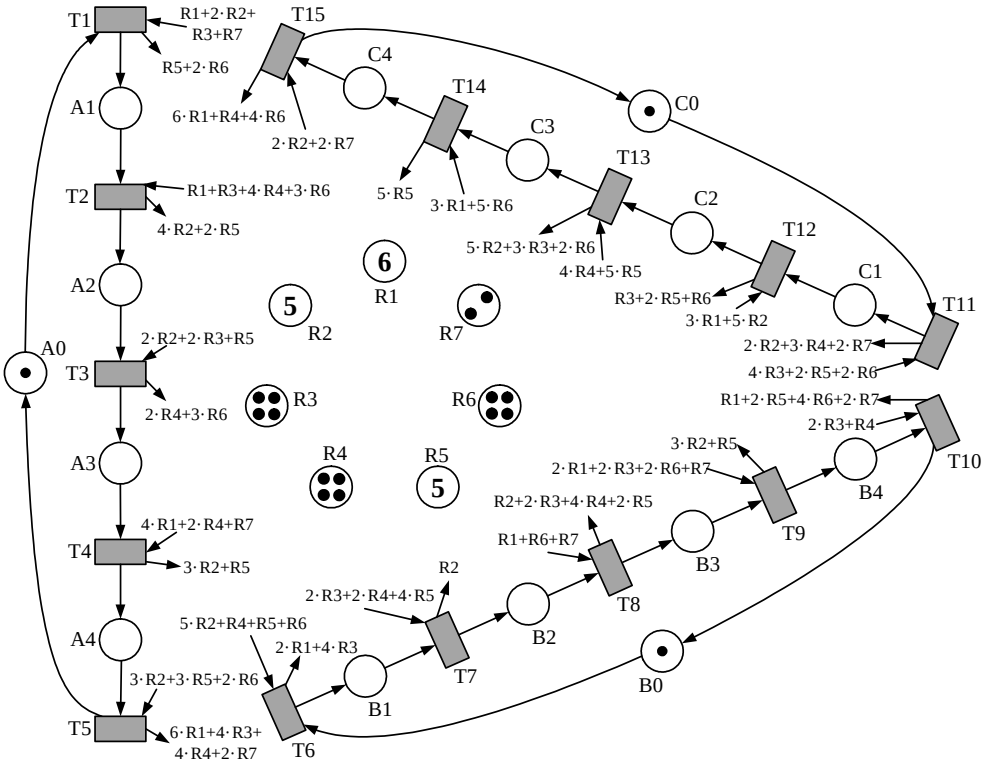



**Figure 2.19**: Live $PC^2R$ net system which has a 1-acceptable initial marking but it has no home state. The arcs from/to $P_R$ are omitted for clarity. Instead, the set of input and output resource places are listed next to each transition

| | |
|---|---|
| $\boxed{\mathrm{L\,R}}$<br>From L-$S^3$PR upwards<br>Fig. 2.12 with $K_0 = K_1 = K_3 = 1$ | $\boxed{\mathrm{L}\,\overline{\mathrm{R}}}$<br>From $S^5$PR upwards<br>Figs. 2.16 and 2.17 |
| $\boxed{\overline{\mathrm{L}}\,\mathrm{R}}$<br>IMPOSSIBLE<br>Corollary 2.43 | $\boxed{\overline{\mathrm{L}}\,\overline{\mathrm{R}}}$<br>From L-$S^3$PR upwards<br>Fig. 2.12 with $K_0 = K_1 = 1$ and $K_3 = 2$ |

**Table 2.2**: Update of Table 2.1 (i.e., possible combinations of liveness (L) and reversibility (R) for $PC^2R$ net systems) assuming an *1-acceptable initial marking*. It should be stressed that the eventual realisability of every t-semiflow is guaranteed in this context. Hence this table is smaller than that presented for 0-acceptable initial markings, i.e., Table 2.1

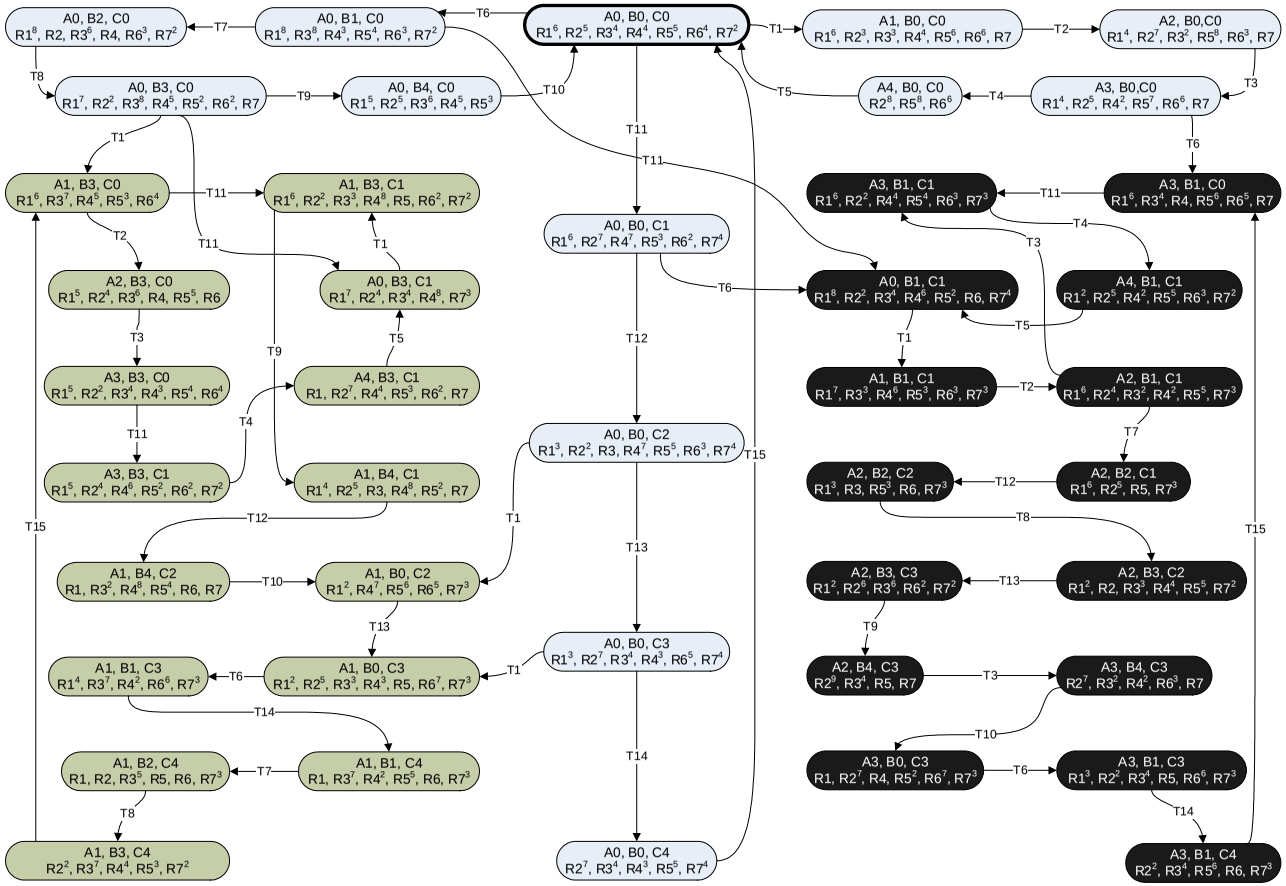



**Figure 2.20**: Reachability graph of the net system in Fig. 2.19. The net has two livelocks. Markings are coloured depending on the strongly connected component they belong to. The initial marking is stressed with a thicker border around it

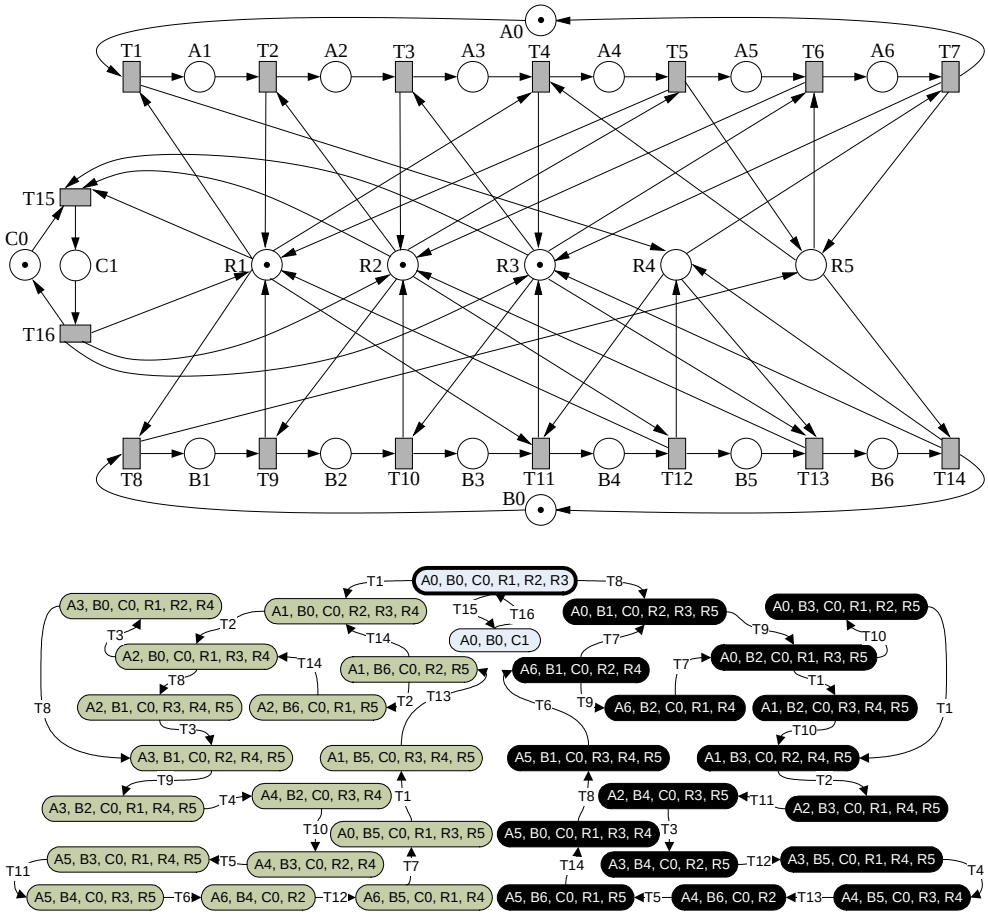



**Figure 2.21**: Non-live PC$^2$R net system with a 0-acceptable marking. The net is not reversible and no minimal t-semiflow is ever realisable. No transition is dead at $\mathbf{m_0}$, though.

expressive power though they are very challenging from an analytical point of view (even more than PC$^2$R nets). They can be roughly described as RAS nets in which the process subnets are acyclic and the processes can lend resources in any possible (conservative) manner. Every PC$^2$R net can be transformed into an SB SPQR net. Note that SB SPQR nets are conservative since they are consistent by construction, and consistency plus structurally boundedness implies conservativeness [Sil93]. In spite of the existence of a transformation rule, it must be remarked that PC$^2$R is *not* a strict subclass of SPQR nets.

The transformation of PC$^2$R nets into SB SPQR nets can be useful to understand the above phenomena from a structural point of view. Intuitively speaking, the con-

cept of *lender process* seems a simple yet powerful instrument which still remains to be fully explored. Yet SB SPQR net systems can present very complex behaviour, as discussed in Subsections 2.5.3 and 2.5.4.

### 2.5.1 Constructing systems with Plain Lender Processes

SPQR is a subclass of RASs resulting from a different abstraction process than in the case of $PC^2R$ nets. Historically, they were proposed previously to the definition of $PC^2R$ in an attempt to provide an answer to the need for a generalisation of the systems considered in the domain of FMSs. This generalisation proceeds from two different motivations: (1) The consideration of open systems, that is, process plans that have no limitations on the number of processes following these process plans. This generalisation was partially visited in the PhD thesis of Fernando García-Vallés. As a result, SPQR nets have no idle places and unboundedness is a property that can appear in these systems. (2) Resources can be used in more general ways. In the previous classes they are used in a conservative way in the sense that the contents of a set of process places, named holders of a resource $r$, and the tokens in the place corresponding to the resource remain constant and equal to the initial number of copies of resource $r$. The generalisation allows another kind of conservation laws than those induced by the p-semiflows of resources. Now, for example, it is possible for a resource $r$ to induce invariant relations like this:

$$\mathbf{m}[r] + \sum_{p \in \|\mathbf{y_r}\| : \mathbf{y_r}[p] \geq 0} \mathbf{y_r}[p] \cdot \mathbf{m}[p] = \mathbf{m_0}[r] - \sum_{p \in \|\mathbf{y_r}\| : \mathbf{y_r}[p] < 0} \mathbf{y_r}[p] \cdot \mathbf{m}[p]$$

.

Observe that in this case we have a marking invariant but different to those arising from $S^4PR$ nets. The left hand side of the invariant contains the resource place $r$ and the holder places $p \in \mathbf{y_r}$ such that $\mathbf{y_r}[p] \geq 0$: this is similar to the $S^4PR$ case. But now this quantity is not constant because of the marking of the places $p \in \{\|\mathbf{y_r}\| \mid \mathbf{y_r}[p] < 0\}$ appearing in the right hand side of the invariant. They are called lender places as opposed to the holder places. This means that, in order to maintain the invariant, if the marking of a lender place is increased then the marking of the resource place $r$ must also be increased. This is the reason why these places are called *lender* places of the resource $r$. This allows to model the starting of processes with allocated resources.

All these generalisations are very natural in the context of FMS or external logistic systems in order to model the RAS view of more general or complex systems. Nevertheless, in the context of multithreaded software these generalisations are not so natural. For example, structuring of code is a primary requirement that in the case of FMS has never been considered and even is not a common requirement at all.

We will see that these two subclasses are independent and the justification comes from the application domain but the bridges between the two subclasses give rise to interesting interpretations of many phenomena using the categories of other domain.

In fact, the SPQR class provides a consistent and very general framework for the study of the RAP in Sequential RASs. It generalises previous, well-known models for modelling the RAS view of FMSs. The assumption of the analytical results developed for the $S^nPR$ family is derived: these can be easily mapped into the new class. On the other hand, the expressive power of previous models is enriched by the addition of some new elements. Some of these elements are already introduced for the $PC^2R$ class. In particular, it is intended to address the following types of systems:

- Systems in which there exist nested internal circuits within the control flow of the processes. This is typical in, e.g., software systems (iterative processes), which are the subject of this thesis, but also in manufacturing systems with recirculating circuits, among others.
- Systems in which there are resources which are already allocated in the initial state.
- Open systems in which it is known the processes structure, but not the number of concurrent instances.
- Systems in which the number of resources is variable; despite the fact that processes use them in a conservative way (i.e., a process neither creates nor destroys resources in the system after completing its execution).

These enhancements can be captured by the new class definition, but they also raise unexpected properties and interesting questions regarding liveness analysis that will be studied in the following subsections.

Many of these enhancements are also considered in the definition of $PC^2R$ nets. However, there exist two significant differences. First, SPQR nets are focused to model open systems. This means that the process subnets lack an idle place, so the number of concurrent processes is only limited (if anything) by the number of available resources. And second, and even more importantly, the iteration blocks or circuits which are internal to the processes are not directly supported by the structure of the process subnets (as opposed to the usage of iterative state machines in the construction of $PC^2R$ nets). Instead, they are simulated by the introduction of *virtual resources*, as discussed in Subsection 2.5.5. As a result, SPQR nets are constructed around two orthogonal and very simple elements (acyclic state machines and resources), which seemingly allows us to focus on the very core of the liveness problem.

We first present SPQR nets in a compositional way, by introducing the kind of subnets which describes a process along with the resources it uses.

**Definition 2.44.** *A* Plain Lender Process (PLP) *is a connected generalised pure P/T net, $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$, where:*

1. *$P = P_i \cup P_{\mathrm{R}}$ where $P_i, P_{\mathrm{R}} \neq \emptyset$, $P_i \cap P_{\mathrm{R}} = \emptyset$.*
2. *The subnet generated by $P_i$, $T$ is a connected acyclic state machine.*
3. *For each $r \in P_{\mathrm{R}}$, exists a unique p-flow $\mathbf{y_r} \in \mathbb{Z}^{|P|}$ such that $\{r\} = \|\mathbf{y_r}\| \cap P_{\mathrm{R}}$, $\|\mathbf{y_r}\| \cap P_i \neq \emptyset$ and $\mathbf{y_r}[r] = 1$.*

Places in $P_i$ are called *process places*, while places in $P_{\mathrm{R}}$, *resource places*. Transitions with no input process place, i.e. $\{t \in T \mid {}^\bullet t \cap P_i = \emptyset\}$, are called *trigger transitions*. Transitions with no output process place are *drain transitions*.

An SPQR net is the result of merging a non-empty set of PLPs via fusion of their common resource places.

### 2.5.2 The SPQR class: Definition

An SPQR net is, in rough words, an $S^4$PR net [Tri03] without idle places, extended in a way such that processes can hold some resources in the initial state, or even when they are inactive. From a structural point of view, this means that every resource place of the net induces a unique p-flow $\mathbf{y_r}$, instead of a minimal p-semiflow.

Similarly to $S^4$PR nets, there exists an invariant relation which rules how the resources are used by the processes. Consequently, a process eventually destroys (returns) all the resources created (acquired) during its lifetime. In $S^4$PR nets, however, the idle place is an absolute minimum with relation to the resource usage state of the process. But an SPQR net may lack an absolute minimum, and this can severely complicate liveness analysis. In fact, due to the absence of the idle place, an SPQR net system can also be unbounded.

The following is the formal definition of the SPQR class:

**Definition 2.45.** *[LGC06] Let $I_{\mathcal{N}}$ be a finite set of indices. A* System of Processes Quarrelling over Resources (SPQR) net *is a connected generalised pure P/T net $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ where:*

1. *$P = P_{\mathrm{S}} \cup P_{\mathrm{R}}$ is a partition such that:*
   - *(a)* [process places] *$P_{\mathrm{S}} = \bigcup_{i \in I_{\mathcal{N}}} P_i$, where:*
     *$\forall i \in I_{\mathcal{N}} : P_i \neq \emptyset$ and $\forall i, j \in I_{\mathcal{N}}, i \neq j : P_i \cap P_j = \emptyset$;*
   - *(b)* [resource places] *$P_{\mathrm{R}} = \{r_1, ..., r_n\}, n > 0$.*
2. *$T = \bigcup_{i \in I_{\mathcal{N}}} T_i$, where $\forall i \in I_{\mathcal{N}} : T_i \neq \emptyset$, and $\forall i, j \in I_{\mathcal{N}}, i \neq j : T_i \cap T_j = \emptyset$.*

3. *For each $i \in I_{\mathcal{N}}$ the subnet generated by restricting $\mathcal{N}$ to $\langle P_i, T_i \rangle$ is a connected acyclic state machine. This is called the $i$-th* process subnet.

4. *For each $r \in P_{\mathrm{R}}$, there exists a unique p-flow $\mathbf{y_r} \in \mathbb{Z}^{|P|}$ such that $\{r\} = \|\mathbf{y_r}\| \cap P_{\mathrm{R}}$, $\|\mathbf{y_r}\| \cap P_{\mathrm{S}} \neq \emptyset$, and $\mathbf{y_r}[r] = 1$.*

The reader can easily check that the composition of a non-empty set of PLPs by fusion of the common resource places is always an SPQR net, assuming that the various sets $P_i$ ($T_i$) are disjoint. Equivalently, any SPQR net can be seen as the composition of a non-empty set of PLPs by fusion of some shared resource places.

The concept of *holder* and *lender* places will be frequently used in the following. A *holder place* is a process place in which processes use one or several instances of a type of resource $r$, i.e. $\{p \in P_i \mid \mathbf{y_r}[p] > 0\}$. As PLPs can have non-positive p-flows, it is necessary to introduce a new concept: *lender places*.

**Definition 2.46.** *[LGC06] Let $\mathcal{N}$ be a PLP, $p \in P_{\mathrm{S}}$ , and $r \in P_{\mathrm{R}}$:*

- *$p$ is a* holder place *of $r$ iff $\mathbf{y_r}[p] > 0$. The set of holders of $r$ is denoted $\mathcal{H}_r$.*
- *$p$ is a* lender place *of $r$, iff $\mathbf{y_r}[p] < 0$. The set of lenders of $r$ is denoted $\mathcal{L}_r$.*

The term *lender process* refers to a process subnet which contains at least one lender place of some resource. Conversely, the term *borrower process* refers to a process subnet which does not contain any lender place of any resource.

For historical reasons, a particular subclass of SPQR, called borrower SPQR (b-SPQR), is identified in the following:

**Definition 2.47.** *[LGC06] A* borrower SPQR (b-SPQR) net *is an SPQR net such that, for every $r \in P_{\mathrm{R}}$, the p-flow $\mathbf{y_r}$ (see point 4 in Definition 2.45) is a minimal p-semiflow, i.e., $\mathbf{y_r} \in \mathbb{N}^{|P|}$.*

In other words, a b-SPQR net is an SPQR net without lender processes. Note that any $S^4PR$ or $S^3PR$ net belongs to the b-SPQR class, taking its idle places as resource places. A significant subclass of b-SPQR is the subclass of Open L-$S^3PR$ [GV99].

### 2.5.3 The SPQR class: Some structural properties

In this subsection, we present a review of some structural properties of the SPQR class, with a special focus on properties which are relevant to liveness analysis. Some of these properties are yet unseen within the family of well-known Petri net models for analysing RASs. As a contrast to the $PC^2R$ class, for instance, it is revealed that SPQR nets are not, in general, Structurally Live (SL).

In the last part of the subsection, it is formally proved that the process subnets of b-SPQR nets hold that 'acquire' operations always precede 'release' operations.

### Conservativeness and consistency

First it will proved that SPQR nets are consistent:

**Proposition 2.48.** *SPQR nets are consistent, i.e.* $\exists \mathbf{x} > \mathbf{0}$ *s.t.* $\mathbf{C} \cdot \mathbf{x} = \mathbf{0}$.

*Proof.* Acyclic state machines are consistent. Therefore, $\exists \mathbf{x} > 0$ s.t. $\mathbf{C}[P_{\mathrm{S}}, T] \cdot \mathbf{x} = 0$. It will be proved now that $\mathbf{C}[P_{\mathrm{R}}, T] \cdot \mathbf{x} = 0$. For each $r \in P_{\mathrm{R}}$, $\mathbf{C}[r, t] = -\mathbf{y_r}[P_{\mathrm{S}}] \cdot \mathbf{C}[P_{\mathrm{S}}, T]$ by point 4 in Definition 2.45. Thus, for each $r \in P_{\mathrm{R}}$, $\mathbf{C}[r, t] \cdot \mathbf{x} = -\mathbf{y_r}[P_{\mathrm{S}}] \cdot \mathbf{C}[P_{\mathrm{S}}, T] \cdot \mathbf{x} = 0$. □

**Corollary 2.49.** *The set of t-semiflows of an SPQR net is equal to the union of t-semiflows of each process subnet (which are acyclic state machines).*

A well-known general result of Petri nets is that conservative nets are SB [SC88]. However, the inverse may not be true. In SPQR nets, nonetheless, conservativeness and structurally boundedness are equivalent. This is due to the fact that SPQRs nets are consistent by construction, and consistency plus SB implies conservativeness [Sil93]. Consequently, the acronym SB SPQR (or SB b-SPQR) denotes the subclass of nets in which every place of the net is covered by a p-semiflow.

### Structural Boundedness and Structural Liveness

It can be inferred from the above discussion that an SPQR net $\mathcal{N}$ is SB iff every process place is SB. A process place is SB only if it is holder of at least one resource place, but this condition is only necessary, not sufficient, e.g., in Fig. 2.22.c), place E is a holder place of R5, while place F is a holder place of R6. However, both places E and F are unbounded if R5 or R6 are initially marked (i.e., non-empty). Note that R5 and R6 are SB. This result differs from $\mathrm{S}^4$PR nets, in which every net of the class is conservative (and hence SB).

Another interesting difference is that resource places are no longer structurally implicit places. In the $\mathrm{S}^4$PR class, this property was derived from the fact that every place was covered by a p-semiflow $\mathbf{y_r}$ plus the existence of the *idle place*, which induced an additional p-semiflow which covered every process place. Here both conditions disappear.

As a consequence, SPQR nets are not, in general, SL; even despite they are always consistent. $\mathrm{S}^4$PR nets were SL because every resource place could be made implicit taking an initial marking higher enough. Doing so, the system became a set of isolated marked strongly connected state machines, hence being live. However, even some very simple SB b-SPQR nets are not structurally live, as the Open L-$\mathrm{S}^3$PR [GV99] in Fig. 2.22.b) proves. Regardless of the initial marking, this net can always be deadlocked by emptying places C2 and D2 by firing transitions T9 and T12 as many times as possible, and subsequently emptying R3 and R4 by repeatedly firing

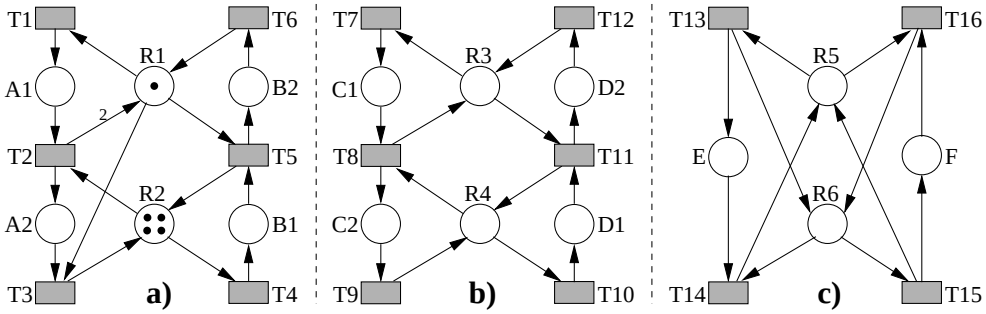



**Figure 2.22**: **a)** Non-live SB SPQR net system with one lender process on the left, and one borrower process on the right. **b)** Open L-S$^3$PR (therefore, SB b-SPQR net) which is not structurally live, i.e., it is non-live for every possible initial marking. **c)** SPQR net with two lender processes. The net is not SB (places E and F are unbounded iff $\mathbf{m_0}[\text{R5}] + \mathbf{m_0}[\text{R6}] \neq 0$)

transitions T7 and T10. This scenario raises interesting questions regarding liveness synthesis that, in some subclasses, are yet to be fully addressed.

### Structural Directedness

The Structural Directedness (SD) property [TS93] states that for every pair of potentially reachable markings of a live system there is always a common successor marking. SD is, indeed, a stronger, structural version of the directedness property [BV84], which will be studied among the behavioural properties, and holds for classes such as Equal Conflict (EQ) systems [TS93]. SD is very interesting from the standpoint of liveness analysis since it implies the absence of killing spurious solutions or, in other words, of live systems with potentially reachable markings being non-live.

It has been proven that SD is satisfied for the L-S$^3$PR class [GV99], but unfortunately it is not for any other known superclass, including the S$^3$PR class. The lack of this property hardens liveness analysis, due to the emergence of killing spurious solutions, and is obviously extensible to the more general SPQR class; although the reachability space of live SB SPQR net systems is not always directed either, as shown in Fig. 2.9 and discussed later.

### On the processes structure

Finally, it will be shown how b-SPQR nets differ from general SPQR nets in the way their processes are constructed: nets belonging to that subclass can be identified at a glance.

Since each process subnet is an acyclic state machine, a partial order can be established for its set of transitions. The operator $\preceq$ will serve the purpose.

**Definition 2.50.** *Let $\mathcal{N} = \langle P_S \cup P_R, T, \mathbf{C}\rangle$ be an SPQR net. We define $\preceq$ as a binary relation in $T$ such that $t_1 \preceq t_2$ ($t_1$* precedes *$t_2$) iff exists a directed path from $t_1$ to $t_2$ in a process subnet or, in other words, in the subnet generated by $P_S, T$.*

Further technical details on this operator are provided in a previous work of ours [LGC06]. For notational convenience, $t_1 \prec t_2$ denotes $t_1 \preceq t_2$ and $t_1 \neq t_2$. The following corollary establishes the relation between t-semiflows and the new precedence operator.

**Corollary 2.51.** *By Corollary 2.49, for every pair $t_1, t_2 \in T$ exists a minimal t-semiflow $\mathbf{x}$ such that $t_1, t_2 \in \|\mathbf{x}\|$ iff $t_1 \preceq t_2$ or $t_2 \preceq t_1$.*

The precedence operator will now prove very handy to show how resources are used in b-SPQR nets. In these nets, transitions that take tokens from $P_R$ always *precede* those that put tokens into $P_R$, in such a way that every resource instance has been borrowed before than released. In other words, resources are always used in an acquire-before-release basis. This is a corollary of the following proposition:

**Proposition 2.52.** *Let $\mathbf{x}$ be a minimal t-semiflow of an SPQR net with no lender places in the t-component induced by $\mathbf{x}$. Then, for every $u \in \|\mathbf{x}\|$, $\mathbf{C}[P_R, T] \cdot \mathbf{1}_\tau \leq \mathbf{0}$, where $\tau = \{t \in \|\mathbf{x}\| \mid t \preceq u\}$. In other words, the sum of weights of the incoming arcs from $P_R$ to the 'prefix' $\tau$, is above or equal to that of the outgoing arcs.*

*Proof.* Let $\bar{\tau} = \|\mathbf{x}\| \setminus \tau$, and let $P_x$ be the set of process places in the t-semiflow induced by $\mathbf{x}$, i.e. $P_x = ({}^\bullet\|\mathbf{x}\| \cup \|\mathbf{x}\|^\bullet) \cap P_S$.

By point 4 in Definition 2.45, for every $r \in P_R$, $\mathbf{C}[r, T] + \mathbf{y_r}[P_S] \cdot \mathbf{C}[P_S, T] = \mathbf{0}$. If both terms are multiplied by $\mathbf{1}_{\bar{\tau}}$, it is obtained: $\mathbf{C}[r, T] \cdot \mathbf{1}_{\bar{\tau}} + \mathbf{y_r}[P_x] \cdot \mathbf{C}[P_x, T] \cdot \mathbf{1}_{\bar{\tau}} = 0$ (note that $P_S$ has been replaced by $P_x$ due to the fact that $\mathbf{C}[p, \bar{\tau}] = \mathbf{0}$ for every $p \in P_S \setminus P_x$). Taking into account that, by Corollaries 2.49 and 2.51, the subnet generated by $P_x$ and $\|\mathbf{x}\|$ is a directed path of a process subnet (from a trigger to a drain transition), and $\bar{\tau}$ is the set of transitions of a suffix of that directed path, then $\mathbf{C}[P_x, T] \cdot \mathbf{1}_{\bar{\tau}} \leq \mathbf{0}$. Since there are no lender places in $P_x$, $\mathbf{y_r}[P_x] \geq \mathbf{0}$ and then $\mathbf{C}[P_R, T] \cdot \mathbf{1}_{\bar{\tau}} \geq \mathbf{0}$.

By Corollary 2.49, $\mathbf{C}[P_S, T] \cdot \mathbf{x} = \mathbf{0}$. Since $\mathbf{C} \cdot \mathbf{x} = \mathbf{0}$, then $\mathbf{C}[P_R, T] \cdot \mathbf{x} = \mathbf{0}$. Also by Corollary 2.49, $\mathbf{x} = \mathbf{1}_{\|\mathbf{x}\|} = \mathbf{1}_\tau + \mathbf{1}_{\bar{\tau}}$. Thus $\mathbf{C}[P_R, T] \cdot \mathbf{1}_\tau = -\mathbf{C}[P_R, T] \cdot \mathbf{1}_{\bar{\tau}} \leq \mathbf{0}$. □

**Corollary 2.53.** *b-SPQR nets do not have lender places. Thus, every 'prefix' $\tau$ of a minimal t-semiflow holds $\mathbf{C}[P_R, T] \cdot \mathbf{1}_\tau \leq \mathbf{0}$; therefore it can be said that resources are always used in an acquire-before-release basis.*

### 2.5.4 The SPQR class: Some behavioural properties

Having presented some basic structural properties of the class, we address on the following a review of relevant behavioural properties regarding liveness. Some of the

findings are shared with the $PC^2R$ class, although they are still somewhat surprising in a class of such syntactical simplicity (for instance, SPQR net systems do not hold the directedness property). The conclusion is that finding efficient liveness analysis methods for SPQR nets is presumably much harder than for previous classes such as the $S^4PR$ class.

**Liveness vs. deadlock-freeness**

One of the characteristic properties of the $S^nPR$ family (including the L-$S^3PR$ class) is that deadlock-freeness does not imply liveness [GV99]. In this sense, they are trickier than other well-known Petri net classes such as strongly connected free choice systems [Hil85], bounded strongly connected EQ systems [TS93] or CSS [LT79], where both properties are equivalent.

On the contrary, liveness is not even monotonic with respect to the initial marking (neither to the marking of the process places, nor that of the resource places), with the L-$S^3PR$ as the unique exception [GV99]. In fact, for $S^3PR$ and $S^4PR$ nets, there is a discontinuity zone between the point where the resource places are empty enough so that no transition is ever firable (all the lower markings imply a deadlock), and the point where every resource place is implicit (higher markings in them imply liveness). The markings within these bounds switch discontinuously between liveness and non-liveness. Of course, the location of those points also depends on the marking of the process places. However, SPQR and b-SPQR nets are not (in general) SL, not even being SB, and this implies that there may no longer exist an upper liveness region (e.g., the net in Fig. 2.22.b).

**Reversibility and directedness**

In an $S^4PR$ net system with an acceptable initial marking, reversibility is a necessary and sufficient condition for liveness [Rev03]. However, the class considered here (SPQR) is more general so, in particular, reversibility is not necessary for liveness. Figure 2.9 depicts an SB SPQR net system which is live but not reversible. Besides, reversibility is neither sufficient for liveness, provided that it is no longer required that all the t-components are firable in isolation from $\mathbf{m_0}$.

The directedness property [BV84] states that, for every pair of reachable markings in a live system, there is always a common successor marking. Although the directedness property obviously holds for the $S^4PR$ class (reversibility is a necessary and sufficient condition for liveness), it is not verified, in general, for the SPQR class, as once more Fig. 2.9 reflects. The reachability space of the net has two terminal strongly connected components, being the net live.

For bounded marked nets, the directedness property is equivalent to the existence of home states [BV84]. Hence, live $S^4PR$ net systems have home states; indeed,

every reachable marking is a home state, including the initial marking $\mathbf{m_0}$, since the net is reversible[2]. This is very useful for determining if the net is non-live, since the death of the system can be reduced to: "Is $\mathbf{m_0}$ unreachable from some reachable marking?". Even more, if it is reachable, a path that leads to $\mathbf{m_0}$ can be systematically constructed, and the length of this path is not higher than the size of the net, due to the structure of the $S^4PR$ class.

The "bad" news here is that, since the directness property is not held by the SPQR class, not even if the net is SB, it cannot be inferred (in general) that a home state will exist, whether the net is live or not. Again, Fig. 2.9 is a good example of this kind of behaviour. This is a severe problem for determining non-liveness in an efficient way.

### 2.5.5 Transformations and class relations

The SPQR class was defined as a general framework in which the philosophy and results on previous RAS subclasses could be deployed while providing some innovative elements for modelling much more complex systems [LGC06]. This includes all the necessary elements for modelling RASs in multithreaded software systems in similar terms to the definition of the $PC^2R$ class.

However, two important remarks are in place. First, the modelling power of general SPQR nets is high enough (unboundedness, etc.) so as to make liveness analysis a very difficult task in the general case. Therefore, common sense dictates that simpler subclasses must be identified in order to succeed in finding structural liveness characterisations. Second, SPQR nets are somewhat 'raw' models. Remarkably, their process subnets are acyclic state machines, while software threads can have internal iterations in which resources are allocated or released, as discussed in Sect. 2.3.

In fact, these models are not thought as the optimum target for a first abstraction of software systems into Petri net models. However, $PC^2R$ nets or simpler models can be transformed or even directly mapped into the framework of the SPQR class. Once a model has been ported into the transformed space, the appealing syntax simplicity of SPQR nets can be helpful from a theoretical point of view in order to explore whether the system is live or not, and why. Somehow, they can work very well as 'low-level' models. Obviously, this requires that an appropriate subclass has been defined so that the focus is on an workspace equivalent to that of the original class, not on that of a more general class of systems.

In this subsection we relate $PC^2R$ nets and other subclasses of the $S^nPR$ family with those subclasses of the SPQR framework, and describe precise net transformation rules to travel from one to each other.

[2]It is interesting to note that, in live L-$S^3PR$ net systems, a stronger condition holds: every potentially reachable marking is a home state [GV99].

**Lemma 2.54.** *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$ be an S⁴PR net [Tri03]. Then $\mathcal{N}' = \langle P_S \cup P'_R, T, \mathbf{C}\rangle$, where $P'_R = P_0 \cup P_R$, is an SB b-SPQR net.*

*Proof.* Let $\mathcal{N}_i$, $i \in I_{\mathcal{N}}$, be the $i$-th process subnet in $\mathcal{N}$ [Tri03]. The subnet generated by restricting $\mathcal{N}_i$ to $\langle P_i, T_i\rangle$ is a connected *acyclic* state machine, since by Definition 1.1, condition 3, every cycle contains the idle place $p_{0_i}$. Note that this idle place is not removed, but 'moved' to $P'_R$. Moreover, there exists a unique p-semiflow $\mathbf{y_{0_i}}$, $\|\mathbf{y_{0_i}}\| = P_i \cup \{p_{0_i}\}$. Since $p_{0_i} \in P'_R$ and $P_i \cap P'_R = \emptyset$, $\mathbf{y_{0_i}}$ holds the condition 4 in Definition 2.45. Hence every process subnet in $\mathcal{N}$ is a borrower process in $\mathcal{N}'$, and $\mathcal{N}'$ is a b-SPQR net. Since $\mathcal{N}$ is SB by construction, then $\mathcal{N}'$ is an SB b-SPQR net. □

However, the inverse of Proposition 2.54 is not strictly true: not every SB b-SPQR net is an S⁴PR net. However, any SB b-SPQR net can be easily transformed into an S⁴PR net by introducing a structural implicit place per process subnet connecting its drain transitions (transitions without output process places) with its trigger transitions (transitions without input process places). Obviously, the initial marking of this place (the new idle place) must be made high enough so as to make the place implicit. Fortunately, this can be accomplished thanks to the fact that the original net is SB.

Obviously, S³PR [ECM95] and L-S³PR [EGVC98] nets can be also redefined as SB b-SPQRs nets, since they are children of the S⁴PR class. Similar redefinitions in terms of the SPQR class can be applied to any previously defined Petri net model for Sequential RASs, except for S*PR nets, since the SPQR class does not directly deal with internal cycles. Note that all the above transformations are presented for completion (i.e., as a proof that previous subclasses can be easily mapped into the new framework), although they have no interest from a practical point of view.

On the other hand, transforming PC²R nets into SPQR nets can make much sense for the aforementioned reasons. Significantly, it is always possible to transform a PC²R net system into an equivalent SB SPQR net system, preserving its behaviour with relation to liveness (indeed, the language of firing sequences is equivalent in the transformed net system).

The transformation rule is based on the idea of converting every `while-do` block into an acyclic process which is activated by a lender resource place. This lender place gets marked once the thread reaches the while-do block. The token is removed at the exit of the iteration. This transformation must be applied from the innermost loops outwards. Figure 2.23 depicts the transformation rule. The rule preserves the language accepted by the net (and thus liveness) since it basically consists in the addition of a implicit place (place P1 in the right hand net of Fig. 2.23, since R_P1 can be seen as a renaming of P1 in the left hand net).

Figure 2.24 illustrates the transformation of the PC²R net system in Fig. 2.13 into the corresponding SB SPQR net system.

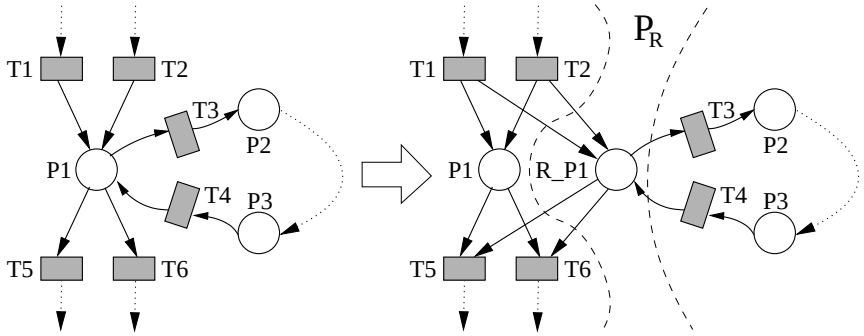



**Figure 2.23**: Transforming PC$^2$R into SB SPQR nets: From iterative to acyclic processes

Structurally speaking, it is also possible to transform an SB SPQR net into a PC$^2$R net. In the same vein than for S$^4$PR nets, it is just necessary to add a structural implicit place per process subnet connecting its drain transitions with its trigger transitions, which will be the idle place of the process subnet of the resulting PC$^2$R net.

Out of curiosity, the concept of *acceptable initial marking* has not been defined for the SPQR class, since it was conceived as a model to support generalised FMSs which support warm booting or where the processes are open plans. On the other hand, the concept of (0/1-) acceptable initial marking does have a physical meaning in the context of PC$^2$R nets modelling real-world systems in software engineering. As a lateral effect, the transformation of an SB SPQR net system $\langle \mathcal{N}, \mathbf{m_0} \rangle$ into a PC$^2$R net system $\langle \mathcal{N}', \mathbf{m_0}' \rangle$ may not produce a net system with a 0-acceptable initial marking $\mathbf{m_0}'$, even when $\mathbf{m_0}[P_S] = \mathbf{0}$.

## 2.6 Conclusions

Although there exist a variety of Petri net classes for RASs, many of these definition efforts have been directed to obtain powerful theoretical results for the analysis and synthesis of this kind of systems. Nevertheless, the process of abstraction is a central issue in order to have useful models from a real-world point of view, and therefore requires careful attention. In this chapter, that path has been followed, constructing a list of requirements for obtaining an interesting Petri net subclass of RAS models applied to the software engineering domain. Considering that list, the class of PC$^2$R nets has been defined. It fulfils those requirements while respecting the design philosophy on the RAS view of systems. Some useful transformation and class relations have also been introduced so as to locate the new class among the myriad of previous models.

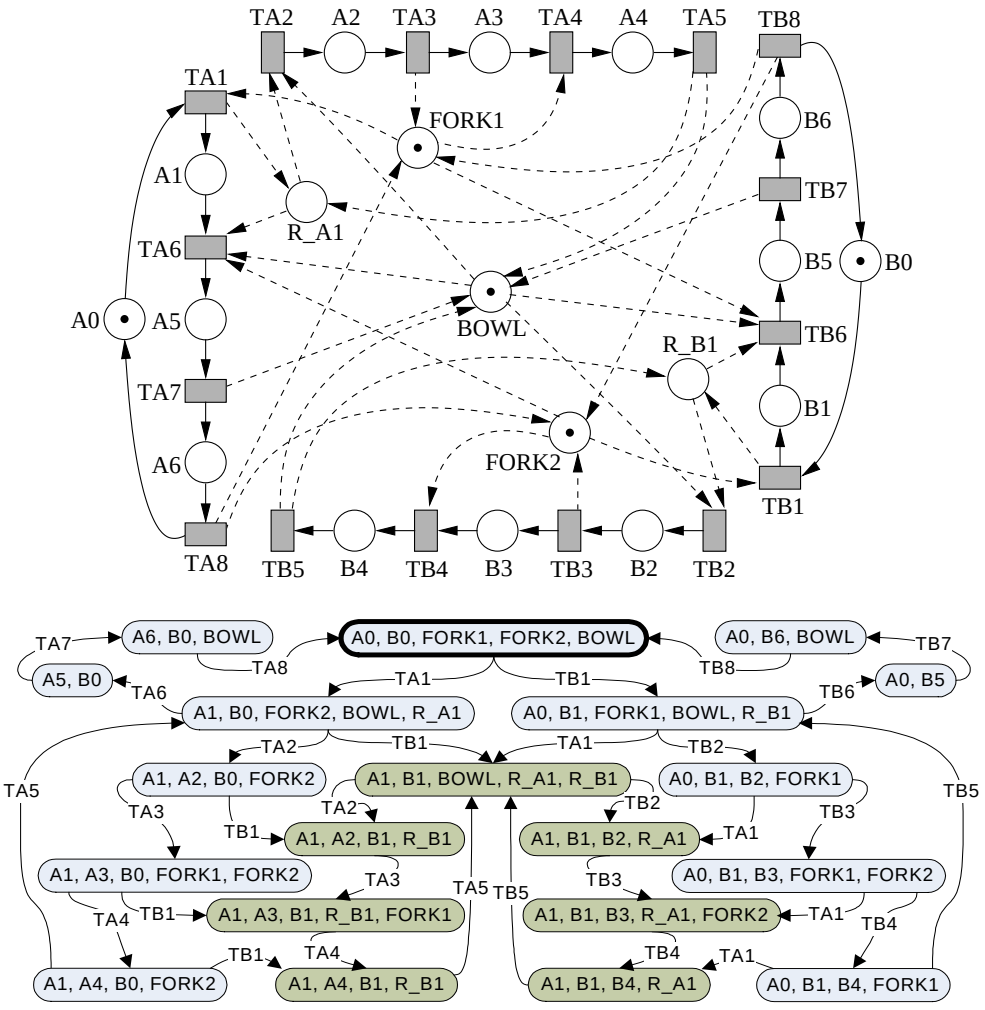



**Figure 2.24**: From PC$^2$R to SB SPQR nets: Two postmodern dining philosophers

Furthermore, it has hopefully been proved that the problem of liveness in the new context is non-trivial and presented some cases of bad behaviour. In some cases, the results are surprising and clearly reveal the inherent complexity of the class. Remarkably, it is revealed that the previous siphon-based characterisations for liveness analysis are no longer valid in this domain. Finally, a Petri net class called SPQR has been introduced for modelling a very general category of Sequential RASs. This allows to model abstractions of FMSs where the number of processes following the process plans is not limited and these processes can have already allocated resources

| **Property** | **L-S$^3$PR** | **S$^3$PR** | **S$^4$PR** | **S$^5$PR** | **PC$^2$R [1]** | **PC$^2$R [2]** | **SPQR** |
|---|---|---|---|---|---|---|---|
| **Structural** | | | | | | | |
| *Well-formedness* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × |
| *Struct. directedness* | ✓ | × | × | × | × | × | × |
| **Behavioural** (for an $\mathbf{m_0}$ acceptable for the class) | | | | | | | |
| *RS = PRS* [3] | × | × | × | × | × | × | × |
| *Deadlock-free = Live* | × | × | × | × | × | × | × |
| *Liveness monotonicity* | ✓ | × | × | × | × | × | × |
| *Directedness* | ✓ | ✓ | ✓ | × | × | × | × |
| *Reversible* $\Longrightarrow$ *Live* | ✓ | ✓ | ✓ | ✓ | ✓ | × | × |
| *Realisable t-semiflows* | ✓ | ✓ | ✓ | ✓ | ✓ | × | × |

**Table 2.3**: Comparison of some basic structural and behavioural properties related to liveness. All behavioural properties are presented for net systems with acceptable initial markings, except for [1] (1-acceptable) and [2] (0-acceptable).

from the start. The new SPQR framework colligates previous models and theoretical achievements for dealing with the RAP. Additionally, the generalisation is enriched providing support for other types of Sequential RASs which were not supported by previous classes. A subclass of SPQR has been identified which looks promising so as to provide a simple yet powerful theoretical framework in which study the liveness analysis problem for multithreaded software systems.

To sum up, Table 2.3 highlights some similarities and differences between the different members of the S$^n$PR family, while Fig. 2.25 introduces the inclusion relations between the Petri net classes for Sequential RASs which have been introduced so far.

[3] Reachability Set = Potentially Reachability Set.

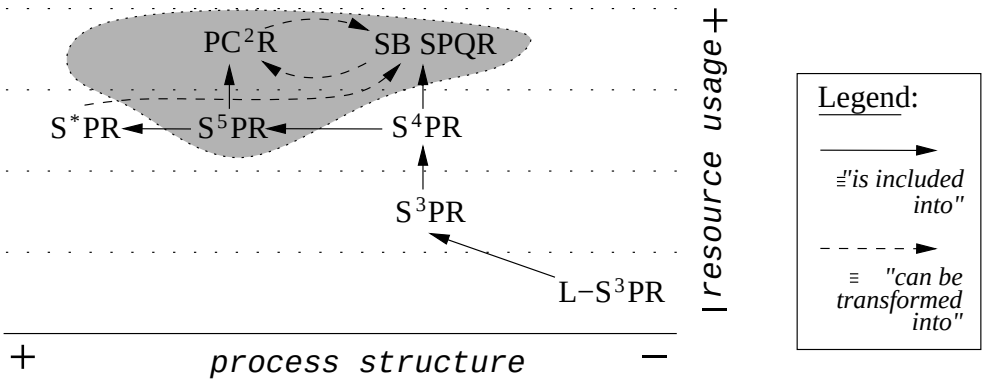



**Figure 2.25**: Inclusion relations between Petri net classes for RASs (first update of Fig. 1.5)

# Chapter 3

# The liveness problem: Characterisation, analysis and synthesis


## Summary

The long interest in finding efficient solutions to deadlock occurrence induced by resource sharing is strong and persistent in the context of parallel/concurrent software production. Among the myriad of techniques discussed over the years, the import of Petri net-based correction techniques which were traditionally applied in the context of FMSs constitutes a promising new approach. In order to properly tackle the problem in this domain, however, it is necessary to introduce a Petri net superclass generalising previous FMS models, as discussed in Chap. 2. The unprecedented behavioural phenomena observed there prove that the well-known siphon-based liveness characterisation falls short in the new context. In this vein, it is necessary to push past boundaries in the quest for new analytical results characterising the deadlock situations in the general case. In this chapter, we take that road to introduce new necessary and sufficient liveness conditions for general $PC^2R$ nets. Unfortunately, there still remains an unexplored gap between both conditions which remains unexplored for some complex subclasses. A basic methodology is therefore proposed which deploys a toolbox of heuristics to approach complex multithreaded software systems and propose corrections to prevent non-liveness in such troublesome scenarios.

## 3.1 Introduction

Petri net-based methodological approaches in *deadlock prevention* are a success story in the domain of FMSs. Along the years, diverse Petri net classes for dealing with the RAP have emerged in the field. These are usually derived from the physical constraints of different plant configurations and are specifically adapted to their modelling necessities. Taking these syntactic restrictions into account, it is usually possible to find a structural characterisation of the liveness property (L-S$^3$PR [GV99], S$^3$PR [ECM95], S$^4$PR [Tri03]), captured through the concept of insufficiently marked siphon, as succinctly introduced in Chap. 1.

Notably, the identification of insufficiently marked siphons as structural elements characterising non-liveness has traditionally led to iterative prevention techniques based on the addition of monitors that prevent the siphons from being emptied. Unfortunately, these monitors can occasionally cut off legal markings which do not lead to any factual deadlock. For these more general Petri net classes, a deeper insight into the permissiveness of the deadlock prevention policy is required so as to not obtain a system with too many unnecessary control elements. Consequently, many works deal with different strategies of computation of monitors, in the quest for an appropriate tradeoff between permissiveness, computational efficiency and control simplicity.

In effect, considering this blooming of publications around Petri net-based RAS approaches in the context of FMSs, it can be observed that the most significant proliferation of works emerges in the context of synthesis. Most of these papers are related to siphon computation [CRC12, LZ08] as well as to applying Integer Linear Programming (ILP) to liveness enforcing [TGVCE05, HZL11]. Another family of works focuses on synthesis based on reachability state analysis and on the theory of regions [GRX03, PCF09].

Recent works in the context of the design of minimal adaptive deadlock-free routing algorithms [Rov11] rely on the privatisation of resources as a mean to correct any potential deadlock without restricting the system concurrency. This crystallises in the Petri net through the splitting of some resource places, being 'duplicated' in such a way that the original bad siphons become broken. In the case of routing algorithms in interconnection networks, these new resource places are materialised through new virtual channels in the system. Again, these results are grounded in the concept of siphon as a mechanism to characterise deadlocks.

The resource pruning graph [CRC12] is a powerful artifact that not only allows the computation of every minimal siphon in a S$^4$PR net, but also provides a concise insight on how these siphons are constructed. For SOAR$^2$ nets (a subclass of S$^4$PR which deals with the aforementioned problem domain [Rov11]), minimal siphons are enough to characterise deadlock situations. Therefore, the information provided by

the resource pruning graph can be highly valuable to prevent deadlocks from happening by breaking any malicious siphon through the privatisation of some resources.

The multithreaded software domain, however, imposes limitations and challenges to those previous approaches. First, the manifestation of insufficiently marked siphons is not necessary for the existence of deadlock situations. Second, even when insufficiently marked siphons reveal that a deadlock exists, it may happen that none of these siphons is minimal. Both of these anomalies are illustrated in Chap. 2 and imply that the previous techniques must be extended or adapted to be applicable in the new domain. On the other hand, the technique of privatisation of resources must be revised and interpreted on the domain of multithreaded programming. Particularly, it must be analysed under which conditions a resource can be privatised in this context.

In short, the goal of this chapter is to draw the boundaries that the multithreaded software domain imposes to the application of analysis and synthesis techniques and results for RASs derived from other domains. At the same time, it seeks to generalise and extend the exploitation of these results to the extent possible. Accordingly, the chapter is completed with the presentation of a methodological proposal to address the correction of liveness problems in these systems through a synthesis toolbox that encompasses the prior knowledge of such systems as well as the new results developed throughout the chapter.

Section 3.2 describes the siphon-based techniques mentioned above with a greater profusion of details. Section 3.3 deals with the analysis of $PC^2R$ models using siphons, as well as the limits of this type of approach. Namely, the boundaries of the liveness characterisations proposed for simpler subclasses are presented with examples. An efficient ILP-based test condition for liveness is also proposed. In addition, some properties of the siphons in this kind of models are introduced. Finally, Sect. 3.4 assembles a toolbox of synthesis techniques for these models along with their interpretation in terms of the application domain.

Note that some additional figures are provided in Appendix B which complement some of the examples throughout the chapter with further information.

## 3.2 On siphon-based liveness enforcing in FMSs

### 3.2.1 The synthesis flow for liveness enforcing

Traditionally, Petri net-based deadlock prevention techniques for RASs have been based on the exploitation of the siphon as a structural element that captures the causal essence of a deadlock through its emptying. As discussed in Chap. 2, this emptying of a siphon is manifested in absolute terms for the most simple RAS net classes (L-$S^3$PR, $S^3$PR) while for more complex types of systems ($S^4$PR) deadlocks are characterised through the concept of insufficiently marked siphon, in which some

tokens may permanently remain in the siphon, yet still being insufficient to allow an eventual firing of certain transitions.

Fruitful results have been obtained in the field of FMSs by exploiting such concept. Siphon inspection allows determining which system processes may block each other and because of which allocations of resources. This allows introducing controllers in the system which inhibit the possibility of the system reaching such undesired states. These controllers are naturally represented in the Petri net as new resource places introducing Generalized Mutual Exclusion Constraints (GMECs) [GDS92] that forbid certain firing sequences. The incarnation of these monitors as resource places allows easy integration of the controlled system in the theoretical framework already developed. This has led to the proposal of iterative control methodologies which deploy strategies of incremental correction of potential deadlocks [TGVCE05].

Most works that approach the problem of live FMSs synthesis from this perspective usually exploit the fact that these monitors are represented by resource places in the Petri net. For sufficiently general net classes for RASs, such as $S^4PR$, this means that the Petri net that models the resulting system after the addition of the monitor still belongs to the same class of nets. In this case, it is possible to approach the augmented model using the same tools. This has facilitated the introduction of iterative correction methods which address possible deadlocks in a gradual manner until obtaining a live system [TGVCE05].

Often, each monitor that is added to the model is capable of removing a family of markings that belong to the same terminal strongly connected component, be it from the reachability space of the net or at least from the solution space of its net state equation. In terms of the net structure, this is achieved by monitoring the transitions that substract tokens from the siphon, so that it is not possible to drain the siphon beyond a certain threshold from which it cannot regain its initial marking. In other words, these transitions which potentially empty the siphon will request tokens from the new monitor place, and therefore always have a minimum of two input resource places, wherein at least one of them belongs to the siphon. That is why in the case of more restrictive Petri net subclasses such as $S^3PR$ the addition of monitor places takes us out of the class studied, since in them, each transition has at most one input resource place (indeed, only one resource is allocated to each active process in $S^3PR$ nets) [ECM95].

On the other hand, the addition of the monitor may cause the appearance of new siphons that include this new resource place in them. Those new siphons can eventually be insufficiently marked, making it necessary to consider the need to control them. Behaviourally, this can be explained by the conversion of non-terminal strongly connected components that inevitably lead to deadlock into terminal strongly connected components that contain now actual deadlocks, having removed the leaf components that hung from them. Consequently, iterative approaches need to consider not only

the siphons of the original net, but also those new siphons which appear when others are being controlled.

Furthermore, this kind of techniques have a major handicap. Although the strategy of addition of monitors works fine for simple RASs classes, such as those that are structurally safe [GVTCE98], such approaches have deeper problems in more general circumstances. Not only happens that sometimes there does not exist a single place to cut a terminal strongly connected component in a complete and isolated way, but occasionally the elimination of one of its markings through a monitor place requires removing legal markings from other strongly connected components. This happens even for RAS classes whose net models are ordinary and their processes are linear, i.e., which lack on-line decisions [GV99]. Later in Subsection 3.2.2 we will see an example of this type of phenomenon.

This circumstance is due to the fact that monitor places introduce linear marking constraints (i.e., the weighted sum of tokens in a set of places is limited by some number), while some of those unwanted markings may be trapped in the convex hull of the space of legal markings. This means that (to make a clean excision) we need a non-linear constraint that can cut this kind of markings. In this context, it has been proposed the introduction of disjunctive constraints through control mechanisms that are no longer a single place, but instead a subnet which overcomes the aforementioned obstacle [IA07]. A major difficulty in this kind of approaches is, again, the fact that the subnet introduced to control the siphon may leave the resulting Petri net out of the class of networks that can be analysed with the classic theory of siphons in RASs.

Another common approach for liveness enforcing through the addition of monitors is based on the exploration and manipulation of the state space of the net, leaving aside siphons as a structural means of approaching non-liveness. In this context, it is essential that bad states are categorised, taking into account not only those in which there exist dead transitions, but also those markings which are doomed to deadlock. Nevertheless, the biggest obstacle that this kind of approaches manifest is to handle the state explosion problem, aggravated especially in highly concurrent RASs.

Due to the computational complexity of dealing with the reachability set, the use of symbolic representations of sets of markings through Ordered Binary Decision Diagrams (OBDDs) has been proposed in order to obtain a compact representation of the set of forbidden markings [GVTCE98]. This ultimately allows the definition of a temporal logic and the efficient computation of the control logic (i.e., the set of monitors) through model checking techniques.

A different subfamily of approaches on the problem relies on the Theory of Regions (or variations of it) to tackle the state explosion problem and the addition of monitors that forbid bad states [GRX03, UZ07]. Recent works address the classification of the reachability space through non-linear classifiers, tackling the problem of bad markings that become trapped in the convex hull of the space of legal markings [NR12].

A different set of techniques is based on the structure of the Petri net as a graph. Typically, they consider the relations among resources and they look for dangerous relations, as circular waiting relations among resources. The Petri net structure typically reflects the relations of use among resources (which are the main problem when dealing with deadlocks) and, in this sense, several approaches have been presented trying to exploit this. One of the most common approaches is based on the usage of digraphs [FZ04]. In this vein, railway networks originally modelled by means of coloured Petri nets have been approached by way of digraphs from a RAS perspective [FGS06]. The resulting control logic is incorporated to the original net system in the form of colored monitor places.

The approaches based on the relation among bad states and structural components of the Petri net have their starting point in the seminal work of J. Ezpeleta, J-M. Colom and J. Martínez [ECM95]. That paper presented a characterisation of the liveness problem in $S^3PR$ in terms of empty siphons. Many papers have later tried to either extend those results to more general classes of Petri nets or to explore them [Tri03, PR01, TGVCE05, HJXC06, Rev07, HZL11].

A first family of approaches to the problem of synthesising live FMSs from a structural perspective is based on the computation of all siphons that may be involved in a deadlock, in order to control them. In the context of $S^3PR$ nets, this often results in the need for algorithms for computing the set of minimal siphons since, for this kind of models, any deadlock situation is associated with at least one minimal empty siphon. This is essentially the approach originally presented in the aforementioned seminal work [ECM95].

Some work has been done later to avoid the computation of all minimal siphons, trying to reduce the computational complexity by only controlling a significant subset of them [LZ08]. The core of this kind of approaches is to try to obtain truly independent siphons by means of its interactions via transitions (dependency is measured by linear combinations of transitions related to the siphon). Other works try to avoid the computation of all minimal siphons through the introduction of hybrid strategies in which the state space of the net is also explored [PCF09].

The pruning graph [CRC12] is a powerful artifact that not only allows the computation of every minimal siphon in a $S^4PR$ net, but also provides a concise insight on how these siphons are constructed. For $S^3PR$ nets, minimal siphons are enough to characterise deadlock situations. The pruning graph will be further introduced in Subsection 3.2.3.

An alternative route for structural control based on siphons passes through the introduction of iterative synthesis strategies based on the individual search of potentially dangerous siphons and their control through monitors. This scheme is repeated until a live system is obtained, which sometimes means avoiding the computation of a significant number of siphons.

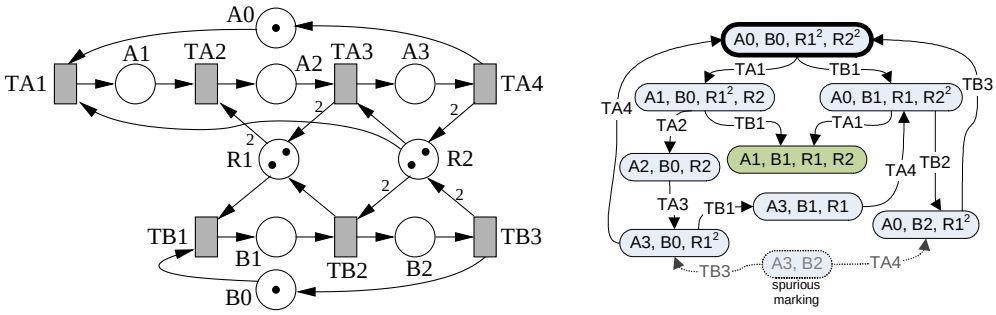



**Figure 3.1**: A non-live $S^4PR$ net system to be controlled

In this vein, several studies have appeared that address the computation of problematic siphons and monitor places through techniques based on ILP [Tri03, TGVCE05, HZL11]. In the next subsection we will discuss an approach based on this last family of techniques, which will be illustrated through examples.

### 3.2.2 Managing siphons for the computation of virtual resources

#### Computing virtual resources through mathematical programming

Fernando Tricas et al. [TGVCE05] present an iterative algorithm for deadlock prevention based on ILP which is reviewed on the following. This result is grounded on the structural characterisation already presented in Theorem 1.5 and deployed in the context of supervisory control of FMSs for the rather general class of $S^4PR$ nets. With the help of the net state equation, a set of Integer Linear Programming Problems (ILPPs) can be constructed which prevents the costly exploration of the state space. As far as we know, other works on ILP-based liveness enforcing depart from a similar strategy even though the objective function to be optimised may differ. To illustrate the algorithm, the $S^4PR$ net system in Fig. 3.1 (which was already introduced in Chap. 1 as Fig. 1.4) will be used.

In each iteration, this algorithm searches for a bad siphon and a potentially reachable marking under which the siphon is insufficiently marked. If found, a control place is suggested to prevent that siphon from ever becoming insufficiently marked. Such control place will be a virtual resource, in such a way that the resulting Petri net remains into the $S^4PR$ class. Thanks to this, a new iteration of the algorithm can be executed. The algorithm terminates as soon as there do not exist more siphons to be controlled, i.e., the system is live.

Prior to the introduction of the algorithm and its related ILPPs, some basic notation must be established.

In the following, for a given insufficiently marked siphon $D$, $D_{\mathrm{R}} = D \cap P_{\mathrm{R}}$ and $\mathbf{y}_{\boldsymbol{D_{\mathrm{R}}}} = \sum_{r \in D_{\mathrm{R}}} \mathbf{y}_{\boldsymbol{r}}$. Notice that $\mathbf{y}_{\boldsymbol{D_{\mathrm{R}}}}$ expresses the total amount of resource units belonging to $D$ (in fact, to $D_{\mathrm{R}}$) used by each active process in their process places. Also:

**Definition 3.1.** *[TGVCE05] Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an $S^4PR$ net system. Let $D$ be a siphon of $\mathcal{N}$. Then, $\mathcal{T}h_D = \|\mathbf{y}_{\boldsymbol{D_{\mathrm{R}}}}\| \setminus D$ is the* set of thieves *of $D$, i.e. the set of process places of the net that use resources of the siphon and do not belong to that siphon.*

The next system of restrictions relates the liveness characterisation introduced in Theorem 1.4 with the ILPPs which are used in the forthcoming algorithm. Essentially, the characterisation is reformulated into a set of linear restrictions given a reachable marking and a related bad siphon.

**Proposition 3.2.** *[TGVCE05] Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an $S^4PR$ net system. The net is non-live if and only if there exist a siphon $D$ and a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that the following set of inequalities has, at least, one solution:*

$$
\begin{cases}
\mathbf{m}[P_{\mathrm{S}}] \not\geq \mathbf{0} & & \texttt{-- } \exists t \in T : t \texttt{ is m-p-e} \\
\forall t \in T \setminus P_0^{\bullet} : & \textit{with } \{p\} = {}^{\bullet}t \cap P_{\mathrm{S}}, & \\
 & \mathbf{m}[p] \geq e_t & \texttt{-- } e_t\texttt{=0} : t \texttt{ is m-p-d} \\
 & e_t \geq \frac{\mathbf{m}[p]}{\mathbf{sb}[p]} & \texttt{-- } e_t\texttt{=1} : t \texttt{ is m-p-e} \\
\forall r \in D_{\mathrm{R}}, \forall t \in r^{\bullet} \setminus P_0^{\bullet} : & \frac{\mathbf{m}[r]}{\mathbf{Pre}[r,t]} \geq e_{rt} & \texttt{-- } e_{rt}\texttt{=0} : t \texttt{ is m-r-d by } r \\
 & e_{rt} \geq \frac{\mathbf{m}[r] - \mathbf{Pre}[r,t] + 1}{\mathbf{m_0}[r] - \mathbf{Pre}[r,t] + 1} & \texttt{-- } e_{rt}\texttt{=1} : t \texttt{ is m-r-e by } r \\
\forall r \in P_{\mathrm{R}} \setminus D_{\mathrm{R}}, \forall t \in r^{\bullet} \setminus P_0^{\bullet} : & e_{rt} = 1 & \texttt{-- } e_{rt}\texttt{=1} : r \notin D \\
\forall t \in T \setminus P_0^{\bullet} : & \sum_{r \in {}^{\bullet}t \cap P_{\mathrm{R}}} e_{rt} < |{}^{\bullet}t \cap P_{\mathrm{R}}| + 1 - e_t & \texttt{-- if } t \texttt{ is m-p-e then} \\
 & e_t \in \{0,1\} & \qquad t \texttt{ is m-r-d by } D_R \\
\forall r \in D_{\mathrm{R}}, \forall t \in r^{\bullet} \setminus P_0^{\bullet} : & e_{rt} \in \{0,1\}. &
\end{cases}
\tag{3.1}
$$

*where $\mathbf{sb}[p]$ denotes the structural bound of $p$ [CS91]*

*Note that* **m-p-e** *(***m-p-d***) stands for* **m***-process-enabled (***m***-process-disabled) and* **m-r-e** *(***m-r-d***) stands for* **m***-resource-enabled (***m***-resource-disabled).*

The following proposition introduces a set of additional restrictions on the system (3.1) that characterise the condition of siphon for the set of places whose respective variables $v_p$ equal zero (observe that, for notational simplicity, we use $v_p$ for process places and $v_r$ for resource places). Note that the minimality of the siphon is not

required, which makes sense considering that no minimal siphon characterises liveness for the class of $S^4$PR nets, as introduced in Sect. 2.4. Therefore, the new proposition captures the characterisation introduced in Theorem 2.33 with a system of linear inequalities.

**Proposition 3.3.** *[TGVCE05] Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an $S^4PR$ net system. The net is non-live if and only if there exist a siphon $D$ and a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that the following set of inequalities has a solution with $D = \{p \in P_\mathrm{S} \cup P_\mathrm{R} \mid v_p = 0\}$:*

$$
\begin{cases}
\forall p \in P \setminus P_0, \forall t \in {}^\bullet p : v_p \geq \sum_{q \in {}^\bullet t} v_q - |{}^\bullet t| + 1 & \texttt{--} \;\; D \texttt{ is a siphon} \\
\sum_{p \in P \setminus P_0} \; v_p < |P \setminus P_0| & \texttt{--} \;\; \texttt{|}D\texttt{| > 1} \\
\mathbf{m}[P_\mathrm{S}] \not\geq \mathbf{0} & \texttt{--} \;\; \exists t \in T : t \texttt{ is m-p-e} \\
\forall t \in T \setminus P_0{}^\bullet : \quad \textit{with } \{p\} = {}^\bullet t \cap P_\mathrm{S}, & \\
\qquad \mathbf{m}[p] \geq e_t & \texttt{--} \;\; e_t\texttt{=0} : t \texttt{ is m-p-d} \\
\qquad e_t \geq \frac{\mathbf{m}[p]}{\mathbf{sb}[p]} & \texttt{--} \;\; e_t\texttt{=1} : t \texttt{ is m-p-e} \\
\forall r \in P_\mathrm{R}, \forall t \in r^\bullet \setminus P_0{}^\bullet : \quad \frac{\mathbf{m}[r]}{\mathbf{Pre}[r,t]} + v_r \geq e_{rt} & \texttt{--} \;\; e_{rt}\texttt{=0} : t \texttt{ is m-r-d by } r \\
& \qquad\qquad\qquad\qquad \texttt{and } r \in D \\
\qquad e_{rt} \geq \frac{\mathbf{m}[r] - \mathbf{Pre}[r,t] + 1}{\mathbf{m_0}[r] - \mathbf{Pre}[r,t] + 1} & \texttt{--} \;\; e_{rt}\texttt{=1} : t \texttt{ is m-r-e by } r \\
\qquad e_{rt} \geq v_r & \texttt{--} \;\; e_{rt}\texttt{=1} : r \notin D \\
\forall t \in T \setminus P_0{}^\bullet : \quad \sum_{r \in {}^\bullet t \cap P_\mathrm{R}} \; e_{rt} < |{}^\bullet t \cap P_\mathrm{R}| + 1 - e_t & \texttt{--} \;\; \texttt{if } t \texttt{ is m-p-e then} \\
\forall p \in P \setminus P_0 : \quad v_p \in \{0,1\} & \qquad t \texttt{ is m-r-d by } D_R \\
\forall t \in T \setminus P_0{}^\bullet : \quad e_t \in \{0,1\} & \\
\forall r \in P_\mathrm{R}, \forall t \in r^\bullet \setminus P_0{}^\bullet : \quad e_{rt} \in \{0,1\}. &
\end{cases}
\tag{3.2}
$$

*where $\mathbf{sb}[p]$ denotes the structural bound of $p$ [CS91].*

*Note that* **m-p-e** *(***m-p-d***) stands for* **m***-process-enabled (***m***-process-disabled) and* **m-r-e** *(***m-r-d***) stands for* **m***-resource-enabled (***m***-resource-disabled).*

Thanks to the addition of the net state equation as another linear restriction, the following theorem constructs an ILPP which can compute a marking and a bad siphon holding System (3.2). Nevertheless, that marking can be a spurious solution of the state equation. Since this kind of nets can have killing spurious solutions (i.e, spurious solutions which are non-live when the original net system is live) then the theorem establishes a necessary but not sufficient condition. This is usually not a problem when the objective is to obtain a live system: the only consequence can be that some harmless, unnecessary control places are added. These control places would forbid some markings which are not really reachable.

Since one siphon must be selected, the ILPP selects that with a minimal number of places, hoping that controlling the smallest siphons first may prevent controlling the bigger ones. Other works present analogous techniques with a different objective function for this ILPP [HZL11].

**Theorem 3.4.** *[TGVCE05] Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an S⁴PR net system. If the net is non-live, then there exist a siphon $D$ and a marking $\mathbf{m} \in \mathrm{PRS}(\mathcal{N}, \mathbf{m_0})$ such that the following set of inequalities has, at least, one solution with $D = \{p \in P_{\mathrm{S}} \cup P_{\mathrm{R}} \mid v_p = 0\}$:*

$$\begin{array}{ll} \max & \sum_{p \in P \setminus P_0} v_p \\ s.t. & \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \\ & \mathbf{m} \geq \mathbf{0}, \boldsymbol{\sigma} \in \mathbb{N}^{|T|} \\ & \text{System } (3.2) \end{array}$$

The previous theorem can compute a marking $\mathbf{m}$ and a related bad siphon $D$. However, siphon $D$ can be related with a high number of deadlocks, and not only with that represented with $\mathbf{m}$. For that reason, the aim is to compute a control place able to cut every *unwanted* marking which the siphon $D$ is related to. Consequently, two different strategies are raised from the observation of the set of unwanted markings: (i) adding a place that introduces a lower bound of the number of available resources in the siphon for every reachable marking (*D-resource-place*), or (ii) adding a place that introduces an upper bound of the number of active processes which are withdrawing tokens from the siphon (*D-control-place*).

In order to define the initial marking of such places, two constants must be computed which are the result of two ILPPs. These ILPPs evaluate every unwanted marking that a bad siphon is related to:

**Definition 3.5.** *[TGVCE05] Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an S⁴PR net system. Let $D$ be an insufficiently marked siphon, $m_D^{\max}$ and $m_D^{\min}$ are defined as follows, with $v_p = 0$ iff $p \in D$:*

$$\begin{array}{ll} m_D^{\max} = \max & \sum_{r \in D_{\mathrm{R}}} \mathbf{m}[r] \\ s.t. & \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \\ & \mathbf{m} \geq \mathbf{0}, \boldsymbol{\sigma} \in \mathbb{N}^{|T|} \\ & \mathbf{m}[P_{\mathrm{S}} \setminus \mathcal{T}h_D] = \mathbf{0} \\ & \text{System } (3.1) \end{array} \qquad \begin{array}{ll} m_D^{\min} = \min & \sum_{p \in \mathcal{T}h_D} \mathbf{m}[p] \\ s.t. & \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \\ & \mathbf{m} \geq \mathbf{0}, \boldsymbol{\sigma} \in \mathbb{N}^{|T|} \\ & \mathbf{m}[P_{\mathrm{S}} \setminus \mathcal{T}h_D] = \mathbf{0} \\ & \text{System } (3.1) \end{array}$$

The next definition establishes the connectivity and the initial marking of the control place proposed for a given bad siphon $D$, both whether that place is a $D$-process-place or a $D$-resource place.

**Definition 3.6.** *[TGVCE05] Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a non-live S$^4$PR net system. Let $D$ be an insufficiently marked siphon, and $m_D^{\max}$ and $m_D^{\min}$ as in Definition 3.5. Then, the associated $D$-resource-place, $p_D$, is defined by means of the addition of the following incidence matrix row and initial marking: $\mathbf{C}^{\boldsymbol{p_D}}[p_D, T] = -\sum_{p \in \mathcal{T}h_D} \mathbf{y}_{\boldsymbol{D_R}}[p] \cdot \mathbf{C}[p, T]$, and $\mathbf{m_0^{p_D}}[p_D] = \mathbf{m_0}[D] - (m_D^{\max} + 1)$. The associated $D$-process-place, $p_D$, is defined by means of the addition of the following incidence matrix row and initial marking: $\mathbf{C}^{\boldsymbol{p_D}}[p_D, T] = -\sum_{p \in \mathcal{T}h_D} \mathbf{C}[p, T]$, and $\mathbf{m_0^{p_D}}[p_D] = m_D^{\min} - 1$.*

Finally, we can state the algorithm that computes the control places for a given S$^4$PR net system. In those cases in which a $D$-resource-place with an acceptable initial marking cannot be computed, the algorithm proposes the corresponding $D$-process-place, which always has an acceptable initial marking [TGVCE05].

**Algorithm 3.1** [TGVCE05] Synthesis of live S$^4$PR net systems

1. Compute an insufficiently marked siphon using the ILPP of Theorem 3.4.
2. Compute $m_D^{\max}$ (Definition 3.5).
   (a) If the associated $D$-resource-place (Definition 3.6) has an acceptable initial marking according to Definition 1.2, then let $p_D$ be that place, and go to step 3.
   (b) Else, compute $m_D^{\min}$ (Definition 3.5). Let $p_D$ be the associated $D$-process-place (Definition 3.6).
3. Add the control place $p_D$.
4. Go to step 1, taking as input the partially controlled systems, until no insufficiently marked siphons exist.

**Theorem 3.7.** *[TGVCE05] Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an S$^4$PR net system. Algorithm 3.1 applied to $\langle \mathcal{N}, \mathbf{m_0} \rangle$ terminates. The resulting controlled system, $\langle \mathcal{N}^{\mathrm{C}}, \mathbf{m_0^C} \rangle$, is a live S$^4$PR net system such that* $\mathrm{RS}(\mathcal{N}^{\mathrm{C}}, \mathbf{m_0^C}) \subseteq \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$.

Let us now apply Algorithm 3.1 to the net depicted in Fig. 3.1. There exists one deadlock ($\mathbf{m} \equiv [\mathrm{A1}, \mathrm{B1}, \mathrm{R1}, \mathrm{R2}]$) and two insufficiently marked siphons in $\mathbf{m}$, $D_1 = \{\mathrm{R1}, \mathrm{R2}, \mathrm{A3}, \mathrm{B2}\}$ and $D_2 = D_1 \cup \{\mathrm{A2}\}$. None of these is minimal. When applied step 1 of Algorithm 3.1, the ILPP of Theorem 3.4 returns $D = D_1$, since $D_1$ has less places than $D_2$. In step 2, we compute $m_D^{\max} = \mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{R2}] = 2$. Since the associated $D$-resource-place has not an acceptable initial marking (only one token in it is insufficient at $\mathbf{m_0}$), then we compute $m_D^{\min} = \mathbf{m}[A1] + \mathbf{m}[A2] + \mathbf{m}[B1] = 2$. In step 3, we add the associated $D$-process-place $p_D$ to the net. And finally, we go

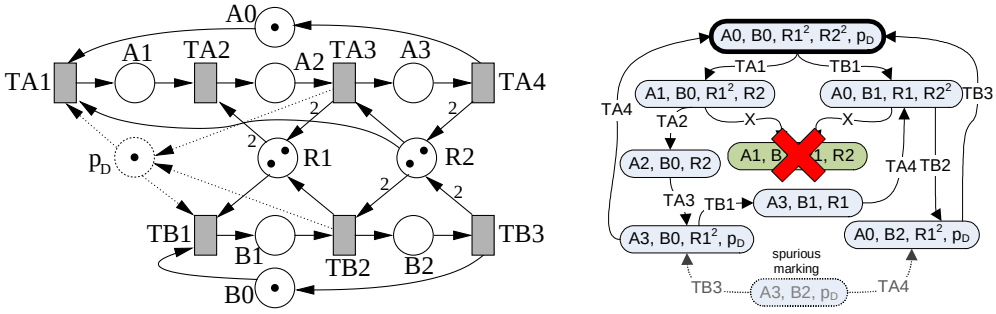



**Figure 3.2**: The controlled system after applying Algorithm 3.1 on the net in Fig. 3.1

back to step 1. But now the net is live and the ILPP of Theorem 3.4 has no solution, so the algorithm finishes after its first iteration. The resulting controlled system is depicted in Fig. 3.2.

### Limits on permissivity

Nevertheless, these techniques have certain limitations. For general $S^4PR$ nets, adding extra monitors to cut 'bad' states off and enforce liveness may also entail the removal of some legal markings. This happens when some unwanted marking can be obtained by a linear combination of two reachable markings which are legal (i.e., those which are not doomed to deadlock, and therefore ideally should not be forbidden). Then the unwanted marking is trapped in the convex hull of the space of legal markings. That means that any additional linear constraint that could be added to effectively forbid that marking necessarily would involve removing some legal extreme points. In other words, any GMEC which eliminates the unwanted marking also requires removing some legal marking.

This type of limitation has already been addressed from the standpoint of general Petri nets [GVTCE98]; in fact, the net in Fig. 3.3 is a variation of an illustrative example presented in that work. Hereinafter, we will address the correction of this net system by means of Algorithm 3.1. This net contains a unique (minimal) bad siphon $D = \{\mathrm{R1}, \mathrm{R2}, \mathrm{A2}, \mathrm{B2}\}$ which is insufficiently marked (in the sense derived from Theorem 1.5) at the following reachable markings:

$$[\mathrm{A0}, \mathrm{A1}, \mathrm{B0}, \mathrm{B1}, \mathrm{R1}, \mathrm{R2}] \equiv \mathbf{m_1}$$
$$[\mathrm{A1}^2, \mathrm{B0}, \mathrm{B1}, \mathrm{R2}]$$
$$[\mathrm{A0}, \mathrm{A1}, \mathrm{B1}^2, \mathrm{R1}]$$

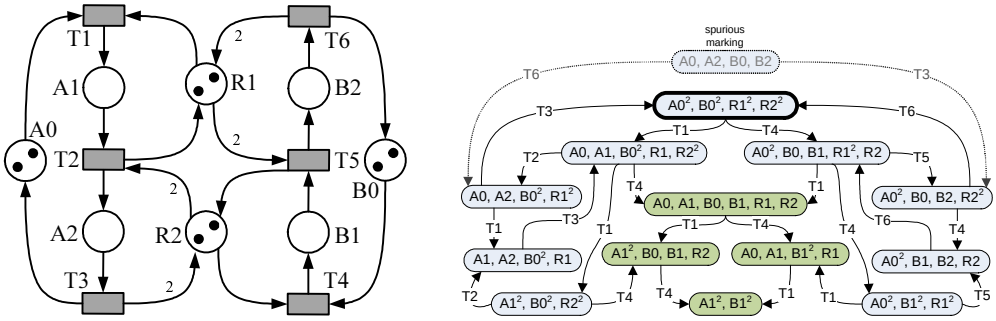



**Figure 3.3**: A non-live $S^4PR$ net system with a non-convex permissible marking space

$$[\mathrm{A0}^2, \mathrm{B0}^2] \equiv \mathbf{m_2}$$

Indeed, the siphon $D$ is empty at $\mathbf{m_2}$, which is a deadlock. Note that the other three markings are doomed to deadlock, and therefore should be forbidden as well. When applied step 1 of Algorithm 3.1, the ILPP of Theorem 3.4 returns $D$. In step 2, we compute $m_D^{\max} = 2$ (this value is obtained because of $\mathbf{m_1}$). In step 3, we add the associated $D$-resource-place $p_D$ to the net. And finally, we go back to step 1. But now the net is live and the ILPP of Theorem 3.4 has no solution, so the algorithm terminates. The resulting controlled system is depicted in Fig. 3.4.

As shown in Fig. 3.4, the proposed solution includes the addition of a resource place $p_D$ cutting the four unwelcome markings that are shaded in a different color in both figures. However, this place also forbids two additional markings which do not inevitably lead to deadlock: $[\mathrm{A1}^2, \mathrm{B0}^2, \mathrm{R2}^2]$ and $[\mathrm{A0}^2, \mathrm{B1}^2, \mathrm{R1}^2]$. Note that $\mathbf{m_0}$ is still reachable from those markings.

This last observation might suggest that the new resource place is not the optimal so as to cut the unwanted states off. However, it is easy to check that the marking $[\mathrm{A0}, \mathrm{A1}, \mathrm{B0}, \mathrm{B1}, \mathrm{R1}, \mathrm{R2}]$, which inevitably leads to deadlock, lies in a midpoint between the two legal markings which have been eliminated. Certainly:

$$\frac{[\mathrm{A1}^2, \mathrm{B0}^2, \mathrm{R2}^2] + [\mathrm{A0}^2, \mathrm{B1}^2, \mathrm{R1}^2]}{2} = [\mathrm{A0}, \mathrm{A1}, \mathrm{B0}, \mathrm{B1}, \mathrm{R1}, \mathrm{R2}]$$

Therefore, any linear constraint added to remove such marking inevitably requires the elimination of any of those two legal markings.

This example also illustrates another limitation of these techniques: by adding linear constraints that merely inhibit reachable markings, and therefore possible firing sequences, the system concurrence is reduced. In many cases, this negatively affects the system performance. The proposed control logic depicted in Fig. 3.4 not only

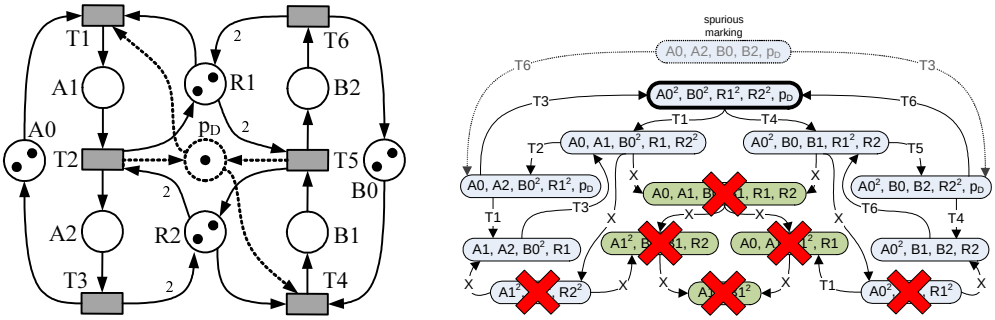



**Figure 3.4**: The controlled system after applying Algorithm 3.1 on the net in Fig. 3.3

inhibits completely the possibility that a process of type A (left process subnet) and a process of type B (right process subnet) ever coexist in the system, but even the circumstances under which two processes of the same type concur are reduced.

### Non-redundant virtual resources

Another possible undesired consequence of using this type of approach is that of obtaining a set of places wherein some of them may be redundant for the control of the net, in the sense that their removal would preserve the language of firing sequences of the net. In other words, some of the control places may be implicit places.

As introduced in Lemma 2.27, every resource place of an $S^4PR$ net is structurally implicit. Therefore, every monitor place added to prevent a siphon from becoming insufficiently marked is also an SIP. This does not necessarily mean that these places are implicit, as this depends on their initial marking. In fact, usually the last resource place added with this type of iterative approach is not an implicit place. This is due to the fact that it must cut at least one bad marking that was allowed so far. Provided that at least one such marking is not a spurious marking we can state that the introduction of the monitor place restricts the language of firing sequences of the net, and thus the place is not implicit.

However, the introduction of a new resource place, even not being implicit, can make redundant (i.e., implicit) some of the previously existing monitor places in the net [Tri03]. From the standpoint of the control engineer, this is an undesirable situation that can lead to the introduction of unnecessary control mechanisms involving a cost overrun or overcomplicate the system maintenance.

For this reason, post-processing techniques should be applied to cut implicit places after obtaining the control logic. Despite the NP-completeness of determining, in the general case, the minimum initial marking that makes an SIP implicit [GVC99], there

still exist efficient techniques based on sufficient conditions for determining if a control place with a given initial marking is implicit. Remarkably, a technique based on Linear Programming which allows to determine it with a polynomial time cost in the worst case has been proposed [GVC99].

Such kind of techniques can be used iteratively on the set of control places in order to eliminate redundancies. In the worst case, this involves solving the above Linear Programming Problem at most as many times as control places exist in the net, which still implies a cost in polynomial time in the size of the net.

### 3.2.3 Siphon computation via the resource pruning graph

#### The resource pruning graph

The resource pruning graph [CRC12] is a powerful artifact that allows an efficient computation of the set of minimal siphons of an $S^4PR$ net. In this graph, nodes represent the minimal siphons containing a single resource place, while the arcs represent the pruning relations between those 'seed' siphons. Particularly, such 'seed' siphons are unique, and every minimal siphon of a $S^4PR$ net containing more than one resource place can be obtained by observation of the pruning relations between the set of minimal siphons containing each of its resource places in isolation [CRC12]. Note that two minimal siphons are in a pruning relation if their union contains places that are non-essential.

For more restrictive classes of nets, such as $S^3PR$ or $SOAR^2$, it is even possible to obtain the set of minimal siphons by purely algebraic methods for the different combinations of the 'seed' minimal siphons [Rov11]. But in general, the technique requires the search and identification of strongly connected subgraphs under certain conditions to ratify the minimality of the resulting siphon.

Figure 3.5 depicts a $S^3PR$ net that illustrates the technique above. The net has already been introduced in Chap. 2 (Fig. 2.12), although here has been instantiated with an acceptable initial marking that makes the system non-live. Indeed, Fig. 3.6 displays the reachability graph of the net system, in which a reachable deadlock can be observed: $[A4, B4, R2, R3^2]$.

Figure 3.7 exposes the resource pruning graph of the net in Fig. 3.5. The figure reveals that the net has four minimal siphons containing a single resource place ($D_{R1}$, $D_{R2}$, $D_{R3}$, $D_{R4}$), symbolised by the four nodes of the pruning graph. Each one of these minimal siphons fits exactly the support of the minimum p-semiflow $\mathbf{y}_{\boldsymbol{r}}$ induced by the corresponding resource place of the net. This always happens in the case of $S^3PR$ nets, but it is not necessarily so for general $S^4PR$ nets: although the support of a p-semiflow is always a siphon, it does not necessarily have to be a minimal one.

The figure shows that every arc of the graph is labelled with at least one pair (transition, process place). Nonetheless, an arc can be, in general, labelled with a

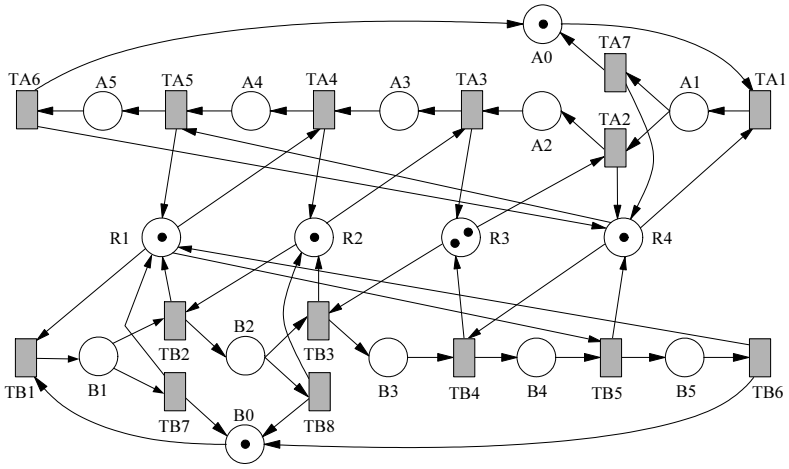



**Figure 3.5**: A non-live $S^3PR$ net

number of these pairs. This set of pairs describes the pruning relation between the two siphons involved. For instance, the directed arc from the node corresponding to $D_{\mathrm{R1}}$ (labelled with R1 in the figure for the sake of simplicity) to the node corresponding to $D_{\mathrm{R4}}$ (labelled with R4) is labelled with the pair $(\mathrm{TB5}, \mathrm{B4})$. This label indicates that siphon $D_{\mathrm{R1}}$, through transition TB5, has the ability to make B4 from siphon $D_{\mathrm{R4}}$ a non-essential place (and, eventually, make the same for some of the preceding process places) in the joint construction of any minimal siphon containing the resource places R1 and R4. In fact, the figure reveals that the unique minimal siphon containing places R1 and R4, $D_{\mathrm{R1R4}} = \{\mathrm{R1}, \mathrm{R4}, \mathrm{B1}, \mathrm{B5}, \mathrm{A1}, \mathrm{A5}\}$, can be obtained from the union of the siphons $D_{\mathrm{R1}}$ and $D_{\mathrm{R4}}$ minus the places B4 and A4, which are the ones in the labels of the arcs between those nodes labelled R1 and R4. In that sense, it is said that siphon $D_{\mathrm{R1}}$ *purges* place B4 from siphon $D_{\mathrm{R4}}$, while siphon $D_{\mathrm{R4}}$ does the same with place A4 from siphon $D_{\mathrm{R1}}$.

From the above it is easy to deduce that the mutual purge of at least one essential place by two siphons is a necessary and sufficient condition for the existence of a minimum siphon containing only the corresponding resource places. Similarly, the purging of a place for every 'seed' siphon in relations that involve more than a siphon through a strongly connected component of the graph is a sufficient condition for the existence of the corresponding minimal siphon, involving all (and only those) resource places derived from the component. However, this condition is no longer necessary due to the possible existence of embedded minimal siphons, being necessary, in the general case of $S^4PR$ nets, the study of the strongly connected subgraphs contained in it.

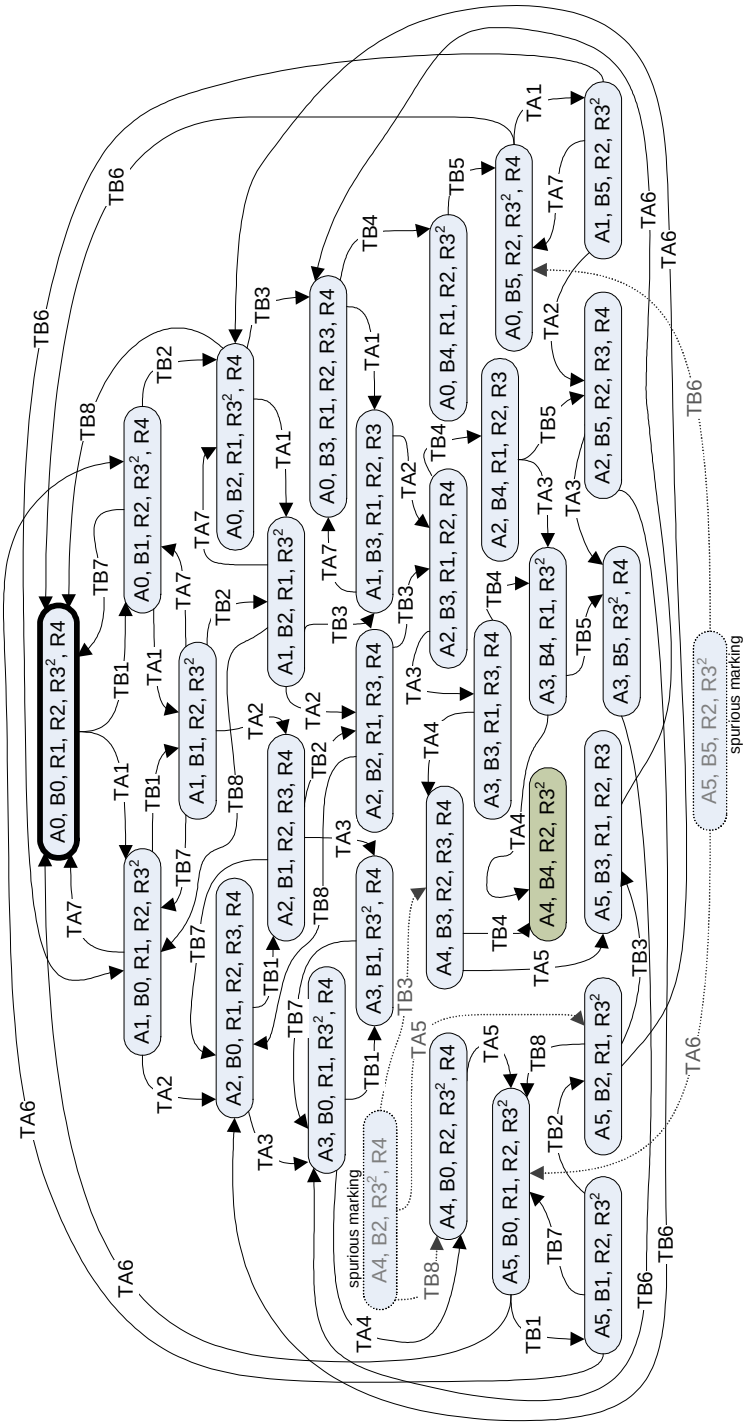
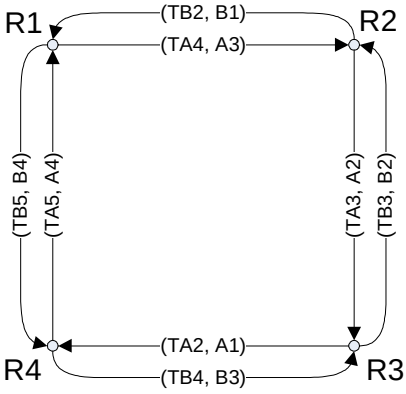



**Figure 3.6**: Reachability graph of the net system in Fig. 3.5. The net has one deadlock, stressed on a different colour




$D_{\mathrm{R1}} = \{\mathrm{R1}, \mathrm{A4}, \mathrm{B1}, \mathrm{B5}\}$
$D_{\mathrm{R2}} = \{\mathrm{R2}, \mathrm{A3}, \mathrm{B2}\}$
$D_{\mathrm{R3}} = \{\mathrm{R3}, \mathrm{A2}, \mathrm{B3}\}$
$D_{\mathrm{R4}} = \{\mathrm{R4}, \mathrm{A1}, \mathrm{A5}, \mathrm{B4}\}$

$\mathrm{K}_G(\mathrm{R1}) = \{\mathrm{A4}\}$
$\mathrm{K}_G(\mathrm{R2}) = \{\mathrm{A3}\}$
$\mathrm{K}_G(\mathrm{R3}) = \{\mathrm{A2}, \mathrm{B3}\}$
$\mathrm{K}_G(\mathrm{R4}) = \{\mathrm{B4}\}$

**Figure 3.7**: Resource pruning graph $G$ of the net in Fig. 3.5

A quick reading of the above could lead to the wrong conclusion that the existence of, for example, a cycle between two nodes necessarily implies the existence of a minimal siphon comprising only the corresponding two resource places. At this point, it is important to note that the appearance of a particular resource place on the label of an arc does not necessarily imply its non-essentiality in any siphon that includes the union of the two minimal siphons involved. On the contrary, it can be observed in the net of the example that there exists no minimal siphon containing the resource places R1 and R2, despite the existence of a cycle between the nodes corresponding to $D_{\mathrm{R1}}$ and $D_{\mathrm{R2}}$. This is due to the fact that $D_{\mathrm{R2}}$ cannot ever purge place B1 of $D_{\mathrm{R1}}$, since transition TB7 makes place B1 essential for every siphon containing the resource place R1.

That is why from the pruning resource graph depicted in Fig. 3.7 it follows that the only minimal siphon possible for the net in Fig. 3.5 is precisely the siphon $D_{\mathrm{R1R4}}$. An algorithm has been provided to compute the set of minimal siphons of a $\mathrm{S}^4\mathrm{PR}$ net from the resource pruning graph [CRC12].

Note that Fig. 3.7 includes the value of a labeling function ($\mathrm{K}_G$) associated with each node of the resource pruning graph $G$. This function actually gives us information about what should be pruned from each node in the eventual construction of a minimal siphon containing all resource places in the graph $G$. In general, it may be necessary to recalculate this function for each of the strongly connected subgraphs contained in $G$. The latter can potentially return different values for the same 'seed' siphon. For example, for the subgraph $G'$ that only contains the nodes R2 and R3, the computation of $\mathrm{K}_{G'}(\mathrm{R2})$ returns the empty set, which contrasts with $\mathrm{K}_G(\mathrm{R2}) = \{\mathrm{A3}\}$ and shows us that there is no minimal siphon containing only R2 and R3, since $D_{\mathrm{R3}}$

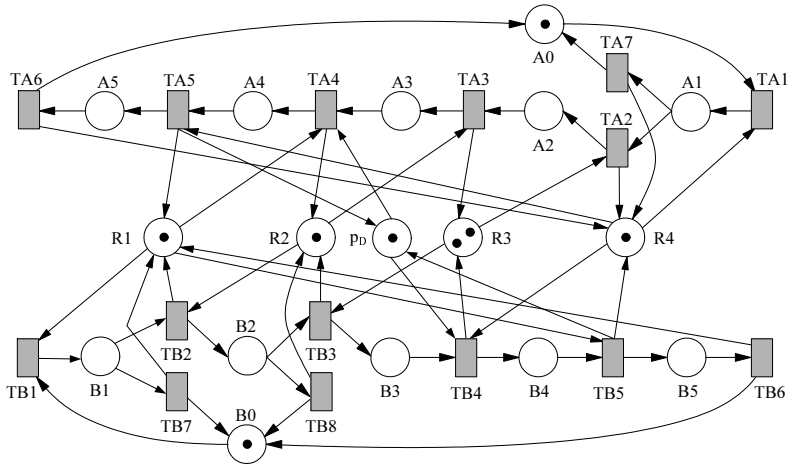



**Figure 3.8**: Controlled version of the net in Fig. 3.5

is not able to purge anything from $D_{\mathrm{R2}}$ by itself.

The value returned by this labelling function is essential in the algorithm for computing the minimal siphons of the net. This function is approached in more detail (and also the algorithm to compute it) in subsequent sections of this chapter.

In the case of $\mathrm{S}^3\mathrm{PR}$ nets, the computation of the set of minimal siphons is especially useful as a tool for controlling the net system since, as we have seen in Chap. 2, non-liveness can be characterised by the existence of a minimal siphon containing more than one resource place and becoming eventually empty. In that sense, the information provided by the resource pruning graph guarantees the possibility of controlling this type of siphons to prevent any eventual emptying and obtain a live system. Indeed, for the net of Fig. 3.5 it is possible to control the minimal siphon $D_{\mathrm{R1R4}}$, computed through the resource pruning graph, by means of the corresponding $D_{\mathrm{R1R4}}$-resource place. Obviously this is the only bad siphon containing more than one place of resource and therefore the siphon which empties in the deadlock $[\mathrm{A4}, \mathrm{B4}, \mathrm{R2}, \mathrm{R3}^2]$ that can be observed in Fig. 3.6 for the given acceptable initial marking. The net that results from adding the control place $p_D$ is shown in Fig. 3.8.

However, the above approach presents some inconvenience. In particular, it should be noted that the introduction of place $p_D$, a new resource place, implies the modification of the resource pruning graph by adding a new node that can close new cycles through the pruning relations that induces in the other siphons. Therefore, the addition of the place can introduce new recomputation needs in the general case. Similarly, it emerges the need to ensure the existence of a breakpoint for the algorithm that computes the control logic so that it always converges in obtaining a live system.

In this regard, it is worth paying attention to a recently introduced synthesis approach [Rov11]. Despite being based on the resource pruning graph, this novel control policy does not entail the introduction of new cycles in it. Although the results are also relevant in the domain of FMSs [LGCT14], these were originally presented in the context of the development of minimal adaptive routing algorithms in interconnection networks [Rov11], and presented for the class of SOAR$^2$ nets. SOAR$^2$ is a superclass of S$^3$PR (but subclass of S$^4$PR) of particular significance in the aforementioned application domain because of the physical restrictions regarding resource usage in this kind of systems.

Such an approach is based on the idea of privatising the use of resources by the processes. In algorithmic terms, this is achieved by breaking the resource pruning graph to make it acyclic and thus avoiding any possibility of deadlock due to the allocation of shared resources. This is obviously a conservative approach which leads to obtaining a live net system for any acceptable initial marking, which is particularly interesting in a context like that of multithreaded software engineering, in which, for example, an upper bound of the number of concurrent threads in the system may be unknown at compile time. Therefore, this kind of techniques is of particular relevance considering the field of study of this thesis.

This type of approach is further addressed in subsection 3.2.4.

#### Putting all together

Throughout the last two subsections, an overview on some cutting edge approaches on structural-based synthesis techniques based on the addition of monitors has been presented, with a detailed look on an iterative technique based on ILP and a novel technique for computing the whole set of minimal (bad) siphons.

This kind of techniques have their origin in the classic control theory, in the sense that the objective is to constrain the behaviour of the FMS by monitoring the system so as to forbid unwanted states or event sequences. Such strategies are specially valuable when trying to strictly comply with the original design of the FMS, i.e., the production plans are fixed and it is not possible or desirable to increase the number of system resources or privatise their use.

Nonetheless, such approaches still present some disadvantages. One obvious drawback exists on the fact that by constraining the language of event occurrences (i.e., firing sequences in the net model) the system concurrency is reduced. This can be detrimental to the overall system performance.

Another metric that can be degraded is the resource utilisation rate. In effect, since the control logic prevents some resources from being used under certain circumstances (i.e., when a deadlock may occur) the involved resources may result underused in the long term.

Another problem which was illustrated in Subsection 3.2.2 is that the strategy of adding linear restrictions through monitors to enforce liveness may inevitably imply that some legal behaviour is prevented from happening. This was already illustrated through the net in Fig. 3.3 and Fig. 3.4.

Finally, the computation cost of computing all siphons that can be problematic so as to control them (and remove the redundant control logic) can be specially demanding for systems in which there exists a high number of types of resources.

In the next subsection we present a different family of structural synthesis techniques which somehow transgress some of the principles of classic control theory but can provide a different category of solutions which can be complementary in certain scenarios.

### 3.2.4 Structural regions and the privatisation of resources

#### Principles of the technique

So far we have reviewed the structural synthesis methods directly derived from the characterisation of the liveness property. In this context the reason for the non-liveness appearing is in the bad siphons with a small number of tokens inside. The synthesis strategy consists of the control of the siphon in such a way that the number of tokens inside the siphon always is greater than or equal to a value guaranteeing the liveness of the transitions covered by the siphon. Therefore the basis of the strategy is a control-based strategy where we introduce extra constraints to the transitions allocating new copies of the resources of the siphon that prevent that the number of these copies go under a dangerous value. Given the iterative nature of the method, the constraints are constructed as new (virtual) resources of the system whose availability represents the constraints to the allocation of the original resources. This procedure allows us to maintain the net controlled in an intermediate iteration within the class of nets for which a siphon-based characterisation of the liveness property exists, and so we can proceed with the next iteration.

We must point out here that these techniques adhere to some common rules or characteristics. The infringement of such rules gives rise to the new techniques introduced later in this section. The rules or characteristics shared by those techniques based on virtual resources are:

- The controlled net respects the *original state machines* (the production plans) and the *initial marking of the state machines* (the initial marking of the idle places). For the designer, this means that the original system is completely maintained and so s/he can identify his/her original design.

- The same happens with respect to the original resources of the net system. The controlled net contains the *same resource types* (resource places) as in the

original net, and the *initial marking of these resource places is the same* (the number of copies of each type of original resource is maintained).

- Liveness enforcing by means of virtual resources is based on the addition of some places that represent additional constraints to the transitions in charge of the allocation of some original resources when a request of a process (a token in a process place) arrives to the controller of the resource. The effect of these additional virtual resources is to forbid some occurrence sequences of transitions of the original net to prevent an excessive decay in the marking of a bad siphon. In other words, it is the classical approach inside control theory, where the goal is to forbid states or occurrence sequences. Therefore, the constraints reduce the concurrency inside the system because some occurrence sequences are forbidden and the resource utilisation rate is reduced because some states where resources are in use are forbidden.

Observe that these three characteristics are shared by all techniques inspired in the use of siphons. Nevertheless, in this section, for a particular subclass of nets, we present a technique in which the three previous characteristics are not respected. That is, we propose a technique where the central point is the addition of copies of the original resources. It is well known that in bounded systems, if we are able to increase the number of copies of resources then all deadlock problems disappear because the resource places become implicit places and can be removed. Nevertheless, we try to add a minimum number of copies of resources because they can be expensive, then in order to minimise this we specialise the original copies of a type of resource in the sense that certain copies only can be used by a subset of processes and the other copies of resources are used only by the rest of processes. This idea will be implemented by splitting a type of resource (a resource place) into two new types of resources each one used in a private way by a a disjoint subset of processes of the original system. Therefore, we increase the number of copies of the original resources but increasing the degree of privatisation of the use (or, in other words, reducing the degree of resource sharing). We will see that this technique, from a structural point of view, breaks the original bad siphons, and then if we are able to break all bad siphons the net cannot have deadlocks.

At our best knowledge, this is a completely novel strategy [Rov11]. The positive effect in the global behavior of the system is that *concurrency is not reduced*. In fact, all original states of the system remain reachable and *new states* are added representing the *recovery states from the deadlocks* to the safe states of the system. Therefore, these techniques are not based on the forbidden state strategy coming from control theory.

In the following, the non-live $S^3PR$ net depicted in Fig. 3.9.a is used to illustrate the different techniques of liveness enforcing presented in this work and the different

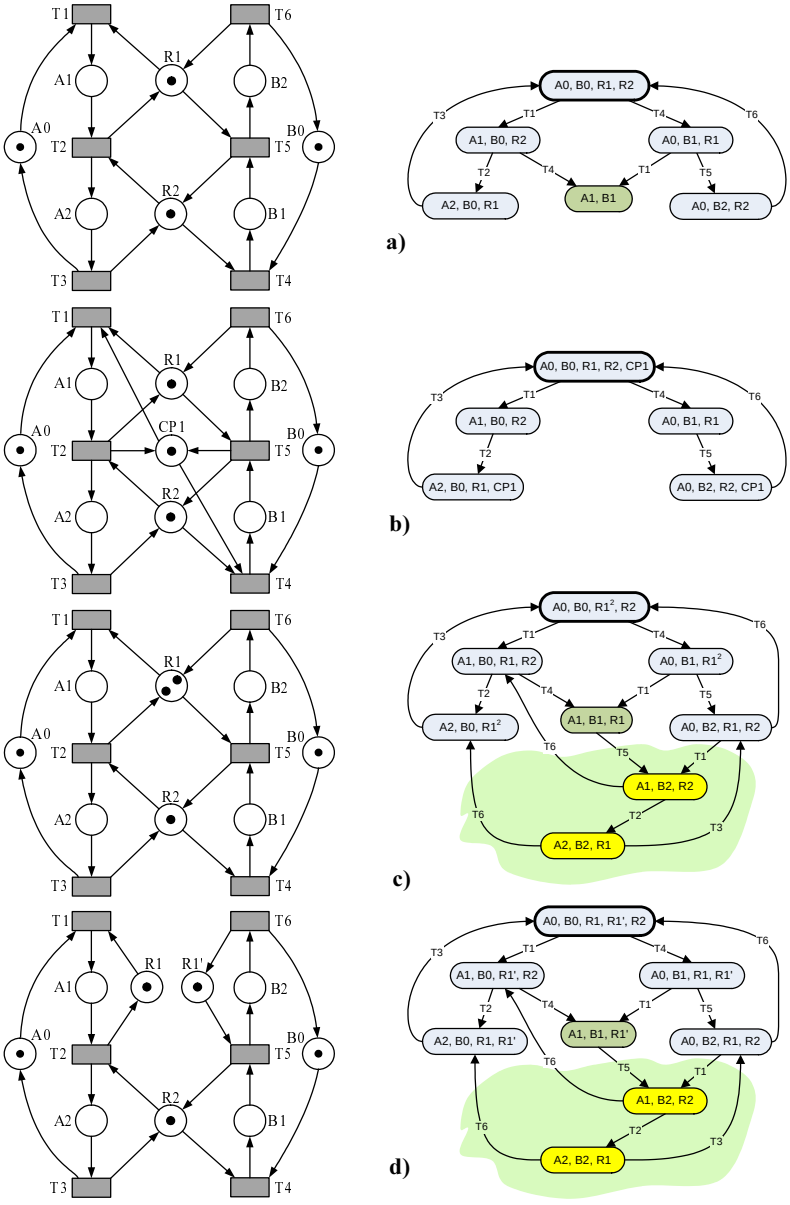



**Figure 3.9**: **a)** $S^3PR$ net which is non-live. **b)** The net in a) enforced to be live by the addition of the virtual resource CP1 computed from the bad siphon $\{A2, B2, R1, R2\}$. **c)** The net in a) that becomes live by the addition of an extra copy of the type of resource R1. **d)** The net in a) that becomes live by splitting the resource type R1 into two new resource types, R1 and R1', and each one is used in a private way by one of the process plans.

effect produced by each one in the behavior of the corrected net. The net in Fig. 3.9.a is non-live because of the reachable marking [A1 + B1] (a total deadlock) that appears in the reachability graph depicted on the right of the Petri Net. Figure 3.9.b illustrates the result of the application of the liveness enforcing technique presented in Subsection 3.2.2. In effect, in this net there exists the bad siphon $\{A2, B2, R1, R2\}$ that becomes empty at the marking [A1 + B1]. Therefore, from this siphon and the initial marking of the net the method computes the virtual resource place CP1 depicted in the figure with an initial marking equal to 1. This virtual resource (constraint) produces the removal of the deadlock marking [A1 + B1] as the reachability graph on the right of the figure points out. In other words, CP1 forbids the reachability of the state [A1 + B1].

In this section we propose the increase of the number of copies of some original resource place. A first approach would advocate for increasing in one unit the initial marking of one of the resources involved in the formation of the deadlock. In this case the two resources R1 and R2 are involved in the deadlock because they belong to the unique minimal bad siphon of the net: $\{A2, B2, R1, R2\}$. In order to enforce liveness, the resource R1 is selected and one extra token is added to the initial marking, i.e., one extra copy of the resource type R1 is added to the system as it is illustrated in Fig. 3.9.c. The effects on the behavior of the corrected model is that all states of the original net are preserved in the corrected net or at least can be identified. The other effect is that new states appear in the reachability graph allowing to recover the system from the deadlock state to a safe state. The result is that the net becomes live. Nevertheless, the reader can observe that with this technique *the bad siphons of the original net persist in the corrected net model*, and then from a structural point of view the problem has not been fixed. In effect, if you increase the initial marking of the idle place A0 making it equal to 2 tokens, then the net system becomes non-live.

The method proposed in this section is illustrated by means of the net in Fig. 3.9.d. We increase the number of copies of resources, but in order to fix the problem at a structural level this extra copy is of a resource type different to the preexisting copy. In other words, the resource type R1 is split into two different resource types R1 and R1', each one containing one copy (one copy more than in the original situation with only the resource type R1), and each one of these resource types is used in a private way from only one process plan. The resulting behavior is very similar to the previous one: the original states are maintained and new states appear allowing to recover from the deadlock state to a safe state. Nevertheless, the reader can observe that with this approach the split of R1 has broken the original bad siphon and now there are not bad siphons in the net. This means that if we have an acceptable initial marking in the net, the net will be live.

Finally, it is interesting to observe that the techniques based on the addition of copies of resources allow to maintain or even increase the existing concurrency in

the original net. The other interesting effect is that the method of virtual resources constrain the allocation of resources sequentialising the processes and therefore the resource utilisation ratio is lower than in the case of the additional copies of existing resources.

The implementation of this technique requires the analysis of the structure of the net identifying the structural regions (also called zones) in the net wherein a copy of a resource is used in a continuous way. These structural regions are the candidates to use the copies of the resource they need in a private way. In the following section, in order to illustrate the approach we introduce a subclass of $S^4PR$ nets named $SOAR^2$ nets that were originally introduced to model minimal adaptive routing algorithms of AGV systems [Rov11]. In this class of nets we define the concept of structural region as an structural object wherein a copy of a type of resource is used in a continuous way, the relations between those objects and the relations of the structural regions with the siphons of the net. The latter allow to identify the structural regions where a privatisation of the use of the copies of some resources would break all the siphons of the net. Taking into account that the method will maintain the corrected net inside the class of the $S^4PR$ nets, and the corrected net has an acceptable initial marking, then the net will be live.

### The class of nets $SOAR^2$ and its related properties

The $SOAR^2$ class of nets is a strict subclass of the $S^4PR$ nets. The acronym stands for *"$S^4PR$ with Ordered Allocation and Release of the Resources"*. This class was introduced in the PhD thesis of C. A. Rovetto to model minimal adaptive routing algorithms of objects or data between a source station/node and a destination station/node throughout a system for the guidance of automated vehicles or an interconnection network composed of communication channels. Therefore, the constraints imposed to define the class from the $S^4PR$ class come from the application domain and are justified by the application domain.

The main properties characterising this subclass with respect to the general class of $S^4PR$ are the following:

- The order in which the resources (rails or channels) are requested and allocated following a given route, is the same order in which these resources are released.
- A process in an intermediate step can have allocated simultaneously a set of resources.
- In a change of the state of a process (occurrence of a transition) only one single operation of allocation or release over a unique copy of a type of resource can be executed. In other words, it is not possible to have concurrency between operations of allocation/release of copies of resources.

- For each type of resource there is only a unique copy of resource belonging to this type of resource.

The translation of the previous characteristics to Petri net terms related to the superclass of $S^4PR$ gives rise to the following structural constraints and to some constraints for the acceptable initial markings valid for these nets.

1. In each circuit of each state machine of the net, the resources are released in the same order than they were allocated.
2. In each transition of the net, it is only possible to find a unique resource place connected to the transition. If the arc connecting the place inputs in the transition, this represents an allocation operation. If the arc connecting the place outputs from the transition, this represents a release operation.
3. A net of the $SOAR^2$ class is an ordinary Petri net, i.e. the weight of all arcs is equal to one.
4. The acceptable initial marking of all resource places in $SOAR^2$ nets is equal to 1.

In the above constraints on the class of $S^4PR$ to obtain the $SOAR^2$ class, the three last constraints are more or less easy to understand and to manage. Nevertheless, the first constraint requires the introduction of some new objects named *structural regions of continuous use of a copy of resource* (shortened: *s-regs*). Structural regions were originally called *zones of continuous use of a copy of resource* or simply *resource zones* [Rov11]. This concept is not only renamed, but generalised, in the context of this PhD thesis. The properties of these structural objects individually considered, and the relations defined over the set of s-regs are the basis of the new deadlock prevention technique intuitively introduced in the previous subsubsection. In the following, these concepts will be introduced with the aid of some examples.

An structural region $j$ of continuous use of a copy of the resource $r$ in the $i$-th process subnet of an $SOAR^2$ net is a maximal set of process places that are holders of the resource $r$, $\mathcal{S}^r_{i,j}$, such that the subnet generated by the s-reg and its input and output transitions is a connected state machine. In the upper process subnet of the $SOAR^2$ net of Fig. 3.10 it is possible to find the following set of s-regs: $\mathcal{S}^{R1}_{1,1} = \{A1, A2, A3\}$, $\mathcal{S}^{R2}_{1,1} = \{A2, A3, A4\}$ and $\mathcal{S}^{R3}_{1,1} = \{A3, A4, A5\}$. In a similar way, the s-regs of the bottom process subnet are: $\mathcal{S}^{R1}_{2,1} = \{B1, B2\}$, $\mathcal{S}^{R2}_{2,1} = \{B2, B3, B4\}$ and $\mathcal{S}^{R3}_{2,1} = \{B4, B5\}$. For a resource $r$ we can find s-regs that belong to different process subnets, and obviously they will be disjoint because they belong to disjoint state machines. It is also possible to find, for a resource $r$, several s-regs belonging to the same process subnet. Obviously, in this last case the s-regs must be disjoint. Other

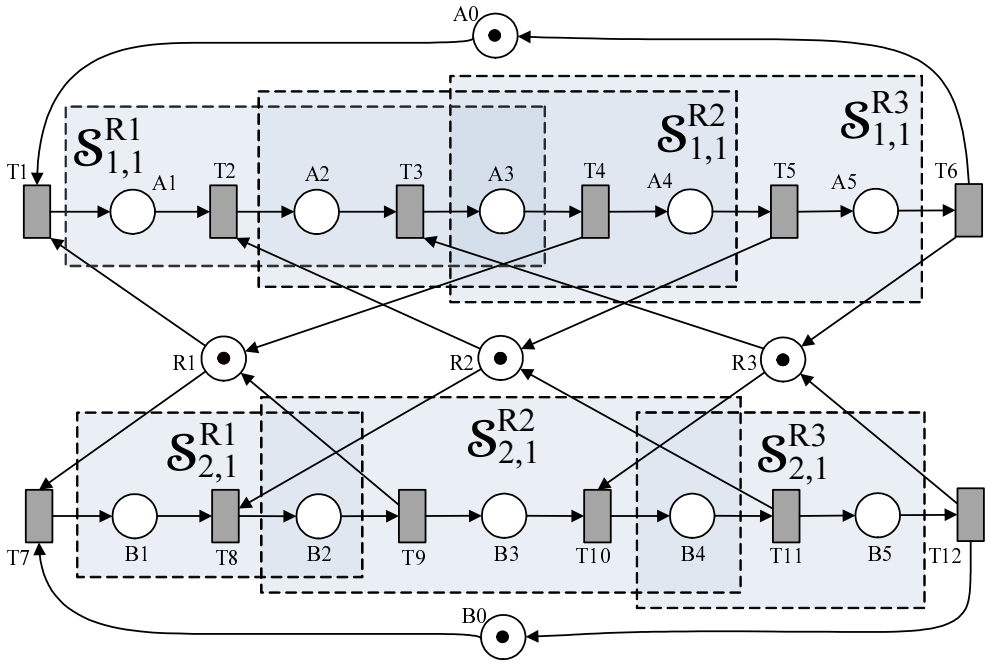
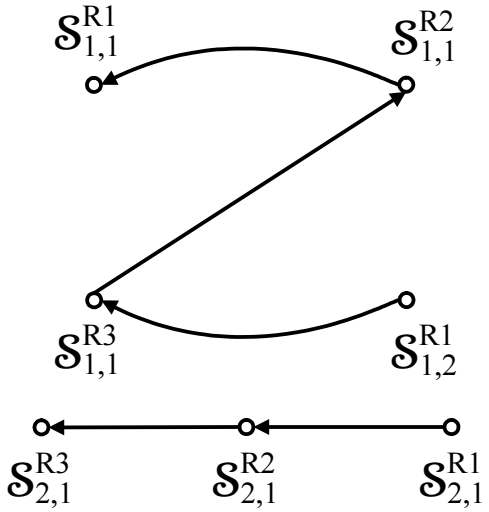



**Figure 3.10**: A SOAR$^2$ net with two process subnets. The set of contained s-regs is depicted in shadowed boxes

important property is that the idle place cannot belong to any s-reg because the idle place cannot be a holder place of a resource (recall that the SOAR$^2$ class is a strict subclass of the S$^4$PR class of nets). The set of all s-regs of a net will be denoted as $\tilde{\mathcal{S}}$, and the set of s-regs associated to a resource $r$ will be denoted as $\tilde{\mathcal{S}}^r$

The first important relation between s-regs is the overlapping relation between s-regs. We say that a set of s-regs $A \subseteq \tilde{\mathcal{S}}$, $A \neq \emptyset$ is a set of overlapping s-regs if and only if $\bigcap_{\mathcal{S}\in A} \mathcal{S} \neq \emptyset$. That is, the set of places belonging to the intersection of a set of s-regs are simultaneously holder places of all resource places associated to the s-regs of the set. In other words, the places of the intersection are process places and a token inside one of these places (a process) is using a copy of each resource associated to the s-regs of the set. Therefore, the computation of the maximal sets of overlapping s-regs is important for the study of the ordering relations in the allocation and release of the set of resources that simultaneously are held by a process inside one of the places in the intersection. For example, for the upper process subnet in Fig. 3.10 there is only one maximal set of s-regs: $\{\mathcal{S}^{R1}_{1,1}, \mathcal{S}^{R2}_{1,1}, \mathcal{S}^{R3}_{1,1}\}$. Observe that, $\mathcal{S}^{R1}_{1,1} \cap \mathcal{S}^{R2}_{1,1} \cap \mathcal{S}^{R3}_{1,1} = \{A3\}$. In other words, when we have a token inside the place A3, the process is using simultaneously the resources R1, R2 and R3. Observe, that

| Maximal set of overlapping s-regs | Order relation |
|---|---|
| $\{\mathcal{S}^{R1}_{1,1}, \mathcal{S}^{R2}_{1,1}\}$ | $\mathcal{S}^{R1}_{1,1} < \mathcal{S}^{R2}_{1,1}$ |
| $\{\mathcal{S}^{R2}_{1,1}, \mathcal{S}^{R3}_{1,1}\}$ | $\mathcal{S}^{R2}_{1,1} < \mathcal{S}^{R3}_{1,1}$ |
| $\{\mathcal{S}^{R1}_{1,2}, \mathcal{S}^{R3}_{1,1}\}$ | $\mathcal{S}^{R3}_{1,1} < \mathcal{S}^{R1}_{1,2}$ |
| $\{\mathcal{S}^{R1}_{2,1}, \mathcal{S}^{R2}_{2,1}, \mathcal{S}^{R3}_{2,1}\}$ | $\mathcal{S}^{R3}_{2,1} < \mathcal{S}^{R2}_{2,1} < \mathcal{S}^{R1}_{2,1}$ |



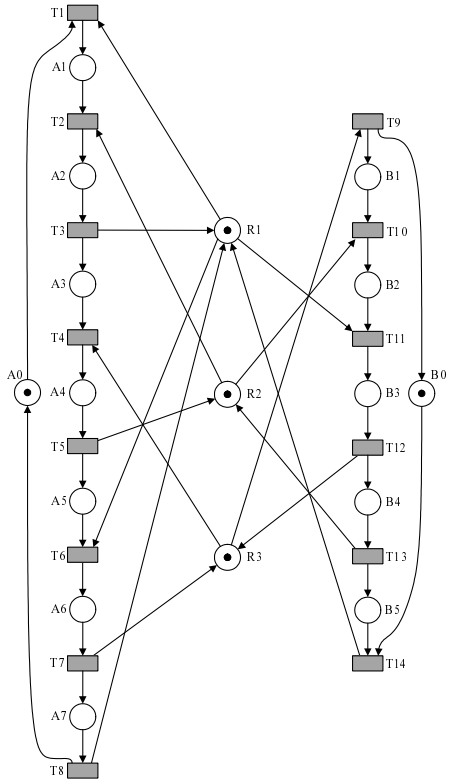




**Figure 3.11**: A SOAR$^2$ net, the maximal sets of overlapping s-regs and the order relation between s-regs

in order to reach this situation the allocation of these resources happened in the order $R1 - R2 - R3$. The moving of the token outside the set of overlapping s-regs release the three resources in the same order: the condition imposed for SOAR$^2$ nets. However, in the bottom process subnet in Fig. 3.10 there are two maximal sets of overlapping s-regs: $\{\mathcal{S}^{R1}_{2,1}, \mathcal{S}^{R2}_{2,1}\}$ and $\{\mathcal{S}^{R2}_{2,1}, \mathcal{S}^{R3}_{2,1}\}$. In these two maximal sets the allocation order of the resources is respected in the release process.

The order relation in the allocation of resources and its further release, induces an order relation on the set of s-regs that is called pruning relation between s-regs. Taking into account that it is a total order relation in a maximal set of overlapping s-regs we are only interested in the maximal chains of ordered elements in the set. This order relation can be represented by means of a graph. In Fig. 3.11 the reader can find a SOAR$^2$ net and in the first column of the table the maximal sets of overlapping s-regs. The second column represents the maximal chains of ordered elements in the

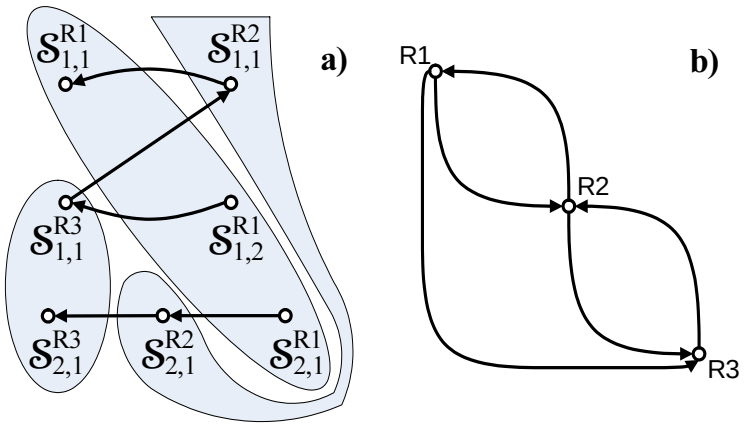



**Figure 3.12**: Pruning graph of the s-regs of the SOAR$^2$ net in Fig. 3.11 and the agglomerated resource pruning graph of the net

set. This ordering relations can be represented in a graphical way in the pruning graph of s-regs.

Observe that the union of all s-regs associated to a same resource gives as result the set of holders of the resource that together with the resource allow to obtain the support of the minimal p-semiflow associated to the resource. The agglomeration, in the pruning graph of s-regs, of all s-regs of a same resource in a single node allows to obtain the pruning graph of resources used to characterise the siphons [CRC12]. This agglomeration operation in the pruning graph of s-regs is illustrated in Fig. 3.12. The left figure represents the pruning graph of s-regs, wherein the nodes inside the shadowed boxes correspond to the s-regs associated to the same resource and are the nodes to be fused in a single node. Therefore each box gives rise to a node of the right figure in Fig. 3.12. The arcs in the right figure have its origin in the order relation (pruning relation) between two s-regs associated to different resources but overlapped.

We must recall that a necessary condition to obtain a siphon containing a given set of resources is that the subgraph of the pruning graph containing the nodes associated to the considered resources and the arcs among them is a strongly connected subgraph. Therefore, if we transform our net in such a way that the resulting pruning graph is acyclic, then we cannot obtain siphons with more than one resource (i.e., the bad siphons of the net causing the non-liveness problems). That is, the acyclic pruning graph obtained after the transformation evidences that the liveness property has been enforced.

**A new deadlock prevention policy based on the specialisation of resources**

In this section we present a new deadlock prevention technique for the previously introduced class of SOAR$^2$ nets. We have presented the advantages of the technique from the point of view of the maintenance (or even increase) of the concurrency of the system and the improvement of the resource utilisation ratio in comparison with other deadlock prevention techniques. Nevertheless, there is another reason for the introduction of this technique: the implementation of the control places added to enforce liveness. In effect, in the methods presented in the previous section, the new control place computed for the liveness enforcing property can be difficult to implement because our new virtual resource is used to control the number of tokens inside a siphon and the arcs of this virtual place can need to be connected to several transitions that are not local. This non-locality of the transitions connected to the new resource place can give rise to new problems with respect to a distributable implementation. The technique we summarise in the following respects this locality principle because the area of intervention in order to enforce the liveness property is constrained to an s-reg and then the implementation issues related to distributability are solved.

The basis of the method to correct the model consists of, in the case we have deadlock states, the transformation of the Petri net in such a way that the pruning graph of resources of the transformed net becomes acyclic. Therefore, and according to the characterisation of the minimal siphons of a S$^4$PR net on the pruning graph of resources, there exist no minimal siphons containing more that one resource since there are not strongly connected sub-graphs containing more that one node. This is the basis of the method because in this way we enforce the net to be live since the Petri net has an acceptable initial marking and there are not bad siphons in the net.

In order to reach this final goal the designer must follow the steps stated in the following for SOAR$^2$ nets:

**Step 1.** *Construction of the pruning graph of the s-regs of the net.* The net in Fig. 3.13, used as illustrative example in this subsection, is a non-live SOAR$^2$ net. In effect, the graph on the right hand of the figure is the pruning graph of resources that, as the reader can observe, contains one cycle. In this case, the cycle exists because of a bad siphon containing the resource places R1 and R2: the minimal siphon $D = \{\mathrm{R1}, \mathrm{R2}, \mathrm{A2}, \mathrm{A3}, \mathrm{B2}, \mathrm{B3}\}$. This siphon is emptied by firing the transitions T1 and T5 from the initial marking. In general, the existence of cycles in the resource pruning graph is only a necessary condition for the existence of bad siphons, but if we can make acyclic the graph then it is ensured that no bad siphon exists, and the resulting net is live. To accomplish this, we first must compute all the s-regs, and after that we must compute the maximal set of overlapping s-regs. This lets us construct the pruning graph of

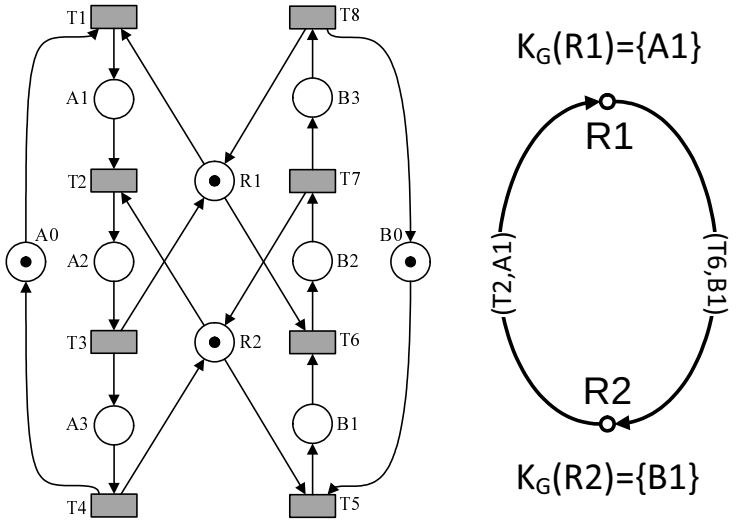



**Figure 3.13**: A non-live SOAR$^2$ net and the corresponding pruning graph of resources of the net

s-regs, which contains four s-regs. The resulting graph is depicted in Fig. 3.14.

**Step 2.** *Construction of the pruning graph of resources of the SOAR$^2$ net from the pruning graph of s-regs.* By aggregating the s-regs which are shadowed in their pruning graph on the right hand of Fig. 3.14, we obtain the pruning graph of resources which is depicted in Fig. 3.13. As already explained in Step 1, if the resulting pruning graph of resources is acyclic, then the net system is live for any acceptable initial marking and no changes are required. Otherwise, the pruning graph of resources provides valuable information on how liveness can be enforced.

**Step 3.** *Computation of a minimal number of arcs of the pruning graph of resources whose removal makes it acyclic.* Essentially, we must strive to select a minimal set of arcs whose removal requires as few changes as possible in the original net. This is often related to the intuitive idea of selecting a set of arcs which involve the smallest possible portion of the original net. Such information can be extracted from the pruning graph of s-regs. In this vein, it must be remarked that a unique arc in the pruning graph of resources may map into several arcs of the pruning graph of s-regs. These arcs would connect different pairs of s-regs involving the same pair of resources. In some cases, the removal of some arc may require the duplication of some places and transitions, so a proper selection of the set of arcs is fundamental to avoid this. Due to the simmetry of the net

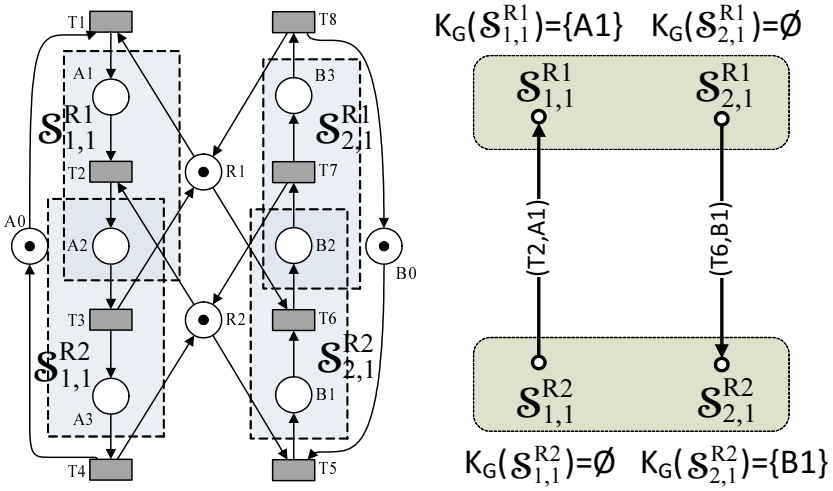



**Figure 3.14**: Structural regions of the SOAR$^2$ net in Fig. 3.13 and the corresponding pruning graph of s-regs of the net

of Fig. 3.13 we can select any of the two arcs of the pruning graph of resources to make it acyclic. In this example, we select the arc drawn from R2 to R1.

**Step 4.** *Removal of the arcs of the pruning graph of resources by the addition of virtual resources between the overlapping s-regs that generate these arcs.* Having obtained a candidate set of arcs, the net is transformed in order to incorporate the new resource places which break the siphons, and the pruning graph of resources is updated accordingly. In the example, the source node R2 in the selected arc is replaced by a new resource place R2'. In the original net, this means that the resource place $R2$ is splitted in such a way that R2' will be used in the old s-reg $\mathcal{S}^{R2}_{1,1}$ (now: $\mathcal{S}^{R2'}_{1,1}$) while R2 will still be used in the s-reg $\mathcal{S}^{R2}_{2,1}$. The result is depicted in Fig. 3.15.

Observe that the concept of s-reg can be generalised beyond the class of SOAR$^2$. Moreover, the order relation over s-regs can be extended in such a way that it is possible to construct a pruning graph of s-regs from which it is possible to obtain the corresponding pruning graph of resources. To complete the picture portrayed in this section, the example of Fig. 3.5 will be reviewed in the light of the new synthesis technique presented, and compared with previous approaches.

Figure 3.16 shows the s-regs for the net of Fig. 3.5. In this case, every s-reg is trivial in the sense that it contains a unique place, and besides they are all disjoint. That is, every maximal set of overlapped s-regs contains a single s-reg, since the net belongs to the S$^3$PR class. Thus, resources are allocated and freed in consecutive transitions

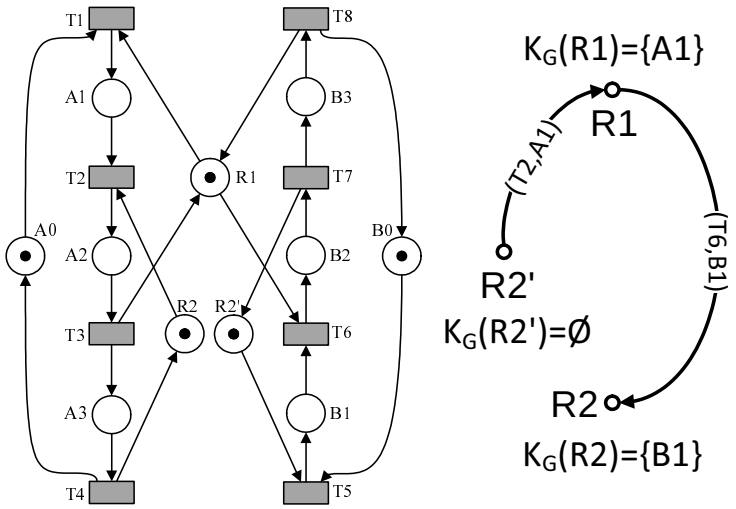



**Figure 3.15**: A live SOAR$^2$ net obtained from the net in Fig. 3.13 by splitting the resource R2 into two resource types used in a private way by each state machine.

and in an exclusive way. This allows an easy extension of both the concept of s-reg and pruning graph of s-regs. Therefore we can informally discuss the application of such techniques over the example, despite falling beyond the SOAR$^2$ class. However, in the general case of PC$^2$R nets the s-regs may extend for several consecutive places of the process subnets and be consequently overlapped in many ways; indeed, PC$^2$R is a strict superclass of SOAR$^2$ (while S$^3$PR is not).

Figure 3.17 depicts the pruning graph of the s-regs of the net in Fig. 3.5, as obtained in Step 1 of the method. Despite the triviality of all maximal sets of overlapped s-regs, some chains can be observed in the graph of the figure. This is because some pairs of s-regs, despite not being overlapped, are still connected by their source and sink transitions, respectively. This applies, for example, for the s-regs $\mathcal{S}^{\mathrm{R1}}_{B,1}$ and $\mathcal{S}^{\mathrm{R2}}_{B,1}$. In this case, there is an arc from $\mathcal{S}^{\mathrm{R2}}_{B,1}$ to $\mathcal{S}^{\mathrm{R1}}_{B,1}$ because both are connected by the transition TB2. I.e., there exists a pruning relation between both s-regs despite not being overlapped.

The shaded areas in Fig. 3.17 represent the nodes that should be merged to build the pruning graph of resources in Step 2. The resulting graph is obviously identical to that already depicted in Fig. 3.7. At this point, the importance of the algorithm of arcs selection used in Step 3 is further evident. As discussed above, only one cycle of the resource pruning graph corresponds to a bad siphon: the one that interconnects the siphons $D_{\mathrm{R1}}$ and $D_{\mathrm{R4}}$ corresponding to the resource places R1 and R4. If the goal of our algorithm is the construction of an acyclic resource pruning graph then the

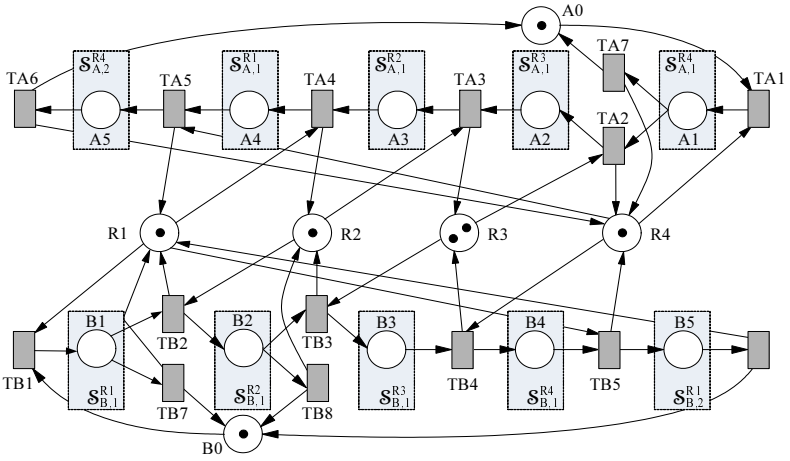
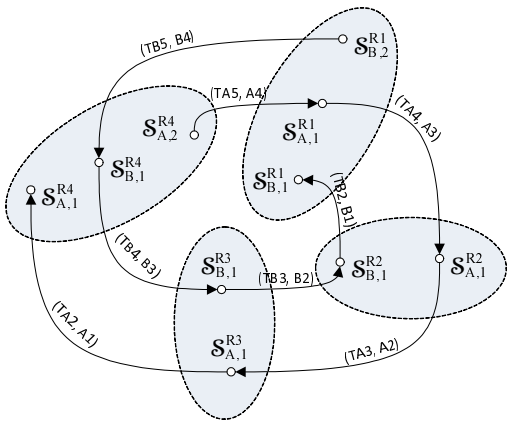



**Figure 3.16**: Structural regions of the net in Fig. 3.5




$$\mathrm{K}_G(\mathcal{S}^{\mathrm{R1}}_{\mathrm{A},1}) = \{\mathrm{A4}\}$$
$$\mathrm{K}_G(\mathcal{S}^{\mathrm{R1}}_{\mathrm{B},1}) = \{\mathrm{B1}\}$$
$$\mathrm{K}_G(\mathcal{S}^{\mathrm{R1}}_{\mathrm{B},2}) = \emptyset$$
$$\mathrm{K}_G(\mathcal{S}^{\mathrm{R2}}_{\mathrm{A},1}) = \{\mathrm{A3}\}$$
$$\mathrm{K}_G(\mathcal{S}^{\mathrm{R2}}_{\mathrm{B},1}) = \{\mathrm{B2}\}$$
$$\mathrm{K}_G(\mathcal{S}^{\mathrm{R3}}_{\mathrm{A},1}) = \{\mathrm{A2}\}$$
$$\mathrm{K}_G(\mathcal{S}^{\mathrm{R3}}_{\mathrm{B},1}) = \{\mathrm{B3}\}$$
$$\mathrm{K}_G(\mathcal{S}^{\mathrm{R4}}_{\mathrm{A},1}) = \{\mathrm{A1}\}$$
$$\mathrm{K}_G(\mathcal{S}^{\mathrm{R4}}_{\mathrm{A},2}) = \emptyset$$
$$\mathrm{K}_G(\mathcal{S}^{\mathrm{R4}}_{\mathrm{B},1}) = \{\mathrm{B4}\}$$

**Figure 3.17**: Pruning graph of the s-regs of the net in Fig. 3.5

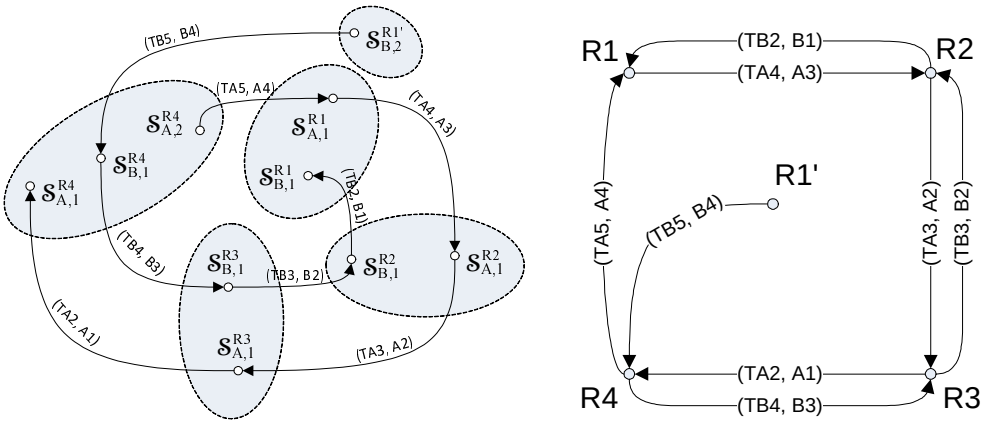



**Figure 3.18**: Pruning graphs of the s-regs and resources of the net in Fig. 3.5 after introducing resource place *R1′*

algorithm necessarily must select at least three different arcs between distinct pairs of connected nodes, where two of these arcs would go in clockwise direction, while the other would go in anticlockwise direction (or vice versa). Actually, this would be the solution obtained by applying an analogous control policy to the one developed in the PhD thesis of C. A. Rovetto [Rov11]. The final result would be the splitting of three resource places.

However, this example shows that this policy can be optimised in certain situations, as long as the splitting of the fewest possible resource places is considered as a target. Indeed, in this case it would be sufficient to break the cycle between R1 and R4 by the selection of one of the two arcs between them in Step 3; for example, the arc from R1 to R4. Figure 3.18 shows the resulting pruning graphs after the addition of a new resource place (R1') that privatises the use of a resource of type R1 in the last operation of the processes of type A[1].

Figure 3.19 presents the net after the transformation. Following the addition of this resource place, there is no longer any cycle in the pruning graph of resources originating a bad siphon, and the system is live. Figure 3.20 illustrates the reachability graph of the resulting net. As can be seen, the concurrence of the system has not been weakened, in contradistinction to the net of Fig. 3.8 (i.e., the system controlled through the classic approach of addition of monitor places).

[1]Processes of type A are those that traverse the left-hand subnet of the figure.

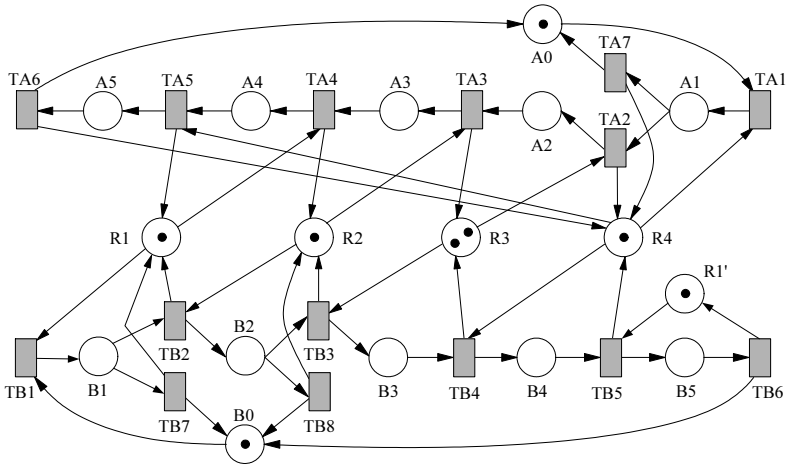



**Figure 3.19**: Controlled version of the net in Fig. 3.5

## 3.3 Liveness analysis of PC$^2$R models through siphons

### 3.3.1 Towards a liveness characterisation of PC$^2$R models

So far, there have been presented a number of synthesis results for real-world systems abstracted and modelled through certain subclasses of PC$^2$R: particularly through the S$^4$PR subclass and subclasses of the latter (e.g., SOAR$^2$). All these techniques are based on a well-known result that has already been presented and discussed in previous chapters. Theorem 1.5 characterises the liveness of a S$^4$PR net system by way of the reachability of a marking in which there exists an insufficiently marked siphon. This result has also been presented from a different perspective through Theorem 1.4. This last theorem presents liveness characterised by the reachability of an equivalent marking condition that does not use (insufficiently marked) siphons yet ultimately explains how these are constructed from it.

Throughout Chapter 2 it has been evidenced that such characterisations do not apply in the case of more general nets as those belonging to the S$^5$PR subclass or the PC$^2$R class discussed in this thesis. In this section, we discuss the limits of those classic characterisation results when trying to extend them to more general net classes such as those that emerge from the abstraction of the resource allocation logic in multithreaded software systems.

Next, some useful basic terminology that has been previously defined in Subsection 3.2.2 for the S$^4$PR subclass will be extended to the PC$^2$R class. In the following,

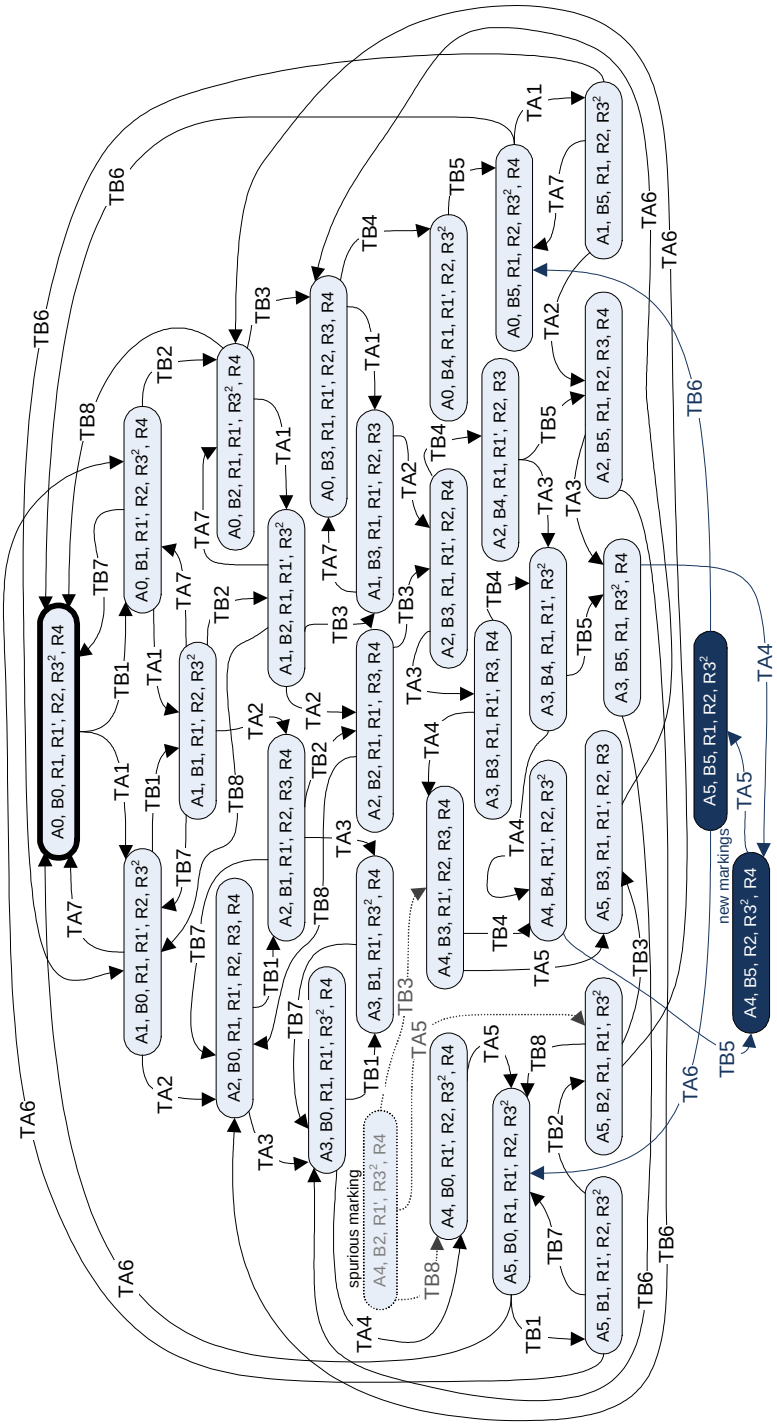



**Figure 3.20**: Reachability graph of the live net system in Fig. 3.19. The new states are stressed on a different colour

for a siphon $D$, $D_{\mathrm{R}} = D \cap P_{\mathrm{R}}$ and:

**Definition 3.8.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a $PC^2R$ net system, and $D$ be a siphon of $\mathcal{N}$ such that $D_{\mathrm{R}} \neq \emptyset$. $\mathcal{T}h_D = \bigcup_{r \in D_{\mathrm{R}}} \|\mathbf{y_r}\| \setminus D$ is the* set of thieves *of $D$, i.e. the set of process places of the net that use resources of the siphon and do not belong to that siphon.*

Similarly, the definition of **m**-process-enabled/disabled and **m**-resource-enabled/disabled transition is generalised, so that, for the $\mathrm{S}^4\mathrm{PR}$ subclass, the new definition is consistent with the one previously introduced at Section 1.4.

**Definition 3.9.** *Let $\mathcal{N}$ be a $PC^2R$ net, and $\mathcal{B}_{\mathcal{N}}$ the set of iteration blocks of $\mathcal{N}$. Given a marking $\mathbf{m}$ of $\mathcal{N}$, a transition $t$ is said to be:*

- **m**-*process-enabled iff it has one marked input process place which is not the idle place of the elementary iteration block to which $t$ belongs, i.e. $t \in (\|\mathbf{m}\| \cap P_{\mathrm{S}})^{\bullet}$ and $\nexists (p, P_{\mathrm{SM}}, T_{\mathrm{SM}}) \in \mathcal{B}_{\mathcal{N}} : t \in p^{\bullet} \cap T_{\mathrm{SM}}$ (otherwise, $t$ is said to be* **m**-*process-disabled).*

- **m**-*resource-enabled iff all input resource places have enough tokens to fire it, i.e., $\mathbf{m}[P_{\mathrm{R}}, t] \geq \mathbf{Pre}[P_{\mathrm{R}}, t]$ (otherwise, $t$ is said to be* **m**-*resource-disabled).*

Note that the generalisation of the definition of **m**-process-enabled is done adhering to the spirit of the original definition for $\mathrm{S}^4\mathrm{PR}$ nets. The original definition captures if the input process place of the transition is marked *except* for those transitions that trigger the execution of a minimal t-semiflow: the latter transitions are always **m**-process-disabled. In the case of $\mathrm{S}^4\mathrm{PR}$ nets, the execution of a minimal t-semiflow is triggered when a token is taken out of the idle place. Therefore, the output transitions of the idle places are assumed to be **m**-process-disabled. In the case of $\mathrm{PC}^2\mathrm{R}$ nets, there may exist minimal t-semiflows which do not traverse the idle place. Therefore, the approach is generalised so that all transitions that trigger the execution of a minimal t-semiflow within an elementary iteration block are permanently **m**-process-disabled. Recall that the idle place of an elementary iteration block is not necessarily the idle place of the corresponding process subnet. Obviously, for $\mathrm{S}^4\mathrm{PR}$ nets, both definitions are equivalent, since the process subnets have one unique elementary iteration block.

Traditionally, the concepts of **m**-process-enabled/disabled transition and **m**-resource-enabled/disabled transition have been tools primarily used to formulate a liveness characterisation that applies to the $\mathrm{S}^4\mathrm{PR}$ subclass (Theorem 1.4). This characterisation relates liveness with the reachability of a certain type of marking which ensures that some active processes are permanently blocked for any future progression. The generalisation of these instruments for the $\mathrm{PC}^2\mathrm{R}$ class leads to a condition for liveness that is sufficient, yet, as we shall see later, not necessary:

**Theorem 3.10.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, be a PC²R net system with a 1-acceptable initial marking. If $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live then $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that the set of* **m***-process-enabled transitions is non-empty, and every* **m***-process-enabled transition is* **m***-resource-disabled.*

*Proof.* If $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live then there exist a firing sequence $\sigma$, a reachable marking $\mathbf{m_1} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, and a transition $t \in T$ such that $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m_1}$ and $t$ is dead at $\langle \mathcal{N}, \mathbf{m_1} \rangle$.

Let $\mathcal{B}_\mathcal{N}$ be the set of the elementary iteration blocks of $\mathcal{N}$, and let $T_{\text{trigger}}$ be the set of transitions that trigger an elementary iteration block of $\mathcal{N}$, i.e. $T_{\text{trigger}} = \bigcup_{(p, P_{\text{SM}} T_{\text{SM}}) \in \mathcal{B}_\mathcal{N}} p^\bullet \cap T_{\text{SM}}$. Now we start at $\langle \mathcal{N}, \mathbf{m_1} \rangle$ and, as far as there exist enabled transitions in the set $T \backslash T_{\text{trigger}}$, we select one of them (any) and fire it. We successively repeat this operation until no transition is firable. Since the subnet induced by $P, T \setminus T_{\text{trigger}}$ is acyclic and every path ends in $P_0$, this eventually happens, and a marking $\mathbf{m}$ is reached such that either no token remains in the process places, i.e. $\mathbf{m}[P_{\text{S}}] = \mathbf{0}$, or for each token remaining, the output transitions of its process place are disabled because of the lack of some resource(s).

In the first case, $\mathbf{m_0} = \mathbf{m}$. Then, by Lemma 2.31, there would exist a firing sequence $\sigma'$, $\mathbf{m_0} \xrightarrow{\sigma'}$, containing the dead transition $t$, which is impossible. Therefore, only the second case is possible, i.e., the set of **m**-process-enabled transitions is non-empty, and every **m**-process-enabled transition is **m**-resource-disabled. □

Theorem 3.10 works for the non-live net system in Fig. 2.13. That net is an S⁵PR net with an acceptable initial marking, and therefore it also is a PC²R with a 1-acceptable initial marking (indeed, it is 0-acceptable as well). In this case, the only reachable marking which holds the condition in Theorem 3.10 is $[\mathrm{A1}, \mathrm{B1}, \mathrm{BOWL}]$.

However, the live net system in Fig. 3.21 proves that the reverse of Theorem 3.10 is not true in general for PC²R nets (in spite of being true for the S⁴PR subclass). This PC²R net with a 1-acceptable initial marking has one reachable marking that holds the condition in Theorem 3.10, yet the system is live. That marking is $[\mathrm{A1}, \mathrm{B1}, \mathrm{C0}, \mathrm{R3}]$.

In fact, the reverse of Theorem 3.10 does not hold even for the S⁵PR subclass. The S⁵PR net in Fig. 2.16 has an acceptable initial marking and is live. However, each one of the **m**-process-enabled transitions is **m**-resource-disabled at the marking $\mathbf{m} \equiv [\mathrm{A0}, \mathrm{A1}, \mathrm{B0}, \mathrm{B1}, \mathrm{R1}, \mathrm{R3}, \mathrm{R4}, \mathrm{R5}, \mathrm{R6}, \mathrm{R7}, \mathrm{R10}, \mathrm{R11}, \mathrm{R13}]$. Indeed, the only **m**-process-enabled transitions are T11 and T26, and both of them are **m**-resource-disabled. Note that transitions T2 and T14 are **m**-process-disabled since their respective input process places are the idle places of the elementary iteration blocks T2 and T14 belong to.

Unfortunately, the condition is neither necessary nor sufficient in general for PC²R nets with a 0-acceptable-initial marking. The non-live PC²R net system depicted in Fig. 2.7 reveals that we can have transitions which are dead from the initial marking,

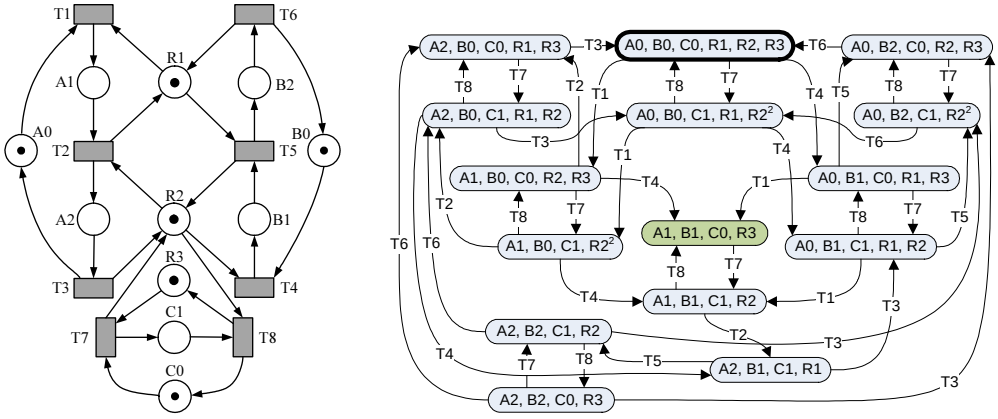



**Figure 3.21**: A live PC$^2$R net system within 'the gap': it holds the condition of Theorem 3.10. Every $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource disabled at $\mathbf{m} \equiv [\mathrm{A1}, \mathrm{B1}, \mathrm{C0}, \mathrm{R3}]$

yet the system is reversible and no marking is reachable such that every $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled.

Historically, Petri net-based synthesis techniques for RASs in FMSs have been based on the addition of GMECs that prevent bad markings such as those described in Theorem 3.10 from being reachable; thus obtaining a live system. This is the kind of approach described in Sect. 3.2.2 for S$^4$PR nets. In that case, bad markings are searched for among the superset of potentially reachable markings described by the net state equation. This implies that occasionally new GMECs may be added to the system when it is already live. However, two important aspects are present in the synthesis algorithm [Tri03]: (i) Liveness is monotonic with respect to the iterations of the algorithm: if the net is already live, any newly proposed GMECs does not make it non-live, and (ii) The algorithm always converges in a system with no 'bad' potentially reachable markings. And Theorem 3.10 grants that a system with no 'bad' (potentially) reachable markings is live.

Unfortunately, an analogous synthesis approach in the context of PC$^2$R nets would introduce at least one new problem. The S$^5$PR net in Fig. 3.22 illustrates this. In this case, every $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled for the reachable marking $\mathbf{m} \equiv [\mathrm{A2}^2, \mathrm{B1}, \mathrm{R3}]$, but the net system is live. Introducing the GMEC $\mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{B1}] \leq 2$ through a virtual resource (i.e., a monitor place) would turn it into a non-live system. So liveness is no longer monotonic with respect to the addition of such kind of GMECs.

In the context of S$^4$PR net subclass, the characterisation of Theorem 1.4 is also used to formally prove Theorem 1.5 [Tri03]. This last theorem relates the system

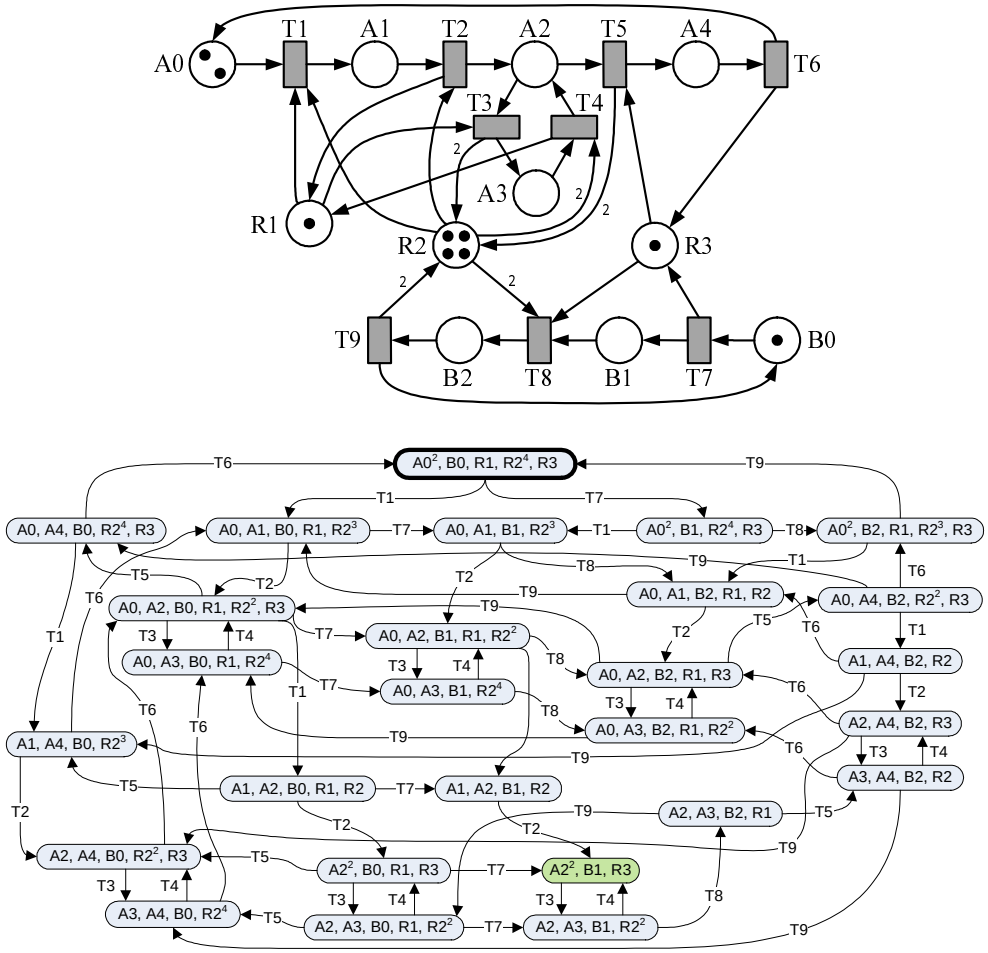



**Figure 3.22**: A live $S^5PR$ net system within 'the gap'. The marking in a different colour is the only one which holds the condition of Theorem 3.10

liveness with the existence of reachable markings with insufficiently marked siphons. That theorem can be rewritten as follows:

**Theorem 3.11.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a $S^4PR$ net system with an acceptable initial marking. $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live iff $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ and a siphon $D$ such that:*

1. *There exists at least one marked thieve place of $D$ at $\mathbf{m}$, i.e., $\|\mathbf{m}\| \cap \mathcal{T}h_D \neq \emptyset$;*
2. *Every output transition of a marked thieve place of $D$ is $\mathbf{m}$-resource-disabled by resource places in $D$, i.e., $\forall t \in (\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet : \exists r \in D_\mathrm{R} : \mathbf{m}[r] < \mathbf{Pre}[r, t]$;*

*3. Process and idle places in $D$ are empty at $\mathbf{m}$, i.e., $\|\mathbf{m}\| \cap D \subseteq P_{\mathrm{R}}$.*

*Proof.* $\Longrightarrow$) Fernando Tricas proved that there exists a reachable marking $m$ and siphon $D$ fulfilling the three conditions of Theorem 1.5, with $D = D_{\mathrm{R}} \cup D_{\mathrm{S}}$:

- $D_{\mathrm{R}} = \{r \in P_{\mathrm{R}} \mid \exists t \in r^{\bullet} : \mathbf{m}[r] < \mathbf{Pre}[r,t] \text{ and } \mathbf{m}[{}^{\bullet}t \cap P_{\mathrm{S}}] > 0\} \neq \emptyset$,
- $D_{\mathrm{S}} = D \cap P_{\mathrm{S}} = \{p \in \bigcup_{r \in D_{\mathrm{R}}} \|\mathbf{y}_{\boldsymbol{r}}\| \mid \mathbf{m}[p] = 0\} \neq \emptyset$.

We prove independently the three points of Theorem 3.11:

1. All process places in $D$ are empty at $\mathbf{m}$. Suppose that no thieve place of $D$ is marked at $\mathbf{m}$. Then, $\mathbf{m}[D_{\mathrm{R}}] = \mathbf{m_0}[D_{\mathrm{R}}]$ and, by the definition of acceptable initial marking, $\forall t \in T, p \in t^{\bullet} \cap (P_0 \cup P_{\mathrm{S}}), r \in D_{\mathrm{R}} : \mathbf{m}[r] = \mathbf{m_0}[r] \geq \mathbf{y}_{\boldsymbol{r}}[p] \geq \mathbf{Pre}[r,t]$. Therefore, no transition is $\mathbf{m}$-resource-disabled by resource places in $D$. But this contradicts the fact that, by Theorem 1.5, at least one $m$-process enabled transition exists, and that it must be $\mathbf{m}$-resource-disabled by resource places in $D$. Thus, at least one marked thieve place of $D$ is marked at $\mathbf{m}$.
2. By the definition of thieve place, $\mathcal{T}h_D \subset P_{\mathrm{S}}$. Therefore, every output transition of a marked thieve place of $D$ is $\mathbf{m}$-process-enabled. By Theorem 1.5, all such transitions are $\mathbf{m}$-resource-disabled by resource places in $D$.
3. By the definition of $D_{\mathrm{S}}$, all process places in $D$ are empty at $\mathbf{m}$. Besides, $D \cap P_0 = \emptyset$. Therefore, $\|\mathbf{m}\| \cap D \subseteq P_{\mathrm{R}}$.

$\Longleftarrow$) The proof of Theorem 3.13 could be copied verbatim to prove this part, since $\mathrm{S}^4\mathrm{PR}$ is a subclass of $\mathrm{PC}^2\mathrm{R}$. For the sake of concision, it is omitted here.

□

Next it is proved that this siphon-based characterisation for $\mathrm{S}^4\mathrm{PR}$ nets is necessary for the liveness of $\mathrm{PC}^2\mathrm{R}$ systems, but it is, in general, not sufficient. For this reason, an intuitive lemma is presented first. That lemma holds for any initial marking of a $\mathrm{PC}^2\mathrm{R}$. The lemma is instrumental for the proof of the theorem that follows.

**Lemma 3.12.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a $PC^2R$ net system, and $r \in P_{\mathrm{R}}$. Let $\mathbf{m}, \mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that $\forall p \in (\|\mathbf{y}_{\boldsymbol{r}}\| \setminus \{r\}) : \mathbf{m}'[p] \geq \mathbf{m}[p]$. Then $\mathbf{m}'[r] \leq \mathbf{m}[r]$.*

*Proof.* From the invariant relation $\mathbf{y}_{\boldsymbol{r}} \cdot \mathbf{m}' = \mathbf{y}_{\boldsymbol{r}} \cdot \mathbf{m_0}$, it can be derived that:

$$\begin{aligned}\mathbf{m}'[r] &= \mathbf{m_0}[r] + \sum_{p \in \|\mathbf{y}_{\boldsymbol{r}}\| \setminus \{r\}} \mathbf{m_0}[p] \cdot \mathbf{y}_{\boldsymbol{r}}[p] - \sum_{p \in \|\mathbf{y}_{\boldsymbol{r}}\| \setminus \{r\}} \mathbf{m}'[p] \cdot \mathbf{y}_{\boldsymbol{r}}[p] \\ &\leq \mathbf{m_0}[r] + \sum_{p \in \|\mathbf{y}_{\boldsymbol{r}}\| \setminus \{r\}} \mathbf{m_0}[p] \cdot \mathbf{y}_{\boldsymbol{r}}[p] - \sum_{p \in \|\mathbf{y}_{\boldsymbol{r}}\| \setminus \{r\}} \mathbf{m}[p] \cdot \mathbf{y}_{\boldsymbol{r}}[p] = \mathbf{m}[r]\end{aligned}$$

□

The condition of Theorem 3.11 for the S⁴PR class is now evaluated for the PC²R class in general.

**Theorem 3.13.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a PC²R net system with a 0-acceptable initial marking. If $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is live then there does not exist a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ and a siphon $D$ such that:*

1. *There exists at least one marked thieve place of $D$ at $\mathbf{m}$, i.e., $\|\mathbf{m}\| \cap \mathcal{T}h_D \neq \emptyset$;*
2. *Every output transition of a marked thieve place of $D$ is $\mathbf{m}$-resource-disabled by resource places in $D$, i.e., $\forall t \in (\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet : \exists r \in D_\mathrm{R} : \mathbf{m}[r] < \mathbf{Pre}[r, t]$;*
3. *Process and idle places in $D$ are empty at $\mathbf{m}$, i.e., $\|\mathbf{m}\| \cap D \subseteq P_\mathrm{R}$.*

*Proof.* Proof by contrapositive: Assume that there exists a siphon $D$ and a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ holding the three conditions of the theorem. It will be proved that $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live.

Let $t \in (\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet$. In order to fire $t$ at least some more tokens are needed in some places belonging to ${}^\bullet t \cap D_\mathrm{R}$. Since tokens in the thieve places of $D$ cannot progress at $\mathbf{m}$, the marking of such resources can only be changed by moving some processes from $(\|\mathbf{m}\| \setminus (\mathcal{T}h_D \cup P_\mathrm{R}))^\bullet$. Let $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$. It will be proved, by induction over the length of $\sigma$ that: (i) $\|\sigma\| \cap \mathcal{T}h_D{}^\bullet = \emptyset$, and (ii) $\forall p \in \mathcal{T}h_D : \mathbf{m}'[p] \geq \mathbf{m}[p]$.

Doing so, and since $\forall p \in D \setminus D_\mathrm{R} : \mathbf{m}'[p] \geq \mathbf{m}[p] = 0$ and $\|\mathbf{y_r}\| \setminus \{r\} \subseteq (D \setminus D_\mathrm{R}) \cup \mathcal{T}h_D$, it can be deduced that $\forall r \in D_\mathrm{R}, p \in (\|\mathbf{y_r}\| \setminus \{r\}) : \mathbf{m}'[p] \geq \mathbf{m}[p]$. But then, by Lemma 3.12, $\forall r \in D_\mathrm{R} : \mathbf{m}'[r] \leq \mathbf{m}[r]$. Therefore, no transition of $(\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet$ can be $\mathbf{m}'$-resource-enabled. Such transitions are dead at $\mathbf{m}$, and $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live.

**Case $\sigma = t$.** Since no transition of $\mathcal{T}h_D{}^\bullet$ is enabled at $\mathbf{m}$, then $t \notin \mathcal{T}h_D{}^\bullet$: in fact, $t \in (\|\mathbf{m}\| \setminus (\mathcal{T}h_D \cup P_\mathrm{R}))^\bullet$. On the other hand, if $t \notin {}^\bullet\mathcal{T}h_D$, $\forall p \in \mathcal{T}h_D : \mathbf{m}'[p] = \mathbf{m}[p]$. If $t \in {}^\bullet\mathcal{T}h_D$, let $t^\bullet \cap (P_0 \cup P_\mathrm{S}) = \{q\} \in \mathcal{T}h_D$. In this case, $\mathbf{m}'[q] = \mathbf{m}[q] + 1$ and $\mathbf{m}'[p] = \mathbf{m}[p]$ for every $p \in \mathcal{T}h_D \setminus \{q\}$.

**General case. $\mathbf{m} \xrightarrow{\sigma''} \mathbf{m}'' \xrightarrow{t} \mathbf{m}'$**, where $\sigma''$, $\mathbf{m}''$ verify the induction hypothesis: $\|\sigma''\| \cap \mathcal{T}h_D{}^\bullet = \emptyset$, and $\forall p \in \mathcal{T}h_D : \mathbf{m}''[p] \geq \mathbf{m}[p]$. Since $\forall p \in D \setminus D_\mathrm{R} : \mathbf{m}''[p] \geq \mathbf{m}[p] = 0$ and $\|\mathbf{y_r}\| \setminus \{r\} \subseteq (D \setminus D_\mathrm{R}) \cup \mathcal{T}h_D$, then $\forall r \in D_\mathrm{R}, p \in (\|\mathbf{y_r}\| \setminus \{r\}) : \mathbf{m}''[p] \geq \mathbf{m}[p]$. Taking into account Lemma 3.12, $\forall r \in D_\mathrm{R} : \mathbf{m}''[r] \leq \mathbf{m}[r]$. Consequently, every transition of $(\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet$ is $\mathbf{m}''$-resource disabled, and $t \notin \mathcal{T}h_D{}^\bullet$. Therefore, $\forall p \in \mathcal{T}h_D : \mathbf{m}'[p] \geq \mathbf{m}''[p] \geq \mathbf{m}[p]$, and we can conclude.

□

Theorem 3.13 works for the live system in Fig. 2.18. The net is a PC²R net with a 0-acceptable initial marking (which is *not* 1-acceptable). This net contains four

minimal siphons apart from those induced by the isolated process subnets. However, none but one of these minimal siphons has thieve places:

$$\begin{aligned} D_{\mathrm{R1}} &= \{\mathrm{R1}, \mathrm{A0}, \mathrm{B1}\} & \mathcal{T}h_{D_{\mathrm{R1}}} &= \emptyset \\ D_{\mathrm{R2}} &= \{\mathrm{R2}, \mathrm{A1}, \mathrm{B2}\} & \mathcal{T}h_{D_{\mathrm{R2}}} &= \emptyset \\ D_{\mathrm{R3}} &= \{\mathrm{R3}, \mathrm{A2}, \mathrm{B0}\} & \mathcal{T}h_{D_{\mathrm{R3}}} &= \emptyset \\ D_{\mathrm{R1,R2,R3}} &= \{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}\} & \mathcal{T}h_{D_{\mathrm{R1,R2,R3}}} &= \{\mathrm{A0}, \mathrm{A1}, \mathrm{A2}, \mathrm{B0}, \mathrm{B1}, \mathrm{B2}\} \end{aligned}$$

As a result, all siphons of the net that contain at least one thieve place are constructed by adding process or idle places to $D_{\mathrm{R1,R2,R3}}$. Therefore they also contain the three resource places of the net (i.e., R1, R2, R3). The reachability graph in Fig. 2.18 reveals that for every reachable marking there exists an output transition of one of these resource places such that it is enabled (therefore: **m**-resource-enabled). Then Theorem 3.13 trivially holds.

Despite the condition of Theorem 3.13 is necessary and sufficient for the $\mathrm{S}^4\mathrm{PR}$ subclass (see Theorem 3.11), note that it is not a sufficient condition of liveness for general $\mathrm{PC}^2\mathrm{R}$ nets. The non-live $\mathrm{PC}^2\mathrm{R}$ net system in Fig. 3.23 proves that. In this case, transition T2 is no longer firable from the markings of the livelock (i.e., from the darkened markings in the figure). However, there exists no insufficiently marked siphon holding Theorem 3.13.

The net has the following set of minimal siphons (excluding those induced by the isolated process subnets):

$$\begin{aligned} D_{\mathrm{R1}} &= \{\mathrm{R1}, \mathrm{A1}, \mathrm{A2}, \mathrm{B0}, \mathrm{C2}\} & \mathcal{T}h_{D_{\mathrm{R1}}} &= \emptyset \\ D_{\mathrm{R2}} &= \{\mathrm{R2}, \mathrm{A2}, \mathrm{B2}, \mathrm{C0}, \mathrm{C2}\} & \mathcal{T}h_{D_{\mathrm{R2}}} &= \{\mathrm{A1}, \mathrm{B1}\} \\ D_{\mathrm{R3}} &= \{\mathrm{R3}, \mathrm{A0}, \mathrm{A2}, \mathrm{B1}, \mathrm{B2}, \mathrm{C1}\} & \mathcal{T}h_{D_{\mathrm{R3}}} &= \emptyset \\ D_{\mathrm{R1,R2}} &= \{\mathrm{R1}, \mathrm{R2}, \mathrm{A2}, \mathrm{C0}, \mathrm{C2}\} & \mathcal{T}h_{D_{\mathrm{R1,R2}}} &= \{\mathrm{A1}, \mathrm{B1}, \mathrm{B2}\} \\ D_{\mathrm{R1,R3}} &= \{\mathrm{R1}, \mathrm{R3}, \mathrm{A2}, \mathrm{B1}, \mathrm{C2}\} & \mathcal{T}h_{D_{\mathrm{R1,R3}}} &= \{\mathrm{A0}, \mathrm{A1}, \mathrm{B0}, \mathrm{B2}, \mathrm{C1}\} \\ D_{\mathrm{R2,R3}} &= \{\mathrm{R2}, \mathrm{R3}, \mathrm{A2}, \mathrm{B2}, \mathrm{C2}\} & \mathcal{T}h_{D_{\mathrm{R2,R3}}} &= \{\mathrm{A0}, \mathrm{A1}, \mathrm{B1}, \mathrm{C0}, \mathrm{C1}\} \\ D_{\mathrm{R1,R2,R3}} &= \{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{A2}, \mathrm{C2}\} & \mathcal{T}h_{D_{\mathrm{R1,R2,R3}}} &= (P_0 \cup P_{\mathrm{S}}) \setminus D_{\mathrm{R1,R2,R3}} \end{aligned}$$

No siphon containing the minimal siphons $D_{\mathrm{R1}}$ or $D_{\mathrm{R3}}$ contains any thieve place. All of these can be dismissed since they infringe point 1 of the condition in Theorem 3.13. The set of siphons (be they minimal or not) having at least one thieve place is: $\Delta = \{D \subsetneq P \mid \exists D_{\min} \in \{D_{\mathrm{R2}}, D_{\mathrm{R1,R2}}, D_{\mathrm{R1,R3}}, D_{\mathrm{R2,R3}}, D_{\mathrm{R1,R2,R3}}\} : (D_{\min} \subseteq D) \wedge (D_{\min} \cap P_{\mathrm{R}} = D_{\mathrm{R}}) \wedge (\mathcal{T}h_{D_{\min}} \not\subseteq D)\}$. Note that we use the previous concise notation in order to avoid enumerating the (rather large) whole set.

Now we evaluate the condition in Theorem 3.13 with every possible pair ($\mathbf{m}$, $D$), where $\mathbf{m}$ is a reachable marking belonging to the livelock and $D$ a siphon belonging

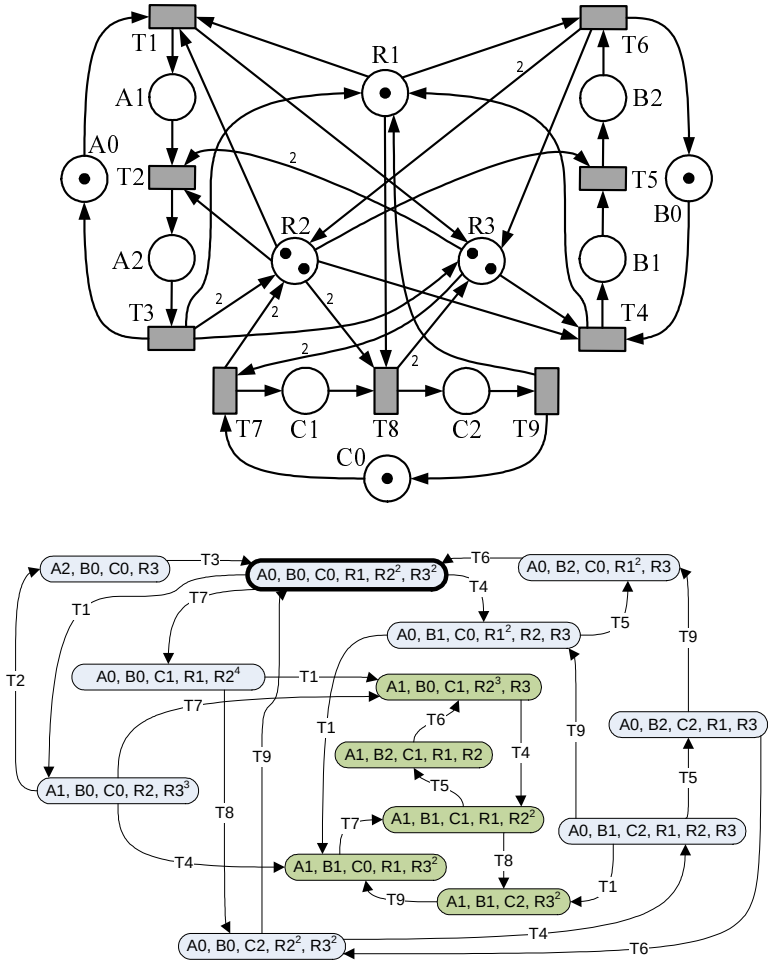



**Figure 3.23**: A non-live $PC^2R$ net system within 'the gap': it holds the condition of Theorem 3.13. Transition T2 is dead at the darkened markings

to $\Delta$. If there exists no combination such that the condition is true, then we have proved that the theorem is not sufficient.

Table 3.1 sums up every possible combination ($\mathbf{m}$, $D$). For each marking, the second column describes the subset of siphons of $\Delta$ whose idle and process places are empty at $\mathbf{m}$. For the sake of concision, this subset is described by way of a logical expression. Note that this expression appears above the arrow, if any (i.e., above $\Downarrow$). Bear in mind that if some idle/process place is marked then point 3 of the condition in Theorem 3.13 is infringed. Therefore, those cases do not have to be considered.

The expression in the second column that appears below the arrow (i.e., below $\Downarrow$) is a logical statement that holds for all combinations ($\mathbf{m}$, $D$) considered in the current row of the table. This logical statement trivially infringes the condition stated in Theorem 3.13 (namely, it infringes point 2).

A quick glance at the table reveals that we cannot conclude that the net is non-live (yet it is); therefore, the condition is not sufficient for liveness.

Table 3.1: Evaluating the condition in Theorem 3.13 for the $PC^2R$ net in Fig. 3.23

| Marking $\mathbf{m}$ | Siphons $D \in \Delta$ such that $\|\mathbf{m}\| \cap D \subseteq P_{\mathrm{R}}$ |
|---|---|
| $[\mathrm{A1}, \mathrm{B1}, \mathrm{C0}, \mathrm{R1}, \mathrm{R3}^2]$ | $D_{\mathrm{R2,R3}} \subseteq D \subseteq D_{\mathrm{R2,R3}} \cup \{\mathrm{A0}, \mathrm{B0}, \mathrm{C1}\} \vee$<br>$D_{\mathrm{R1,R2,R3}} \subseteq D \subseteq D_{\mathrm{R1,R2,R3}} \cup \{\mathrm{A0}, \mathrm{B0}, \mathrm{B2}, \mathrm{C1}\}$<br>$\Downarrow$<br>$\forall D:$ T7 $\in (\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet$ and T7 is $\mathbf{m}$-resource enabled |
| $[\mathrm{A1}, \mathrm{B1}, \mathrm{C1}, \mathrm{R1}, \mathrm{R2}^2]$ | $D_{\mathrm{R2}} \subseteq D \subseteq D_{\mathrm{R2}} \cup \{\mathrm{A0}, \mathrm{B0}\} \vee$<br>$D_{\mathrm{R1,R2}} \subseteq D \subseteq D_{\mathrm{R1,R2}} \cup \{\mathrm{A0}, \mathrm{B0}, \mathrm{B2}\} \vee$<br>$D_{\mathrm{R2,R3}} \subseteq D \subseteq D_{\mathrm{R2,R3}} \cup \{\mathrm{A0}, \mathrm{B0}, \mathrm{C0}\} \vee$<br>$D_{\mathrm{R1,R2,R3}} \subseteq D \subseteq D_{\mathrm{R1,R2,R3}} \cup \{\mathrm{A0}, \mathrm{B0}, \mathrm{B2}, \mathrm{C0}\}$<br>$\Downarrow$<br>$\forall D: \{\mathrm{T5}, \mathrm{T8}\} \subseteq (\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet$<br>and both T5 and T8 are $\mathbf{m}$-resource enabled |
| $[\mathrm{A1}, \mathrm{B0}, \mathrm{C1}, \mathrm{R2}^3, \mathrm{R3}]$ | $D_{\mathrm{R2}} \subseteq D \subseteq D_{\mathrm{R2}} \cup \{\mathrm{A0}, \mathrm{B1}\} \vee$<br>$D_{\mathrm{R1,R2}} \subseteq D \subseteq D_{\mathrm{R1,R2}} \cup \{\mathrm{A0}, \mathrm{B1}, \mathrm{B2}\} \vee$<br>$D_{\mathrm{R1,R3}} \subseteq D \subseteq D_{\mathrm{R1,R3}} \cup \{\mathrm{A0}, \mathrm{B2}, \mathrm{C0}\} \vee$<br>$D_{\mathrm{R2,R3}} \subseteq D \subseteq D_{\mathrm{R2,R3}} \cup \{\mathrm{A0}, \mathrm{B1}, \mathrm{C0}\} \vee$<br>$D_{\mathrm{R1,R2,R3}} \subseteq D \subseteq D_{\mathrm{R1,R2,R3}} \cup \{\mathrm{A0}, \mathrm{B1}, \mathrm{B2}, \mathrm{C0}\}$<br>$\Downarrow$<br>$\forall D:$ T4 $\in (\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet$ and T4 is $\mathbf{m}$-resource enabled |

Continued on next page

**Table 3.1 – continued from previous page**

| Marking $\mathbf{m}$ | Siphons $D \in \Delta$ such that $\|\mathbf{m}\| \cap D \subseteq P_{\mathrm{R}}$ |
|---|---|
| $[\mathrm{A1}, \mathrm{B2}, \mathrm{C1}, \mathrm{R1}, \mathrm{R2}]$ | $D_{\mathrm{R1,R2}} \subseteq D \subseteq D_{\mathrm{R1,R2}} \cup \{\mathrm{A0}, \mathrm{B0}, \mathrm{B1}\} \vee$<br>$D_{\mathrm{R1,R3}} \subseteq D \subseteq D_{\mathrm{R1,R3}} \cup \{\mathrm{A0}, \mathrm{B0}, \mathrm{C0}\} \vee$<br>$D_{\mathrm{R1,R2,R3}} \subseteq D \subseteq D_{\mathrm{R1,R2,R3}} \cup \{\mathrm{A0}, \mathrm{B0}, \mathrm{B1}, \mathrm{C0}\}$<br>$\Downarrow$<br>$\forall D:\ \mathrm{T6} \in (\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet$ and T6 is $\mathbf{m}$-resource enabled |
| $[\mathrm{A1}, \mathrm{B1}, \mathrm{C2}, \mathrm{R3}^2]$ | *None*<br>(Every siphon in $\Delta$ contains C2 except $D_{\mathrm{R3}}$,<br>and $D_{\mathrm{R3}}$ contains B1) |

Similarly, the condition is neither sufficient for the $\mathrm{S}^5\mathrm{PR}$ subclass. A counterexample that proves it is that of the postmodern dining philosophers (see Sect. 2.3). For the sake of concision, we take the most simple version of the problem in which there exist only two philosophers. The corresponding net is depicted in Fig. 2.13. The set of minimal siphons (excluding those induced by the isolated process subnets) is enumerated next:

$$
\begin{aligned}
D_{\mathrm{FORK1}} &= \|\mathbf{y}_{\mathrm{FORK1}}\| \\
D_{\mathrm{FORK2}} &= \|\mathbf{y}_{\mathrm{FORK2}}\| \\
D_{\mathrm{BOWL}} &= \|\mathbf{y}_{\mathrm{BOWL}}\| \\
D_{\mathrm{F1,B}} &= \{\mathrm{A2}, \mathrm{A4}, \mathrm{A5}, \mathrm{A6}, \mathrm{B2}, \mathrm{B3}, \mathrm{B4}, \mathrm{B5}, \mathrm{B6}, \mathrm{FORK1}, \mathrm{BOWL}\} \\
D_{\mathrm{F2,B}} &= \{\mathrm{A2}, \mathrm{A3}, \mathrm{A4}, \mathrm{A5}, \mathrm{A6}, \mathrm{B2}, \mathrm{B4}, \mathrm{B5}, \mathrm{B6}, \mathrm{FORK2}, \mathrm{BOWL}\} \\
D_{\mathrm{F1,F2,B}} &= \{\mathrm{A2}, \mathrm{A4}, \mathrm{A5}, \mathrm{A6}, \mathrm{B2}, \mathrm{B4}, \mathrm{B5}, \mathrm{B6}, \mathrm{FORK1}, \mathrm{FORK2}, \mathrm{BOWL}\}
\end{aligned}
$$

Only the three last minimal siphons have at least one thieve place (namely, $\mathcal{T}h_{D_{\mathrm{F1,B}}} = \{\mathrm{A1}, \mathrm{A3}\}$, $\mathcal{T}h_{D_{\mathrm{F2,B}}} = \{\mathrm{B1}, \mathrm{B3}\}$ and $\mathcal{T}h_{D_{\mathrm{F1,F2,B}}} = \{\mathrm{A1}, \mathrm{A3}, \mathrm{B1}, \mathrm{B3}\}$. The set of siphons (be they minimal or not) having at least one thieve place is: $\Delta = \Delta_{\mathrm{F1,B}} \cup \Delta_{\mathrm{F2,B}} \cup \Delta_{\mathrm{F1,F2,B}}$, where:

$$
\begin{aligned}
\Delta_{\mathrm{F1,B}} = \{D \subsetneq P \mid\ & (D_{\mathrm{F1,B}} \subseteq D) \wedge (D_{\mathrm{F1,B}} \cap P_{\mathrm{R}} = D_{\mathrm{R}}) \wedge (\mathcal{T}h_{D_{\mathrm{F1,B}}} \nsubseteq D) \wedge \\
& (\mathrm{B1} \in D \implies \mathrm{B0} \in D)\}, \\
\Delta_{\mathrm{F2,B}} = \{D \subsetneq P \mid\ & (D_{\mathrm{F2,B}} \subseteq D) \wedge (D_{\mathrm{F2,B}} \cap P_{\mathrm{R}} = D_{\mathrm{R}}) \wedge (\mathcal{T}h_{D_{\mathrm{F2,B}}} \nsubseteq D) \wedge \\
& (\mathrm{A1} \in D \implies \mathrm{A0} \in D)\} \text{ and}
\end{aligned}
$$

$$\Delta_{\mathrm{F1,F2,B}} = \{D \subsetneq P \mid (D_{\mathrm{F1,F2,B}} \subseteq D) \wedge (D_{\mathrm{F1,F2,B}} \cap P_{\mathrm{R}} = D_{\mathrm{R}}) \wedge (\mathcal{T}h_{D_{\mathrm{F1,F2,B}}} \nsubseteq D)\}.$$

Table 3.2 is constructed in a similar way to Table 3.1 and reveals that we cannot conclude that the $S^5PR$ net system is non-live. Thus, the condition is also not sufficient for liveness of $S^5PR$ nets.

Table 3.2: Evaluating the condition in Theorem 3.13 for the $S^5PR$ net in Fig. 2.13

| Marking $\mathbf{m}$ | Siphons $D \in \Delta$ such that $\|\mathbf{m}\| \cap D \subseteq P_{\mathrm{R}}$ |
|---|---|
| $[\mathrm{A1}, \mathrm{B1}, \mathrm{BOWL}]$ | $D_{\mathrm{F1,B}} \subseteq D \subseteq D_{\mathrm{F1,B}} \cup \{\mathrm{A0}, \mathrm{A3}, \mathrm{B0}\} \vee$<br>$D_{\mathrm{F2,B}} \subseteq D \subseteq D_{\mathrm{F2,B}} \cup \{\mathrm{A0}, \mathrm{B0}, \mathrm{B3}\} \vee$<br>$D_{\mathrm{F1,F2,B}} \subseteq D \subseteq D_{\mathrm{F1,F2,B}} \cup \{\mathrm{A0}, \mathrm{B0}, \mathrm{A3}, \mathrm{B3}\}$<br>$\Downarrow$<br>$\forall D:$ TA2 $\in (\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet$ or TB2 $\in (\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet$<br>and both TA2 and TB2 are $\mathbf{m}$-resource enabled |
| $[\mathrm{A2}, \mathrm{B1}]$ | *None* (Every siphon in $\Delta$ contains A2) |
| $[\mathrm{A1}, \mathrm{B2}]$ | *None* (Every siphon in $\Delta$ contains B2) |
| $[\mathrm{A3}, \mathrm{B1}, \mathrm{FORK1}]$ | $D_{\mathrm{F1,B}} \subseteq D \subseteq D_{\mathrm{F1,B}} \cup \{\mathrm{A0}, \mathrm{A1}, \mathrm{B0}\} \vee$<br>$D_{\mathrm{F1,F2,B}} \subseteq D \subseteq D_{\mathrm{F1,F2,B}} \cup \{\mathrm{A0}, \mathrm{A1}, \mathrm{B0}, \mathrm{B3}\}$<br>$\Downarrow$<br>$\forall D:$ TA4 $\in (\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet$ and TA4 is $\mathbf{m}$-resource enabled |
| $[\mathrm{A1}, \mathrm{B3}, \mathrm{FORK2}]$ | $D_{\mathrm{F2,B}} \subseteq D \subseteq D_{\mathrm{F2,B}} \cup \{\mathrm{A0}, \mathrm{B0}, \mathrm{B1}\} \vee$<br>$D_{\mathrm{F1,F2,B}} \subseteq D \subseteq D_{\mathrm{F1,F2,B}} \cup \{\mathrm{A0}, \mathrm{A3}, \mathrm{B0}, \mathrm{B1}\}$<br>$\Downarrow$<br>$\forall D:$ TB4 $\in (\|\mathbf{m}\| \cap \mathcal{T}h_D)^\bullet$ and TB4 is $\mathbf{m}$-resource enabled |
| $[\mathrm{A4}, \mathrm{B1}]$ | *None* (Every siphon in $\Delta$ contains A4) |
| $[\mathrm{A1}, \mathrm{B4}]$ | *None* (Every siphon in $\Delta$ contains B4) |

Summing up, a necessary and a sufficient condition for liveness have been established for the $PC^2R$ class through Theorems 3.10 and 3.13. These conditions are again necessary and sufficient (respectively) for the $S^5PR$ subclass. However, they converge in two characterisations of liveness for the $S^4PR$ subclass (Theorems 1.4 and

3.11). Finally, some examples of nets within the gap between the necessary and the sufficient condition have been presented.

### 3.3.2 Liveness of PC²R models with 1-acceptable initial markings

Theorem 3.10 establishes a sufficient condition for the liveness of a $\mathrm{PC}^2\mathrm{R}$ net with a 1-acceptable initial marking. This sufficient condition can be used to define an efficient ILP-based test condition for deciding if further correction may be needed. This test is analogous to the one proposed for $\mathrm{S}^4\mathrm{PR}$ nets in Sect. 3.2.2 (Theorem 3.4). However, the test condition proposed there relied on a siphon-based liveness characterisation. Since the condition proposed in Theorem 3.10 is strictly marking-based (no siphons are used) the set of restrictions of the resulting ILPP slightly differs.

The following proposition is instrumental to the proposal of the test condition. It is worth mentioning that the proof of the proposition is almost literally that presented by F. Tricas for a rather similar proposition proposed for $\mathrm{S}^4\mathrm{PR}$ nets [Tri03].

**Proposition 3.14.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an PC²R net system with a 1-acceptable-initial marking, and $T_{\text{trigger}}$ be the set of transitions that trigger an elementary iteration block of $\mathcal{N}$, i.e., $T_{\text{trigger}} = \bigcup_{\forall (p, P_{\text{SM}}, T_{\text{SM}}) \in \mathcal{B}_{\mathcal{N}}} p^\bullet \cap T_{\text{SM}}$. If the net is non-live, then there exists a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that the following set of inequalities has, at least, one solution:*

$$
\begin{cases}
\mathbf{m}[P_{\text{S}}] \not\geq \mathbf{0} & & \texttt{-- } \exists t \in T : t \texttt{ is m-p-e} \\
\forall t \in T \setminus T_{\text{trigger}} : & \text{with } \{p\} = {}^\bullet t \cap P_{\text{S}}, & \\
& \mathbf{m}[p] \geq e_t & \texttt{-- } e_t\texttt{=0} : t \texttt{ is m-p-d} \\
& e_t \geq \frac{\mathbf{m}[p]}{\mathbf{sb}[p]} & \texttt{-- } e_t\texttt{=1} : t \texttt{ is m-p-e} \\
\forall r \in P_{\text{R}}, \forall t \in r^\bullet \setminus T_{\text{trigger}} : & \frac{\mathbf{m}[r]}{\mathbf{Pre}[r,t]} \geq e_{rt} & \texttt{-- } e_{rt}\texttt{=0} : t \texttt{ is m-r-d by } r \\
& e_{rt} \geq \frac{\mathbf{m}[r] - \mathbf{Pre}[r,t] + 1}{\mathbf{m_0}[r] - \mathbf{Pre}[r,t] + 1} & \texttt{-- } e_{rt}\texttt{=1} : t \texttt{ is m-r-e by } r \\
\forall t \in T \setminus T_{\text{trigger}} : & \sum_{r \in {}^\bullet t \cap P_{\text{R}}} e_{rt} < |{}^\bullet t \cap P_{\text{R}}| + 1 - e_t & \\
& \texttt{-- if } t \texttt{ is m-p-e then } t \texttt{ is m-r-d} & \\
\forall t \in T \setminus T_{\text{trigger}} : & e_t \in \{0, 1\} & \\
\forall r \in P_{\text{R}}, \forall t \in r^\bullet \setminus T_{\text{trigger}} : & e_{rt} \in \{0, 1\}. &
\end{cases}
\tag{3.3}
$$

*where $\mathbf{sb}[p]$ denotes the structural bound of $p$ [CS91]*

*Note that* **m-p-e** *(***m-p-d***) stands for* **m***-process-enabled (***m***-process-disabled) and* **m-r-e** *(***m-r-d***) stands for* **m***-resource-enabled (***m***-resource-disabled).*

*Proof.* First of all, let us make some comments about the variables used in these inequalities:

1. For each $t \in T \setminus T_{\text{trigger}}$, $e_t$ indicates whether $t$ is $\mathbf{m}$-process-enabled or not. It follows immediately from the following facts:
   - since $\mathbf{sb}[p] > 0$, $\mathbf{m}[p] > 0$ iff $\mathbf{m}[p]/\mathbf{sb}[p] > 0$, which is equivalent to state that $e_t = 1$ (remember that $e_t \in \{0, 1\}$)
   - $\mathbf{m}[p] = 0$ iff $e_t = 0$
2. For each $r \in P_{\text{R}}$ and $t \in r^\bullet \setminus T_{\text{trigger}}$, $e_{rt}$ indicates whether $t$ is enabled by the resource place $r$ at $\mathbf{m}$:
   - If $t$ is enabled by $r$ at $\mathbf{m}$ (i.e., $\mathbf{m}[r] \geq \mathbf{Pre}[r,t]$), $\mathbf{m}[r]/\mathbf{Pre}[r,t] \geq 1$ and $1 \geq \frac{\mathbf{m}[r]-\mathbf{Pre}[r,t]+1}{\mathbf{m_0}[r]-\mathbf{Pre}[r,t]+1} > 0$; therefore, $e_{rt}$ must be 1.
   - If $t$ is not enabled by $r$ at $\mathbf{m}$ (i.e., $\mathbf{m}[r] < \mathbf{Pre}[r,t]$), $\mathbf{m}[r]/\mathbf{Pre}[r,t] < 1$ and $\frac{\mathbf{m}[r]-\mathbf{Pre}[r,t]+1}{\mathbf{m_0}[r]-\mathbf{Pre}[r,t]+1} \leq 0$; then, $e_{rt}$ must be 0.
3. The system of inequalities without the last one always has a solution, and the value of variables $e_t$ and $e_{rt}$ is determined only by $\mathbf{m}$. Therefore, the existence of a solution of the complete system depends on the last inequality. Two cases can be distinguished:
   - If $t \in T \setminus T_{\text{trigger}}$ is not $\mathbf{m}$-process-enabled, $e_t = 0$ and the inequality for $t$ is trivially fulfilled, because $\sum_{r \in {}^\bullet t \cap P_{\text{R}}} e_{rt} \leq |{}^\bullet t \cap P_{\text{R}}$.
   - If $t \in T \setminus T_{\text{trigger}}$ is $\mathbf{m}$-process-enabled, $e_t = 1$ and the inequality becomes $\sum_{r \in {}^\bullet t \cap P_{\text{R}}} e_{rt} < |{}^\bullet t \cap P_{\text{R}}$. Therefore, there is a solution if, and only if, $\exists r \in {}^\bullet t \cap P_{\text{R}}$ such that $t$ is not enabled by $r$.

If the net is non-live, Theorem 3.10 ensures that there exists a marking $\mathbf{m} \in \text{RS}(\mathcal{N}, \mathbf{m_0})$, with $\mathbf{m}[P_{\text{S}}] > 0$, such that each $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled. This means that there exist places with $e_t = 1$. Since each one of these transitions is $\mathbf{m}$-resource disabled, there exists $r \in {}^\bullet t \cap P_{\text{R}}$ such that $\mathbf{m}[r] < \mathbf{Pre}[r,t]$, and then, $e_{rt} = 0$. In consequence, for these transitions, it holds that: $\sum_{r \in {}^\bullet t \cap P_{\text{R}}} e_{rt} < |{}^\bullet t \cap P_{\text{R}}| + 1 - e_t$. □

The following theorem forms the test condition that can be used to evaluate if the net may need further correction. If the resulting ILPP returns no solution, then the net is live, and no correction is needed to ensure that the resource allocation scheme is safe.

**Theorem 3.15.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an $PC^2R$ net system with a 1-acceptable initial marking, and $T_{\text{trigger}}$ be the set of transitions that trigger an elementary iteration*

*block of $\mathcal{N}$, i.e., $T_{\text{trigger}} = \bigcup_{\forall(p,P_{\text{SM}},T_{\text{SM}})\in\mathcal{B}_{\mathcal{N}}} p^{\bullet} \cap T_{\text{SM}}$. If the net is non-live, then there exists a marking $\mathbf{m} \in \text{PRS}(\mathcal{N}, \mathbf{m_0})$ such that the following set of inequalities has, at least, one solution:*

$$\begin{aligned} &\min \textstyle\sum_{p\in P\setminus P_0} \mathbf{m}[p] \\ &\textit{s.t.} \quad \mathbf{m} = \mathbf{m_0} + \mathbf{C}\cdot\boldsymbol{\sigma} \\ &\qquad \mathbf{m} \geq \mathbf{0}, \boldsymbol{\sigma} \in \mathbb{N}^{|T|} \\ &\qquad \text{System } (3.3) \end{aligned}$$

*Proof.* By Theorem 3.10, if $\langle \mathcal{N}, \mathbf{m_0}\rangle$ is non-live, there exists a marking $\mathbf{m} \in \text{RS}(\mathcal{N}, \mathbf{m_0})$ such that the above set of inequalities has, at least, one solution. Obviously, $\mathbf{m} \in \text{PRS}(\mathcal{N}, \mathbf{m_0})$. □

To the purpose pursued in this section (i.e., merely testing the system liveness), the objective function in Theorem 3.15 is essentially irrelevant. For now we are just interested on the existence (or not) of feasible solutions for the system of restrictions of the ILPP: if none exists, we can state that the model is live. However, that objective function is substantial if some of those solutions were subsequently used to compute structural artifacts to enforce liveness such as, e.g., monitor places preventing those markings from being reachable.

In fact, the technique introduced in Sect. 3.2.2 for the S$^4$PR subclass follows that rationale. The solution space to the set of restrictions of the ILPP in Theorem 3.4 establishes a representative set of undesirable markings. If the net is non-live, the set is non-empty. Meanwhile, the heuristic implied by the objective function draws the attention in some of those markings. This ultimately leads to the successive addition of monitor places which forbids (at least) one of such markings and probably some other reachable markings. Essentially, these monitor places sever those markings off from the reachability space. Depending on the selection of the objective function, it is possible to converge faster to liveness so that the control logic is minimised.

Please note, however, that there exists a dead transition in every solution of the set of restrictions of the ILPP in Theorem 3.4. Therefore, the addition of the monitor place grants that at least one 'bad' marking is removed, unless the optimal solution to the ILPP is an spurious solution of the reachability space[2]. Unfortunately, this is not necessarily the case for the solutions of the ILPP in Theorem 3.15. For instance, the reachable marking $[\text{A1}, \text{B1}, \text{C0}, \text{R3}]$ is obtained as the optimal solution to the ILPP applied to the net of Fig. 3.21. However, no transition is dead at that marking; indeed, the net system is live. This observation is very relevant to the effect of liveness

[2]In case that the optimal solution is a spurious marking then the corresponding monitor place does not remove any reachable marking: i.e., redundant control logic is inserted [Tri03]. Unfortunately, deciding if a solution is reachable is NP-complete for general S$^4$PR models, as we shall see in Chap. 5.

synthesis departing from that solution space. Indeed, the removal of such markings may imply severing off purely 'good' behaviour or even turning a live system into a non-live one.

That said, the objective function of the ILPP above aims at the marking with the least number of tokens in the process places of the net (i.e., a marking with as less active processes in the system as possible). This is obviously just one of the many possible heuristics that could have been chosen. This heuristic is based on our perception of the nature of deadlocks in multithreaded software systems. In many scenarios in the domain, the activation of new threads does not help in unlocking those threads which are **m**-process-enabled, **m**-resource-disabled. We claim here that, in practice, *holder* threads (i.e., those that consume resources when active) are more frequent than *lender* threads (i.e., those that produce them). Under such conditions, the activation of new threads does not resolve the problem. Indeed, the number of free resources is decreased. In those cases, we hope that by preventing those markings from being reached we also get rid of many other feasible solutions, converging faster to the synthesis of a live system. Yet, for the reasons expressed in the previous paragraph, further work is needed to apply the solutions of the ILPP to this aim. Synthesis techniques for the $\mathrm{PC}^2\mathrm{R}$ class are approached in Sect. 3.4.

### 3.3.3 Some properties of siphons in $\mathrm{PC}^2\mathrm{R}$ nets

As seen throughout this section, siphons are still very useful structural tools when studying the liveness of a $\mathrm{PC}^2\mathrm{R}$ net system. This is true even though it is no longer possible to use them to characterise liveness in this type of models. Indeed, the lack of certain types of siphons is sufficient to ensure liveness.

In this subsection some properties concerning siphons are studied, as well as their relation to other structural elements of such models, and particularly, to p-semiflows. The study of these properties allows us to propose techniques that make use of siphons to correct liveness problems in multithreaded software systems.

#### Some useful properties of p-semiflows

Some instrumental properties with respect to the form of p-semiflows are presented below. These properties will be used later to unveil the close relationship between p-semiflows and siphons for this class of Petri nets.

Given a $\mathrm{PC}^2\mathrm{R}$ net, Lemma 2.17 proves that $\mathbf{B}$ is a basis of the left null space of the incidence matrix, where $\mathbf{B}$ is an integer matrix of dimensions $(|P_\mathrm{R}| + |I_\mathcal{N}|) \times |P|$ such that its rows are the set of vectors $\{\mathbf{y}_{\mathbf{S}_i} \mid i \in I_\mathcal{N}\} \cup \{\mathbf{y}_{\boldsymbol{r}}^- \mid r \in P_\mathrm{R}\}$, where $\mathbf{y}_{\boldsymbol{r}}^- = \mathbf{y}_{\boldsymbol{r}} - \sum_{i \in I_\mathcal{N}} \mathbf{y}_{\boldsymbol{r}}[p_{0_i}] \cdot \mathbf{y}_{\mathbf{S}_i}$. This means that every p-semiflow $\mathbf{y}$ can be obtained by a linear combination of this set of vectors. Due to the way $\mathbf{y}_{\boldsymbol{r}}^-$ is defined, the same applies for the set of vectors $\{\mathbf{y}_{\mathbf{S}_i} \mid i \in I_\mathcal{N}\} \cup \{\mathbf{y}_{\boldsymbol{r}} \mid r \in P_\mathrm{R}\}$. The following lemma

determines the parameters of this last linear combination in function of the actual values of $\mathbf{y}[P_0]$ and $\mathbf{y}[P_R]$:

**Lemma 3.16.** *Let $\mathcal{N}$ be a $PC^2R$ net, $\mathbf{y} \in \mathbb{N}^{|P|}$ be a p-semiflow of $\mathcal{N}$.*

$$\mathbf{y} = \sum_{r \in P_R} \mathbf{y}[r] \cdot \mathbf{y_r} - \sum_{i \in I_\mathcal{N}} \mathrm{K}_i \cdot \mathbf{y_{S_i}}$$

*where* $\mathrm{K}_i = \sum_{r \in P_R} \mathbf{y}[r] \cdot \mathbf{y_r}[p_{0_i}] - \mathbf{y}[p_{0_i}]$.

*Proof.* By reordering columns in $\mathbf{B}$ such that the first ones correspond to $P_0 \cup P_R$, and subsequently reordering the rows of $\mathbf{B}$, a matrix of the form $[\mathbf{I}|\mathbf{B'}]$ is obtained, where $\mathbf{I}$ is the identity matrix of dimension $|P_R| + |I_\mathcal{N}|$. Since Lemma 2.17 proves that $\mathbf{B}$ is a basis of the left null space of the incidence matrix:

$$\mathbf{y} = \sum_{r \in P_R} \mathbf{y}[r] \cdot \mathbf{y_r^-} + \sum_{i \in I_\mathcal{N}} \mathbf{y}[p_{0_i}] \cdot \mathbf{y_{S_i}}$$

Substituting $\mathbf{y_r^-} = \mathbf{y_r} - \sum_{i \in I_\mathcal{N}} \mathbf{y_r}[p_{0_i}] \cdot \mathbf{y_{S_i}}$ we obtain:

$$\begin{aligned}
\mathbf{y} &= \sum_{r \in P_R} \mathbf{y}[r] \cdot \mathbf{y_r} - \sum_{r \in P_R} \sum_{i \in I_\mathcal{N}} \mathbf{y}[r] \cdot \mathbf{y_r}[p_{0_i}] \cdot \mathbf{y_{S_i}} + \sum_{i \in I_\mathcal{N}} \mathbf{y}[p_{0_i}] \cdot \mathbf{y_{S_i}} \\
&= \sum_{r \in P_R} \mathbf{y}[r] \cdot \mathbf{y_r} + \sum_{i \in I_\mathcal{N}} \left( \mathbf{y}[p_{0_i}] \cdot \mathbf{y_{S_i}} - \sum_{r \in P_R} \mathbf{y}[r] \cdot \mathbf{y_r}[p_{0_i}] \cdot \mathbf{y_{S_i}} \right) \\
&= \sum_{r \in P_R} \mathbf{y}[r] \cdot \mathbf{y_r} - \sum_{i \in I_\mathcal{N}} \left( \sum_{r \in P_R} \mathbf{y}[r] \cdot \mathbf{y_r}[p_{0_i}] - \mathbf{y}[p_{0_i}] \right) \cdot \mathbf{y_{S_i}}
\end{aligned}$$

□

The following lemma is a direct consequence of the previous result. It refers to the fact that every minimal p-semiflow $\mathbf{y}$ is covered by the union of each minimal p-semiflow $\mathbf{y_r}$ corresponding to a resource place $r$ contained in its support $\|\mathbf{y}\|$, if any. We will later see that a similar property holds for minimal siphons containing at least one resource place (see Lemma 3.24).

**Lemma 3.17.** *Let $\mathcal{N}$ be a $PC^2R$ net, $\mathbf{y} \in \mathbb{N}^{|P|}$ be a minimal p-semiflow of $\mathcal{N}$ such that $\|\mathbf{y}\| \cap P_R \neq \emptyset$. $\|\mathbf{y}\| \subseteq \bigcup_{r \in \|\mathbf{y}\| \cap P_R} \|\mathbf{y_r}\|$.*

*Proof.* Considering the linear decomposition expressed by Lemma 3.16, and the fact that the first component is non-negative (i.e., $\sum_{r \in P_R} \mathbf{y}[r] \cdot \mathbf{y_r} \geq \mathbf{0}$) then the minimality of $\mathbf{y}$ implies that $\forall i \in I_\mathcal{N} : K_i \geq 0$. From this fact, it follows that, despite some non-null components of the various $\mathbf{y_r}$ p-semiflows can be annulled, all positive components of $\mathbf{y}$ must also be positive components of some $\mathbf{y_r}$, i.e., $\forall p \in \|\mathbf{y}\| : \exists r \in \|\mathbf{y}\| \cap P_R : p \in \|\mathbf{y_r}\|$. □

The next trivial lemma formalises the fact that a minimal p-semiflow containing at lease one resource place cannot ever cover every place of a process subnet: at least one idle or process place of each process subnet must be uncovered.

**Lemma 3.18.** *Let $\mathcal{N}$ be a $PC^2R$ net, and $\mathbf{y} \in \mathbb{N}^{|P|}$ be a minimal p-semiflow of $\mathcal{N}$ such that $\|\mathbf{y}\| \cap P_{\mathrm{R}} \neq \emptyset$. For each $i \in I_{\mathcal{N}} : \{p_{0_i}\} \cup P_i \not\subseteq \|\mathbf{y}\|$, i.e., $\exists p \in \{p_{0_i}\} \cup P_i : p \notin \|\mathbf{y}\|$.*

*Proof.* By Definition 2.12, each iterative state machine induces a p-semiflow $\mathbf{y_{S_i}} \in \{0,1\}^{|P|} : \|\mathbf{y_{S_i}}\| = \{p_{0_i}\} \cup P_i$. Since $\mathbf{y}$ is minimal, $\forall i \in I_{\mathcal{N}} : \|\mathbf{y_{S_i}}\| \not\subseteq \|\mathbf{y}\|$. □

### Properties on the uniqueness of siphons

The next set of results explores the uniqueness of the set of minimal siphons with respect to the set of resource places covered. This is an interesting property because it sets an upper bound regarding the number of minimal siphons in the net.

The proof of the following lemma about uniqueness is closely analogous to that presented for the $S^4PR$ subclass [CRC12], with some slight variation.

**Lemma 3.19.** *Let $\mathcal{N}$ be a $PC^2R$ net and $D \subseteq P$ a non-empty minimal siphon of $\mathcal{N}$ containing at least one resource place. $D$ is the unique minimal siphon of $\mathcal{N}$ containing exactly the set of resources $D_{\mathrm{R}} = D \cap P_{\mathrm{R}}$.*

*Proof.* This result will be proven by contradiction. Let us suppose that there exist two minimal siphons $D$ and $D'$ such that $D \neq D'$ and $D_{\mathrm{R}} = D \cap P_{\mathrm{R}} = D' \cap P_{\mathrm{R}} \neq \emptyset$. Let $\tau_0$ be the set of input transitions to the resource places belonging to $D_{\mathrm{R}}$ such that they do not have input places belonging to $D_{\mathrm{R}}$, i.e. $\tau_0 = \{t \in T \mid t \in {}^\bullet D_{\mathrm{R}}, {}^\bullet t \cap D_{\mathrm{R}} = \emptyset\}$. If $\tau_0 = \emptyset$ then $D_{\mathrm{R}}$ is a siphon. But $D_{\mathrm{R}} \subseteq D$ and $D_{\mathrm{R}} \subseteq D'$, contradicting the minimality of $D$ and $D'$. Then $D = D' = D_{\mathrm{R}}$. Otherwise, the set $\tau_0$ must be non-empty. Taking into account that $D$ and $D'$ are minimal siphons of $\mathcal{N}$, for all $t \in \tau_0$ there exists $p \in D \cap (P_{\mathrm{S}} \cup P_0)$ and $p' \in D' \cap (P_{\mathrm{S}} \cup P_0)$ such that $p \in {}^\bullet t$ and $p' \in {}^\bullet t$. But $p$ and $p'$ are process or idle places belonging to the same state machine containing the transition $t$, therefore $p = p'$. Let $D_{\mathrm{S}_0}$ be the set of process or idle places belonging to both siphons that they are input places to the transitions of the set $\tau_0$. Moreover, if $p \in D_{\mathrm{S}_0}$, there exists a resource place $r \in D_{\mathrm{R}}$ such that $p \in \|\mathbf{y_r}\|$. Let $\tau_1$ be the set of input transitions to the places belonging to $D_1 = D_{\mathrm{R}} \cup D_{\mathrm{S}_0}$ such that they do not have input places belonging to $D_1$, i.e., $\tau_1 = \{t \in T \mid t \in {}^\bullet D_1, {}^\bullet t \cap D_1 = \emptyset\}$. Now, two cases must be distinguished:

**$\tau_1 = \emptyset$.** In this case we have proven that for all $t \in {}^\bullet D_1 \implies t \in D_1{}^\bullet$, that is, $D_1$ is a siphon. We have also proven that $D_1 \subseteq D$ and $D_1 \subseteq D'$, contradicting the minimality of $D$ and $D'$. Therefore, $D = D' = D_1$.

$\tau_1 \neq \emptyset$**.** In this case $t \in \tau_1 \implies t \in {}^\bullet D_{\mathrm{S}_0}$, and for the same reasons stated previously for the transitions in $\tau_0$, there exists a process place $q$ belonging to both siphons $D$ and $D'$ such that $q \in {}^\bullet t$. Moreover, if the place $p \in t^\bullet \cap D_{\mathrm{S}_0}$ belongs to the support of the p-semiflow $\mathbf{y}_{\boldsymbol{r}}$, for some $r \in D_{\mathrm{R}}$, then $q \in \|\mathbf{y}_{\boldsymbol{r}}\|$. Therefore, we will obtain the sets $D_{\mathrm{S}_1}$ and $D_2 = D_{\mathrm{R}} \cup D_{\mathrm{S}_0} \cup D_{\mathrm{S}_1}$ as in the case of the set $\tau_0$. We can iterate this procedure a finite number of times, $k$, reaching a set $\tau_k = \emptyset$. In effect, from the iteration $j$ to the iteration $j+1$ we are making a backward propagation from each place $p \in D_{\mathrm{S}_j}$ to a place $q \in D_{\mathrm{S}_{j-1}}$ if and only if $\tau_j \cap {}^\bullet p \neq \emptyset$. Both places $p$ and $q$ belong to the support of a p-semiflow $\mathbf{y}_{\boldsymbol{r}}$ of a resource $r \in D_{\mathrm{R}}$. Therefore, this backward propagation from each place $p \in D_{\mathrm{S}_0}$ always finishes in a place $q \in r^{\bullet\bullet}$ in at most $k$ iterations, where $k$ is the maximal length of a sequence of process and idle places of $\mathbf{y}_{\boldsymbol{r}}$, $r \in D_{\mathrm{R}}$, in the same state machine. In this case we have proven that for all $t \in {}^\bullet D_k : t \in D_k{}^\bullet$, that is, $D_k$ is a siphon. We have also proven that $D_k \subseteq D$ and $D_k \subseteq D'$, contradicting the minimality of $D$ and $D'$. Therefore, $D = D' = D_k$.

□

As a result of Lemma 3.19, the set of minimal siphons of a PC²R net, $\mathcal{N}$, can be partitioned into $|P_{\mathrm{R}}| + 1$ classes. Each one of these classes is characterised by the number of resource places in its siphons. The partition of the set $\mathcal{D}$ of minimal siphons of $\mathcal{N}$ is denoted as $\mathcal{D} = \bigcup_{i=0}^{|P_{\mathrm{R}}|} \mathcal{D}^i$; where $\mathcal{D}^i$ is the subset of minimal siphons that contain exactly $i$ resource places; and for all $i, j \in \{0, 1, ..., |P_{\mathrm{R}}|\}$, with $i \neq j$, $\mathcal{D}^i \cap \mathcal{D}^j = \emptyset$.

This partition is consistent with the one devised for S⁴PR nets [CRC12]. Indeed, as it happened with the partition of the minimal siphons of a S⁴PR net, the fact that a class $\mathcal{D}^i$ is empty does not necessarily imply that there all 'higher' classes $\mathcal{D}^j$, $j > i$, are empty. This result is straightforward since PC²R is a superclass of S⁴PR [CRC12].

Class $\mathcal{D}^0$ is completely characterised by Definition 2.12, so that it does not need to be computed. Indeed, the only minimal siphons containing zero resources are the respective sets of places of each process subnet, i.e., $\mathcal{D}^0 = \{P_i \cup \{p_{0_i}\} \mid i \in I_{\mathcal{N}}\}$.

In the end, the bounds in the number of minimal siphons set for the S⁴PR subclass [CRC12] remain valid for the more general PC²R net class:

**Lemma 3.20.** *Let $\mathcal{N}$ be PC²R net. The number of minimal siphons of $\mathcal{N}$,* $\mathrm{n_D}$*, satisfies,* $\mathrm{n_I} + \mathrm{n_R} \leq \mathrm{n_D} \leq \mathrm{n_I} + 2^{\mathrm{n_R}} - 1$*, where* $\mathrm{n_I}$ *is the number of strongly connected state machines of $\mathcal{N}$, and* $\mathrm{n_R} = |P_{\mathrm{R}}|$.

Previous examples of nets reaching these bounds [CRC12] are also valid in this context, as PC²R is a superclass of S⁴PR. These bounds relate the number of minimal siphons with that of resource places in the net. This is an interesting fact since the

number of potentially conflicting resources may be rather constricted when dealing with real world situations in the context of multithreaded software systems.

### Properties on the composability of siphons

The following set of results explore the fact that every minimal siphon $D$ of a $\mathrm{PC}^2\mathrm{R}$ net can be constructed starting from the set of siphons in $\mathcal{D}^1$ which share a resource place with $D$. In other words, the union of such siphons always contains the siphon. However, this union may contain a non-empty set of non-essential places that must be pruned in order to obtain the minimal siphon $D$.

Therefore, in some sense we can say that the set of siphons in $\mathcal{D}^1$ can be used to *compose* every minimal siphon in the net, except for those trivial minimal siphons in $\mathcal{D}^0$. This result is exploited in the construction of the resource pruning graph of $\mathrm{S}^4\mathrm{PR}$ nets [CRC12], which is introduced in Subsection 3.2.3. It is not difficult to derive, from the results ahead, an analogous method to construct a resource pruning graph of $\mathrm{PC}^2\mathrm{R}$ nets.

The next technical result states that every siphon in $\mathcal{D}^1$ excludes at least one place from each process subnet. The result about the composability of every minimal siphon starting from those siphons in $\mathcal{D}^1$ follows immediately below.

**Lemma 3.21.** *Let $\mathcal{N}$ be a $PC^2R$ net, and $D_r \subseteq P$ a non-empty minimal siphon of $\mathcal{N}$ such that $D_r \in \mathcal{D}^1$. For every $i \in I_{\mathcal{N}}$, $(\{p_{0_i}\} \cup P_i) \setminus D_r \neq \emptyset$.*

*Proof.* By Lemma 3.18, $(\{p_{0_i}\} \cup P_i) \setminus \|\mathbf{y_r}\| \neq \emptyset$. And by Lemma 3.23, $D_r \subseteq \|\mathbf{y_r}\|$. Therefore, $(\{p_{0_i}\} \cup P_i) \setminus D_r \neq \emptyset$. □

**Lemma 3.22.** *Let $\mathcal{N}$ be a $PC^2R$ net, and $D \subseteq P$ a minimal siphon of $\mathcal{N}$ such that $D_{\mathrm{R}} = D \cap P_{\mathrm{R}} \neq \emptyset$. $D \subseteq \bigcup_{r \in D_{\mathrm{R}}} D_r$, $D_r \in \mathcal{D}^1$.*

*Proof.* $D' = \bigcup_{r \in D_{\mathrm{R}}} D_r$ is a siphon because each $D_r$ also is. Let us suppose that $D \not\subseteq D'$. Then $\exists p_1 \in D \setminus D'$ and $p_1 \in P_0 \cup P_{\mathrm{S}}$ (otherwise, $p_1 \in D \cap P_{\mathrm{R}} = D_{\mathrm{R}} \subseteq D'$). Besides, $\nexists r \in D_{\mathrm{R}} : p_1 \in D_r \subseteq D'$.

Since $D$ is a minimal siphon, $p_1$ is essential for $D$, i.e., $\exists t_2 \in p_1{}^\bullet \cap {}^\bullet D : \{p_1\} = {}^\bullet t_2 \cap D$. Moreover, $\nexists r \in t_2{}^\bullet \cap D_{\mathrm{R}}$; otherwise, $p_1$ is essential for $D_r \subseteq D'$, reaching a contradiction. Then $\exists p_2 \in P_0 \cup P_{\mathrm{S}} : \{p_2\} = t_2{}^\bullet \cap D$. Since $D$ is minimal, $p_2$ is essential for $D$.

Now we can reapply the same reasoning for $p_1$ over $p_2$. Proceeding iteratively, we construct a path $p_1\, t_2\, p_2\, t_3 \ldots p_{n_0}$, where $n_0 \geq 2$ and $\forall i \in [1, n_0) : (\{p_i\} = {}^\bullet t_{i+1} \cap D) \wedge (t_{i+1}{}^\bullet \cap D_{\mathrm{R}} = \emptyset) \wedge (\{p_{i+1}\} = t_{i+1}{}^\bullet \cap D)$. The path is ended up as soon as one of the following two conditions occurs:

**(a)** $\exists t_{\mathrm{end}} \in p_{n_0}{}^\bullet \cap {}^\bullet D_{\mathrm{R}} : \{p_{n_0}\} = {}^\bullet t_{\mathrm{end}} \cap D$.

**(b)** $\exists k \in [1, n_0) : p_k = p_{n_0}$, i.e., we have constructed a circuit.

Such a path is finite since the process subnets are strongly connected state machines. In the case (a) we can backtrack and conclude that every place in the path is essential for $D_r$, for every $r \in t_{\text{end}}{}^{\bullet} \cap D_{\text{R}}$. Therefore, $p_1$ is essential for $D_r \subseteq D'$, reaching a contradiction.

In the case (b) there must exist a place $p_i$ in the circuit, where $k \leq i < n_0$, such that an output transition not belonging to the circuit "makes $p_i$ essential for $D$", i.e.:

$$\exists p_i \exists t'_{i+1} \neq t_{i+1} : (\{p_i\} = {}^{\bullet}t'_{i+1} \cap D) \wedge (t'_{i+1}{}^{\bullet} \cap D \neq \emptyset). \tag{3.4}$$

Otherwise, the set of places $D \setminus P_{\text{C}}$, where $P_{\text{C}}$ is the set of places in the strongly connected state machine induced by the circuit, would for a siphon. Therefore, $D$ could not be a minimal siphon, reaching a contradiction. This is proved next.

First, we prove that ${}^{\bullet}(D \setminus P_{\text{C}}) \cap P_{\text{C}}{}^{\bullet} \subseteq (D \setminus P_{\text{C}})^{\bullet}$. This is instrumental and it will be accomplished in two steps:

1. ${}^{\bullet}(D \setminus P_{\text{C}}) \cap ({}^{\bullet}P_{\text{C}} \cap P_{\text{C}}{}^{\bullet}) = \emptyset$, since, by construction, every transition in ${}^{\bullet}P_{\text{C}} \cap P_{\text{C}}{}^{\bullet}$ (i.e., every transition in the aforementioned strongly connected state machine) does not output to $D_{\text{R}}$, and consequently, every output place belonging to $D$ also belongs to $P_{\text{C}}$.

2. ${}^{\bullet}(D \setminus P_{\text{C}}) \cap (P_{\text{C}}{}^{\bullet} \setminus {}^{\bullet}P_{\text{C}}) \subseteq (D \setminus P_{\text{C}})^{\bullet}$, since by negating equation (3.4) we infer the following fact: if some transition in $P_{\text{C}}{}^{\bullet} \setminus {}^{\bullet}P_{\text{C}}$ has an output place in $D$ then it also has some input place in $D_{\text{R}}$, where $D_{\text{R}} \subseteq D \setminus P_{\text{C}}$.

Now we prove that ${}^{\bullet}(D \setminus P_{\text{C}}) \setminus P_{\text{C}}{}^{\bullet} \subseteq (D \setminus P_{\text{C}})^{\bullet}$. Since ${}^{\bullet}(D \setminus P_{\text{C}}) \subseteq {}^{\bullet}D \subseteq D^{\bullet}$, ${}^{\bullet}(D \setminus P_{\text{C}}) \setminus P_{\text{C}}{}^{\bullet} \subseteq D^{\bullet} \setminus P_{\text{C}}{}^{\bullet}$. On the other hand, $D^{\bullet} \setminus P_{\text{C}}{}^{\bullet} \subseteq (D \setminus P_{\text{C}})^{\bullet}$. Thus, ${}^{\bullet}(D \setminus P_{\text{C}}) \setminus P_{\text{C}}{}^{\bullet} \subseteq (D \setminus P_{\text{C}})^{\bullet}$.

Finally, ${}^{\bullet}(D \setminus P_{\text{C}}) \cap P_{\text{C}}{}^{\bullet} \subseteq (D \setminus P_{\text{C}})^{\bullet}$ and ${}^{\bullet}(D \setminus P_{\text{C}}) \setminus P_{\text{C}}{}^{\bullet} \subseteq (D \setminus P_{\text{C}})^{\bullet}$ together imply that ${}^{\bullet}(D \setminus P_{\text{C}}) \subseteq (D \setminus P_{\text{C}})^{\bullet}$, contradicting the minimality of $D$.

Next, following an analogy with the case of $p_1$ and $t_2$, we can construct a new path $p_i\, t'_{i+1}\, p'_{i+1} \ldots p'_{n_1}$, where $n_1 > i + 1$, in an iterative fashion. Obviously, this new path shares at least one node with the previous path, but at least one transition ($t'_{i+1}$) was not visited before. The path is ended up as soon as a transition $t'_{\text{end}}$ is found (a contradiction is reached backtracking to $p_1$) or a new circuit is closed. In the second case, the new circuit is aggregated to the strongly connected state machine and thus $P'_{\text{C}}$ represents the set of places of it after aggregating the new circuit.

Now there must exist a place $p_j \in P'_{\text{C}}$ such that an output transition not belonging to the strongly connected state machine "makes $p_j$ essential for $D$"; otherwise, the set of places $D \setminus P'_{\text{C}}$ would form a siphon smaller than $D$, for analogous reasons to $D \setminus P_{\text{C}}$.

This reasoning pattern can be reapplied iteratively until either a contradiction against $p_1$ not belonging to $D'$ is reached or the whole process subnet is reconstructed by aggregating new circuits. In the latter case, a contradiction against the minimality of $D$ is reached, since the whole set of places of the process subnet can be detracted from $D$. This iterative algorithm is granted to eventually finish since each new circuit visits at least one new transition (the first one) and the superposition of a set of circuits containing every transition of a strongly connected state machine also contains the whole set of places of the same strongly connected state machine (here: the process subnet). □

**Relations between siphons and p-semiflows**

So far throughout this section, the discussion portrays certain parallels between the way siphons are constructed and the way the supports of the minimal p-semiflows of a $\mathrm{PC^2R}$ net are related with each other. Indeed, Lemma 3.22 shows how every minimal siphon is contained in the union of the 'seed' siphons in $\mathcal{D}^1$ corresponding to its resource places. Similarly, Lemma 3.17 proves that the support of every minimal p-semiflow is contained in the union of the supports of the minimal p-semiflows $\mathbf{y_r}$ of the resource places contained in it. Note that the earlier applies to the minimal siphons (p-semiflows) that contain at least one resource place.

Below we inspect in depth those apparent parallels. In the most general case for $\mathrm{PC^2R}$ nets, there exist particularities in the relation between siphons and p-semiflows that contravene what had previously been observed for $\mathrm{S^4PR}$ nets and transgress what would be expected from the standpoint of intuition.

The first result shows that the class $\mathcal{D}^1$ of the minimal siphons containing only one resource can be directly obtained from the set of minimal p-semiflows associated to the resources.

**Lemma 3.23.** *Let $\mathcal{N}$ be a $PC^2R$ net. For each $r \in P_{\mathrm{R}}$ there exists a minimal siphon, $D_r \in \mathcal{D}^1$ such that $D_r \subseteq \|\mathbf{y_r}\|$.*

*Proof.* By Lemma 2.23, the support of the minimal p-semiflow associated to a resource place $r \in P_{\mathrm{R}}$, $\mathbf{y_r} \in \mathbb{N}^{|P|}$, is a siphon of the net. Nevertheless, in general, $\|\mathbf{y_r}\|$ is not a minimal siphon, but it contains a minimal siphon $D_r$. We prove that $r \in D_r$ by contradiction. Let us suppose that $r \notin D_r$, since $r$ is the only resource in $\|\mathbf{y_r}\|$ and $\forall i \in I_{\mathcal{N}} : \exists p \in \{p_{0_i}\} \cup P_i : p \notin \|\mathbf{y_r}\|$. By Lemma 3.18, the subnet generated by $D_r$ is a set of connected state machines contained in the iterative state machines (process subnets) of $\mathcal{N}$. For each connected state machine, there exists at least a transition without input places and then $D_r$ cannot be a siphon, contradicting the hypothesis. □

From the above it is obvious that (as happened with $\mathrm{S^4PR}$ nets [CRC12]) every

minimal siphon can be directly obtained from the (union of the) set of minimal p-semiflows $\mathbf{y_r}$ associated to each resource place $r$ of the siphon:

**Lemma 3.24.** *Let $\mathcal{N}$ be a $PC^2R$ net, and $D \subseteq P$ a minimal siphon of $\mathcal{N}$ such that $D_{\mathrm{R}} = D \cap P_{\mathrm{R}} \neq \emptyset$. $D \subseteq \bigcup_{r \in D_{\mathrm{R}}} \|\mathbf{y_r}\|$.*

*Proof.* Straightforward from Lemmas 3.22 and 3.23. □

From the above, and considering Lemma 3.17, one might feel invited to think that every minimal siphon should be supported by every p-semiflow whose support exactly shares the same set of resource places. Next we prove that this is indeed true, yet the proof is far from obvious, in general, for $\mathrm{PC^2R}$ nets.

**Lemma 3.25.** *Let $\mathcal{N}$ be a $PC^2R$ net, $\mathbf{y} \in \mathbb{N}^{|P|}$ be a p-semiflow of $\mathcal{N}$, and $D$ be a minimal siphon of $\mathcal{N}$ such that $D_{\mathrm{R}} = D \cap P_{\mathrm{R}} \neq \emptyset$. $D_{\mathrm{R}} = \|\mathbf{y}\| \cap P_{\mathrm{R}} \implies D \subseteq \|\mathbf{y}\|$.*

*Proof.* By contradiction. Let us suppose that $\exists p_1 \in D \setminus \|\mathbf{y}\|$. Obviously, $p_1 \in P_0 \cup P_{\mathrm{S}}$, since $D_{\mathrm{R}} = \|\mathbf{y}\| \cap P_{\mathrm{R}}$. Let $i \in I_{\mathcal{N}}$ be the index of the process subnet to which $p_1$ belongs, i.e., $p_1 \in P_i$.

By Lemma 3.24 it follows: $p_1 \in D \setminus \|\mathbf{y}\| \subseteq \bigcup_{r \in D_{\mathrm{R}}} \|\mathbf{y_r}\| \setminus \|\mathbf{y}\|$. This implies that $\bigcup_{r \in D_{\mathrm{R}}} \|\mathbf{y_r}\|$ contains the whole $i$-th process subnet, i.e., $P_i \subseteq \bigcup_{r \in D_{\mathrm{R}}} \|\mathbf{y_r}\|$. This is necessary so as to annul place $p_1$ in $\|\mathbf{y}\|$ by establishing $K_i > 0$ in the linear decomposition of $\mathbf{y}$ expressed by Lemma 3.16 (otherwise, $p_1 \in \|\mathbf{y}\|$, which is not possible). Furthermore, since $\mathbf{y_{S_i}}[P_i] = \mathbf{1}$, the component $p_1$ in $\sum_{r \in P_{\mathrm{R}}} \mathbf{y}[r] \cdot \mathbf{y_r}$ must return a minimum among the set of places of $P_i$ in order to be annulled, i.e.:

$$\sum_{r \in P_{\mathrm{R}}} \mathbf{y}[r] \cdot \mathbf{y_r}[p_1] \leq \min_{p \in P_i} \sum_{r \in P_{\mathrm{R}}} \mathbf{y}[r] \cdot \mathbf{y_r}[p]$$

Since $D$ is a minimal siphon, $p_1$ is essential for $D$, i.e., $\exists t_2 \in p_1{}^\bullet \cap {}^\bullet D : \{p_1\} = {}^\bullet t_2 \cap D$. Let $\{p_2\} = t_2{}^\bullet \cap P_i$. If $t_2{}^\bullet \cap D_{\mathrm{R}} \neq \emptyset$ then $\forall r \in D_{\mathrm{R}} : \mathbf{y_r}[p_1] \geq \mathbf{y_r}[p_2]$, and $\exists r' \in D_{\mathrm{R}} : \mathbf{y_{r'}}[p_1] > \mathbf{y_{r'}}[p_2]$. Therefore, $\sum_{r \in P_{\mathrm{R}}} \mathbf{y}[r] \cdot \mathbf{y_r}[p_1] > \sum_{r \in P_{\mathrm{R}}} \mathbf{y}[r] \cdot \mathbf{y_r}[p_2]$, and $p_1$ cannot be annulled in $\mathbf{y}$, reaching a contradiction. Therefore, $t_2{}^\bullet \cap D_{\mathrm{R}} = \emptyset$, i.e., $\{p_2\} = t_2{}^\bullet \cap D$. Since $D$ is minimal, $p_2$ is also essential for $D$.

Now we can reapply the same reasoning for $p_1$ over $p_2$. Proceeding iteratively, we construct a path $p_1\, t_2\, p_2\, t_3 \ldots p_{n_0}$, where $n_0 \geq 2$ and $\forall i \in [1, n_0) : (\{p_i\} = {}^\bullet t_{i+1} \cap D) \wedge (t_{i+1}{}^\bullet \cap D_{\mathrm{R}} = \emptyset) \wedge (\{p_{i+1}\} = t_{i+1}{}^\bullet \cap D)$. The path is ended up as soon as one of the following two conditions occurs:

**(a)** $\exists t_{\mathrm{end}} \in p_{n_0}{}^\bullet \cap {}^\bullet D_{\mathrm{R}} : \{p_{n_0}\} = {}^\bullet t_{\mathrm{end}} \cap D$.

**(b)** $\exists k \in [1, n_0) : p_k = p_{n_0}$, i.e., we have constructed a circuit.

Such a path is finite since the process subnets are strongly connected state machines. In the case (a) we can conclude that $\sum_{r\in P_R} \mathbf{y}[r]\cdot \mathbf{y_r}[p_1] > \sum_{r\in P_R} \mathbf{y}[r]\cdot \mathbf{y_r}[p_{n_0}]$, and $p_1$ cannot be annulled in $\mathbf{y}$, reaching a contradiction.

In the case (b) there must exist a place $p_i$ in the circuit, where $k \leq i < n_0$, such that an output transition not belonging to the circuit "makes $p_i$ essential for $D$", i.e.:

$$\exists p_i \exists t'_{i+1} \neq t_{i+1} : (\{p_i\} = {}^\bullet t'_{i+1} \cap D) \wedge ({t'_{i+1}}^\bullet \cap D \neq \emptyset). \tag{3.5}$$

Otherwise, the set of places $D \setminus P_C$, where $P_C$ is the set of places in the strongly connected state machine induced by the circuit, would for a siphon. Therefore, $D$ could not be a minimal siphon, reaching a contradiction. This is proved next.

First, we prove that ${}^\bullet(D \setminus P_C) \cap {P_C}^\bullet \subseteq (D \setminus P_C)^\bullet$. This is instrumental and it will be accomplished in two steps:

1. ${}^\bullet(D \setminus P_C) \cap ({}^\bullet P_C \cap {P_C}^\bullet) = \emptyset$, since, by construction, every transition in ${}^\bullet P_C \cap {P_C}^\bullet$ (i.e., every transition in the aforementioned strongly connected state machine) does not output to $D_R$, and consequently, every output place belonging to $D$ also belongs to $P_C$.

2. ${}^\bullet(D \setminus P_C) \cap ({P_C}^\bullet \setminus {}^\bullet P_C) \subseteq (D \setminus P_C)^\bullet$, since by negating equation (3.5) we infer the following fact: if some transition in ${P_C}^\bullet \setminus {}^\bullet P_C$ has an output place in $D$ then it also has some input place in $D_R$, where $D_R \subseteq D \setminus P_C$.

Now we prove that ${}^\bullet(D \setminus P_C) \setminus {P_C}^\bullet \subseteq (D \setminus P_C)^\bullet$. Since ${}^\bullet(D \setminus P_C) \subseteq {}^\bullet D \subseteq D^\bullet$, ${}^\bullet(D \setminus P_C) \setminus {P_C}^\bullet \subseteq D^\bullet \setminus {P_C}^\bullet$. On the other hand, $D^\bullet \setminus {P_C}^\bullet \subseteq (D \setminus P_C)^\bullet$. Thus, ${}^\bullet(D \setminus P_C) \setminus {P_C}^\bullet \subseteq (D \setminus P_C)^\bullet$.

Finally, ${}^\bullet(D \setminus P_C) \cap {P_C}^\bullet \subseteq (D \setminus P_C)^\bullet$ and ${}^\bullet(D \setminus P_C) \setminus {P_C}^\bullet \subseteq (D \setminus P_C)^\bullet$ together imply that ${}^\bullet(D \setminus P_C) \subseteq (D \setminus P_C)^\bullet$, contradicting the minimality of $D$.

Next, following an analogy with the case of $p_1$ and $t_2$, we can construct a new path $p_i\, t'_{i+1}\, p'_{i+1} \ldots p'_{n_1}$, where $n_1 > i+1$, in an iterative fashion. Obviously, this new path shares at least one node with the previous path, but at least one transition ($t'_{i+1}$) was not visited before. The path is ended up as soon as a transition $t'_{\text{end}}$ is found (a contradiction is reached against $p_1$ being annulled in $\mathbf{y}$) or a new circuit is closed. In the second case, the new circuit is aggregated to the strongly connected state machine and thus $P'_C$ represents the set of places of it after aggregating the new circuit.

Now there must exist a place $p_j \in P'_C$ such that an output transition not belonging to the strongly connected state machine "makes $p_j$ essential for $D$"; otherwise, the set of places $D \setminus P'_C$ would form a siphon smaller than $D$, for analogous reasons to $D \setminus P_C$.

This reasoning pattern can be reapplied iteratively until either a contradiction against $p_1$ not belonging to $D'$ is reached or the whole process subnet is reconstructed

by aggregating new circuits. In the latter case, a contradiction against the minimality of $D$ is reached, since the whole set of places of the process subnet can be detracted from $D$. This iterative algorithm is granted to eventually finish since each new circuit visits at least one new transition (the first one) and the superposition of a set of circuits containing every transition of a strongly connected state machine also contains the whole set of places of the same strongly connected state machine (here: the process subnet). □

However, the reverse of Lemma 3.25 is in general false even considering only the set of minimal p-semiflows:

**Property 3.26.** *There exists a $PC^2R$ net $\mathcal{N}$, a minimal p-semiflow $\mathbf{y}$ and a minimal siphon $D$ of $\mathcal{N}$ such that $D \subseteq \|\mathbf{y}\|$ and $D \cap P_{\mathrm{R}} \subsetneq \|\mathbf{y}\| \cap P_{\mathrm{R}}$.*

The net in Fig. 3.24 (which is the same net that appears in Fig. 2.8) has a minimal siphon $D = \{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{R4}, \mathrm{B5}, \mathrm{C1}\}$ which is covered by the minimal p-semiflow $\mathbf{y} \equiv \mathbf{m}[\mathrm{R1}]+\mathbf{m}[\mathrm{R2}]+\mathbf{m}[\mathrm{R3}]+\mathbf{m}[\mathrm{R4}]+\mathbf{m}[\mathrm{R5}]+\mathbf{m}[\mathrm{A4}]+\mathbf{m}[\mathrm{A6}]+\mathbf{m}[\mathrm{B3}]+\mathbf{m}[\mathrm{B5}]+3\cdot\mathbf{m}[\mathrm{C1}]$, where $\{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{R4}\} = D \cap P_{\mathrm{R}} \subsetneq \|\mathbf{y}\| \cap P_{\mathrm{R}} = \{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{R4}, \mathrm{R5}\}$.

Indeed, there may exist a siphon $D'$ which belongs to a higher level in the taxonomy of siphons and is also supported by the same minimal p-semiflow $\mathbf{y}$:

**Property 3.27.** *There exists a $PC^2R$ net $\mathcal{N}$, a minimal p-semiflow $\mathbf{y}$ and two minimal siphons $D \in \mathcal{D}^i$ and $D' \in \mathcal{D}^j$ of $\mathcal{N}$ such that $i > j > 0$, $D \subseteq \|\mathbf{y}\|$ and $D' \subseteq \|\mathbf{y}\|$.*

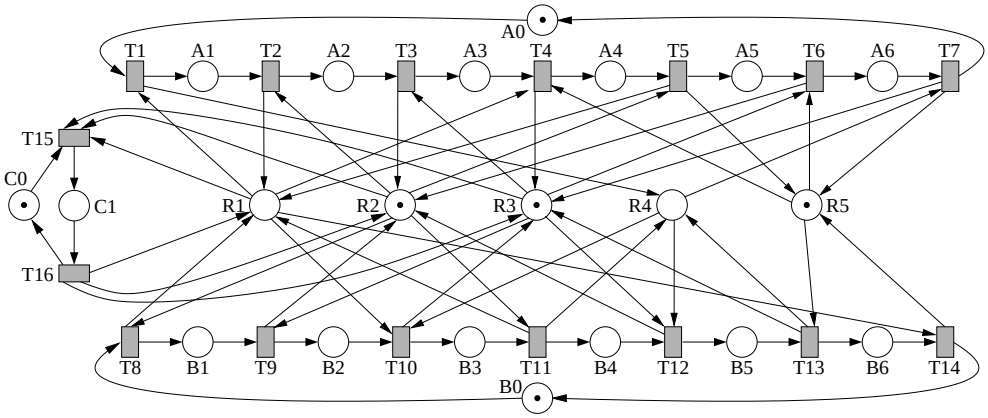
Returning to the example of the net in Fig. 3.24, the p-semiflow $\mathbf{y}$ does not only supports the places of $D \in \mathcal{D}^4$ but also those of $D' \in \mathcal{D}^5$, where $D' = \{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{R4}, \mathrm{R5}, \mathrm{C1}\}$.

Obviously the above property does not hold for simpler subclasses such as $\mathrm{S}^4\mathrm{PR}$ [CRC12]. The same applies for the next property, which states that there can also exist several minimal p-semiflows with different sets of resource places while supporting the same minimal siphon:

**Property 3.28.** *There exists a $PC^2R$ net $\mathcal{N}$, a minimal siphon $D$ and two minimal p-semiflows $\mathbf{y}, \mathbf{y}'$ of $\mathcal{N}$ such that $D \subseteq \|\mathbf{y}\|$, $D \subseteq \|\mathbf{y}'\|$ and $\emptyset \neq \|\mathbf{y}\| \cap P_R \subsetneq \|\mathbf{y}'\| \cap P_R$.*

In the example of the net in Fig. 3.24, $\mathbf{y}$ is not the minimal p-semiflow supporting $D$ and the lowest number of resource places. That minimal p-semiflow is: $\mathbf{y}' \equiv \mathbf{m}[\mathrm{R1}]+\mathbf{m}[\mathrm{R2}]+\mathbf{m}[\mathrm{R3}]+\mathbf{m}[\mathrm{R4}]+\mathbf{m}[\mathrm{B0}]+\mathbf{m}[\mathrm{B1}]+\mathbf{m}[\mathrm{B2}]+2\cdot\mathbf{m}[\mathrm{B3}]+\mathbf{m}[\mathrm{B4}]+2\cdot\mathbf{m}[\mathrm{B5}]+3\cdot\mathbf{m}[\mathrm{C1}]$. In this case, $D \cap P_{\mathrm{R}} = \|\mathbf{y}'\| \cap P_{\mathrm{R}}$ with $D \subseteq \|\mathbf{y}'\|$.

Another novelty regarding $\mathrm{PC}^2\mathrm{R}$ nets is that, given a non-empty subset of resource places $\mathcal{R} \subseteq P_{\mathrm{R}}$ there may exist two minimal p-semiflows $\mathbf{y}$, $\mathbf{y}'$ of $\mathcal{N}$ such that $\|\mathbf{y}\| \cap P_{\mathrm{R}} = \|\mathbf{y}'\| \cap P_{\mathrm{R}} = \mathcal{R}$. A minimal siphon can be supported by several minimal p-semiflows which share the same set of resource places:




*Minimal siphons in $\mathcal{D}^n$, with $n > 1$:*

$$\mathcal{D}^2 \begin{cases} \{\text{R2}, \text{R4}, \text{A0}, \text{A2}, \text{A5}, \text{B1}, \text{B5}, \text{C1}\} \\ \{\text{R2}, \text{R5}, \text{A2}, \text{A6}, \text{B1}, \text{B4}, \text{B6}, \text{C1}\} \\ \{\text{R4}, \text{R5}, \text{A0}, \text{A4}, \text{B3}, \text{B6}\} \end{cases}$$

$$\mathcal{D}^3 \begin{cases} \{\text{R1}, \text{R2}, \text{R3}, \text{A6}, \text{B5}, \text{C1}\} \\ \{\text{R1}, \text{R2}, \text{R4}, \text{A2}, \text{A5}, \text{B1}, \text{B5}, \text{C1}\} \\ \{\text{R1}, \text{R2}, \text{R5}, \text{A2}, \text{A6}, \text{B1}, \text{B4}, \text{C1}\} \\ \{\text{R2}, \text{R3}, \text{R4}, \text{A0}, \text{A3}, \text{B5}, \text{C1}\} \\ \{\text{R2}, \text{R3}, \text{R5}, \text{A6}, \text{B2}, \text{B6}, \text{C1}\} \\ \{\text{R2}, \text{R4}, \text{R5}, \text{A0}, \text{A2}, \text{B1}, \text{B6}, \text{C1}\} \end{cases}$$

$$\mathcal{D}^4 \begin{cases} D_{\text{R1R2R3R4}} = \{\text{R1} - \text{R4}, \text{B5}, \text{C1}\} \\ D_{\text{R1R2R3R5}} = \{\text{R1} - \text{R3}, \text{R5}, \text{A6}, \text{C1}\} \\ \{\text{R1}, \text{R2}, \text{R4}, \text{R5}, \text{A2}, \text{B1}, \text{C1}\} \\ \{\text{R2}, \text{R3}, \text{R4}, \text{R5}, \text{A0}, \text{B6}, \text{C1}\} \end{cases}$$

$$\mathcal{D}^5 : \; D_{\text{R1-R5}} = \{\text{R1} - \text{R5}, \text{C1}\}$$

*Minimal p-semiflows supporting more than one resource place:*

$$\begin{aligned} \mathbf{y^1_{R1R2R3R4}} : \mathbf{m}[\text{R1}] + \mathbf{m}[\text{R2}] + \mathbf{m}[\text{R3}] + \mathbf{m}[\text{R4}] + \mathbf{m}[\text{B0}] + \mathbf{m}[\text{B1}] + \\ \mathbf{m}[\text{B2}] + 2 \cdot \mathbf{m}[\text{B3}] + \mathbf{m}[\text{B4}] + 2 \cdot \mathbf{m}[\text{B5}] + 3 \cdot \mathbf{m}[\text{C1}] = 3 \\ \mathbf{y^1_{R1R2R3R5}} : \mathbf{m}[\text{R1}] + \mathbf{m}[\text{R2}] + \mathbf{m}[\text{R3}] + \mathbf{m}[\text{R5}] + \mathbf{m}[\text{A1}] + \mathbf{m}[\text{A2}] + \\ \mathbf{m}[\text{A3}] + 2 \cdot \mathbf{m}[\text{A4}] + \mathbf{m}[\text{A5}] + 2 \cdot \mathbf{m}[\text{A6}] + 3 \cdot \mathbf{m}[\text{C1}] = 3 \\ \mathbf{y^1_{R1-R5}} : \mathbf{m}[\text{R1}] + \mathbf{m}[\text{R2}] + \mathbf{m}[\text{R3}] + \mathbf{m}[\text{R4}] + \mathbf{m}[\text{R5}] + \mathbf{m}[\text{A4}] + \\ \mathbf{m}[\text{A6}] + \mathbf{m}[\text{B3}] + \mathbf{m}[\text{B5}] + 3 \cdot \mathbf{m}[\text{C1}] = 3 \end{aligned}$$

*Minimal siphons $D$ covered by a minimal p-semiflow $y$ such that $D \cap P_R = \|\mathbf{y}\| \cap P_R$:*

$$D_{\text{R1R2R3R4}} \subsetneq \|\mathbf{y^1_{R1R2R3R4}}\|; \; D_{\text{R1R2R3R5}} \subsetneq \|\mathbf{y^1_{R1R2R3R5}}\|; \; D_{\text{R1-R5}} \subsetneq \|\mathbf{y^1_{R1-R5}}\|$$

**Figure 3.24**: A PC$^2$R net with minimal siphons that are not covered by the support of any minimal p-semiflow. Note that no p-semiflow holds that its support is a minimal siphon

**Property 3.29.** *There exists a PC²R net $\mathcal{N}$, a minimal siphon $D$ and two minimal p-semiflows $\mathbf{y}, \mathbf{y}'$ of $\mathcal{N}$ such that $D \subseteq \|\mathbf{y}\|$, $D \subseteq \|\mathbf{y}'\|$ and $\|\mathbf{y}\| \cap P_{\mathrm{R}} = \|\mathbf{y}'\| \cap P_{\mathrm{R}} = D \cap P_{\mathrm{R}}$.*

The net in Fig. 3.25 (which is the same net of Fig. 2.7, yet recovered here) proves the previous property. The siphon $D_{\mathrm{R1R2R3R4}}$ is supported by the minimal p-semiflows $\mathbf{y^1_{R1R2R3R4}}$, $\mathbf{y^2_{R1R2R3R4}}$ and $\mathbf{y^3_{R1R2R3R4}}$, which share the same set of resource places. Indeed, the whole set of minimal siphons in $\mathcal{D}^n$, with $n > 1$, is tightly bonded with the set of minimal p-semiflows supporting more than one resource place. All these minimal siphons are covered by the support of a minimal p-semiflow that contains the same set of resource places. And most of these siphons (all of them except $D_{\mathrm{R1R2R3R4}}$) are equal to the support of a minimal p-semiflow.

Note that the remaining minimal siphon ($D_{\mathrm{R1R2R3R4}}$) is equal to the intersection of the supports of the minimal p-semiflows mentioned above. In general, every minimal siphon is contained in the intersection of the supports of the minimal p-semiflows whose supports exactly have the set of resources of the siphon (if they exist). In fact, this can be seen as a corollary of Lemma 3.25:

**Lemma 3.30.** *Let $\mathcal{N}$ be a PC²R net, and $\mathcal{Y}$ be a non-empty set of minimal p-semiflows of $\mathcal{N}$ such that $\forall \mathbf{y}, \mathbf{y}' \in \mathcal{Y} : \|\mathbf{y}\| \cap P_{\mathrm{R}} = \|\mathbf{y}'\| \cap P_{\mathrm{R}} = P_{\mathrm{R}}^{\mathcal{Y}}$. if there exists one minimal siphon $D$ of $\mathcal{N}$ such that $D_{\mathrm{R}} = D \cap P_{\mathrm{R}} = P_{\mathrm{R}}^{\mathcal{Y}}$, then $D \subseteq \bigcap_{\mathbf{y} \in \mathcal{Y}} \|\mathbf{y}\|$.*

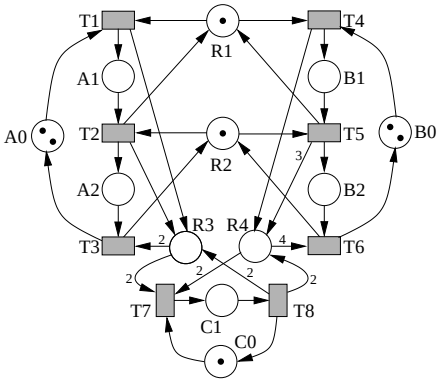
*Proof.* By Lemma 3.25, for every $\mathbf{y} \in \mathcal{Y} : D \subseteq \|\mathbf{y}\|$. Therefore, $D \subseteq \bigcap_{\mathbf{y} \in \mathcal{Y}} \|\mathbf{y}\|$. □

On the other hand, the relation between minimal siphons and minimal p-semiflows is rather more diffuse for the net in Fig. 3.24. In this case, most of the minimal siphons are not supported by any minimal p-semiflow, and none of them is equal to the support of a p-semiflow.

The last property inspects the form of the siphons contained in a minimal p-semiflow $\mathbf{y}$ of $\mathcal{N}$ whose support contains more than one resource place. By Lemma 3.19, there exists *at most* one minimal siphon $D$ covering strictly the set of resources $\|\mathbf{y}\| \cap P_{\mathrm{R}}$, i.e., $D_{\mathrm{R}} = \|\mathbf{y}\| \cap P_{\mathrm{R}}$. Besides, Lemma 3.25 states that if $D$ exists then $D \subseteq \|\mathbf{y}\|$. The result prove that there can exist a minimal siphon supported by $\mathbf{y}$ even when such a minimal siphon $D$, with $D_{\mathrm{R}} = \|\mathbf{y}\| \cap P_{\mathrm{R}}$, does not exist:

**Property 3.31.** *There exists a PC²R net $\mathcal{N}$, and a minimal p-semiflow of $\mathcal{N}$, $\mathbf{y} \in \mathbb{N}^{|P|}$, such that its support contains more resource places than every minimal siphon it covers, i.e., there exists at least one minimal siphon $D$ of $\mathcal{N}$ such that $D \subseteq \|\mathbf{y}\|$, and for every such minimal siphon $D$ the following holds: $D \cap P_{\mathrm{R}} \subsetneq \|\mathbf{y}\| \cap P_{\mathrm{R}}$.*

The PC²R net in Fig. 3.26 has three minimal siphons ($D_{\mathrm{R1R2}}$, $D_{\mathrm{R1R3}}$, and $D_{\mathrm{R2R3}}$) which are supported by the minimal p-semiflow ($\mathbf{y^1_{R1R2R3}}$), but none of these contains the three resource places R1, R2 and R3. It is noticeable that the net is live for every 0-acceptable initial marking.




*Minimal siphons in $\mathcal{D}^n$, with $n > 1$:*

$$\mathcal{D}^2 \begin{cases} D_{\mathrm{R2R3}} = \{\mathrm{R2}, \mathrm{R3}, \mathrm{A0}, \mathrm{B2}, \mathrm{C1}\} \\ D_{\mathrm{R2R4}} = \{\mathrm{R2}, \mathrm{R4}, \mathrm{A2}, \mathrm{B0}, \mathrm{C1}\} \end{cases}$$

$$\mathcal{D}^3 \begin{cases} D_{\mathrm{R1R2R3}} = \{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{B1}, \mathrm{B2}, \mathrm{C1}\} \\ D_{\mathrm{R1R2R4}} = \{\mathrm{R1}, \mathrm{R2}, \mathrm{R4}, \mathrm{A1}, \mathrm{A2}, \mathrm{C1}\} \\ D_{\mathrm{R2R3R4}} = \{\mathrm{R2}, \mathrm{R3}, \mathrm{R4}, \mathrm{A0}, \mathrm{B0}, \mathrm{C1}\} \end{cases}$$

$$\mathcal{D}^4 \left\{ D_{\mathrm{R1R2R3R4}} = \{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{R4}, \mathrm{C1}\} \right.$$

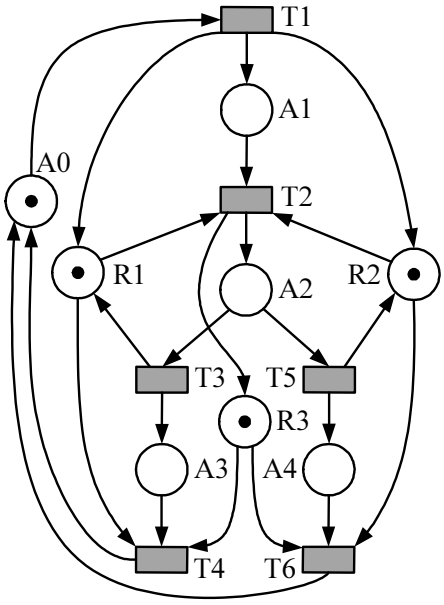
*Minimal p-semiflows supporting more than one resource place:*

$$\begin{aligned}
\mathbf{y^1_{R2R3}} &: \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{B2}] + 2 \cdot \mathbf{m}[\mathrm{C1}] &&= 3 \\
\mathbf{y^1_{R2R4}} &: 3 \cdot \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R4}] + 3 \cdot \mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{B0}] + 2 \cdot \mathbf{m}[\mathrm{C1}] &&= 5 \\
\mathbf{y^1_{R1R2R3}} &: \mathbf{m}[\mathrm{R1}] + 2 \cdot \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{B1}] + 2 \cdot \mathbf{m}[\mathrm{B2}] + 2 \cdot \mathbf{m}[\mathrm{C1}] &&= 3 \\
\mathbf{y^1_{R1R2R4}} &: \mathbf{m}[\mathrm{R1}] + 4 \cdot \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R4}] + \mathbf{m}[\mathrm{A1}] + 4 \cdot \mathbf{m}[\mathrm{A2}] + 2 \cdot \mathbf{m}[\mathrm{C1}] &&= 5 \\
\mathbf{y^1_{R2R3R4}} &: 3 \cdot \mathbf{m}[\mathrm{R2}] + 3 \cdot \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + 3 \cdot \mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{B0}] + 8 \cdot \mathbf{m}[\mathrm{C1}] &&= 11 \\
\mathbf{y^1_{R1R2R3R4}} &: 2 \cdot \mathbf{m}[\mathrm{R1}] + 4 \cdot \mathbf{m}[\mathrm{R2}] + 2 \cdot \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + \mathbf{m}[\mathrm{B1}] + 6 \cdot \mathbf{m}[\mathrm{C1}] &&= 6 \\
\mathbf{y^2_{R1R2R3R4}} &: \mathbf{m}[\mathrm{R1}] + 4 \cdot \mathbf{m}[\mathrm{R2}] + 3 \cdot \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + 2 \cdot \mathbf{m}[\mathrm{A0}] + 8 \cdot \mathbf{m}[\mathrm{C1}] &&= 9 \\
\mathbf{y^3_{R1R2R3R4}} &: \mathbf{m}[\mathrm{R1}] + 4 \cdot \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{R4}] + 2 \cdot \mathbf{m}[\mathrm{A2}] + 4 \cdot \mathbf{m}[\mathrm{C1}] &&= 5
\end{aligned}$$

*Minimal siphons $D$ covered by a minimal p-semiflow $\mathbf{y}$ such that $D \cap P_R = \|\mathbf{y}\| \cap P_R$:*

$$\begin{aligned}
&[\mathcal{D}^1] \quad D_{\mathrm{R1}} = \|\mathbf{y_{R1}}\|;\ D_{\mathrm{R2}} = \|\mathbf{y_{R2}}\|;\ D_{\mathrm{R3}} = \|\mathbf{y_{R3}}\|;\ D_{\mathrm{R4}} = \|\mathbf{y_{R4}}\| \\
&[\mathcal{D}^2] \quad D_{\mathrm{R2R3}} = \|\mathbf{y^1_{R2R3}}\|;\ D_{\mathrm{R2R4}} = \|\mathbf{y^1_{R2R4}}\| \\
&[\mathcal{D}^3] \quad D_{\mathrm{R1R2R3}} = \|\mathbf{y^1_{R1R2R3}}\|;\ D_{\mathrm{R1R2R4}} = \|\mathbf{y^1_{R1R2R4}}\|;\ D_{\mathrm{R2R3R4}} = \|\mathbf{y^1_{R2R3R4}}\| \\
&[\mathcal{D}^4] \quad D_{\mathrm{R1R2R3R4}} = \|\mathbf{y^1_{R1R2R3R4}}\| \cap \|\mathbf{y^2_{R1R2R3R4}}\| \cap \|\mathbf{y^3_{R1R2R3R4}}\|
\end{aligned}$$

**Figure 3.25**: A PC$^2$R net such that every minimal siphon is covered by the support of a minimal p-semiflow




*Minimal siphons:*

$$D_{\mathrm{R1}} = \|\mathbf{y_{R1}}\|;\ D_{\mathrm{R2}} = \|\mathbf{y_{R2}}\|;\ D_{\mathrm{R3}} = \|\mathbf{y_{R3}}\|$$
$$D_{\mathrm{R1R2}} = \{\mathrm{R1}, \mathrm{R2}, \mathrm{A0}, \mathrm{A2}\}$$
$$D_{\mathrm{R1R3}} = \{\mathrm{R1}, \mathrm{R3}, \mathrm{A0}, \mathrm{A2}\}$$
$$D_{\mathrm{R2R3}} = \{\mathrm{R2}, \mathrm{R3}, \mathrm{A0}, \mathrm{A2}\}$$

*Minimal p-semiflows:*

$$\mathbf{y_{R1}} : \mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{A4}]$$
$$\mathbf{y_{R2}} : \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{A3}]$$
$$\mathbf{y_{R3}} : \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{A1}]$$
$$\mathbf{y^1_{R1R2R3}} : \mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + 2 \cdot \mathbf{m}[\mathrm{A0}] + \mathbf{m}[\mathrm{A2}]$$

**Figure 3.26**: A $PC^2R$ net with a minimal p-semiflow whose support contains more resource places than every minimal siphon it covers

In the context of the more simple category of $S^4PR$ nets, no siphon is the support of a p-semiflow except for those contained in $\mathcal{D}^1$ and for those corresponding to the set of places in the process subnets. As a result, all the rest of siphons are *potentially* bad siphons. This does not mean (at all) that all those siphons can eventually be emptied for a given initial marking. However, there must exist thieve places that could steal tokens from these siphons until they become undermarked if the initial marking allows certain firing sequences.

The fact that there may exist other minimal siphons that are the support of p-semiflows is an interesting novelty in the scenario portrayed by the $PC^2R$ class. In the case of Fig. 3.25, the fact that every minimal siphon is covered by p-semiflows in the way expressed by Lemmas 3.25 and 3.30 is a powerful property. Thanks to this, the net is live for every initial marking that has enough tokens in each resource subnet. In other words, the net is live if the initial marking is 1-acceptable. Unfortunately, this is not necessarily true if the initial marking is not 1-acceptable but 0-acceptable, as proved in Chap. 2 (see the discussion surrounding Fig. 2.7). Although this is here discussed in a pretty informal way, this result will be further discussed and formalised in future work.

## 3.4 A toolbox for synthesising live PC$^2$R models

### 3.4.1 No room for despair: Heuristics to obtain live models

Earlier in this chapter, concepts and strategies usually deployed to correct RAS models are reviewed. This is done mainly from the perspective of liveness enforcing in FMSs but also from the domain of the correction of adaptive routing algorithms in interconnection networks. At the same time, boundaries are drawn that traverse the taxonomy of RAS models beyond which those correction strategies are no longer useful or valid. From that perspective, bridges have been built with the most general RAS classes. From what is exposed up to this point, it should be evident that those synthesis methods are deeply rooted in the (half-behavioural, half-structural) liveness characterisations.

As discussed in the previous section, these characterizations cannot be extrapolated to the more general PC$^2$R net class which is suitable for modelling many multithreaded software systems. In light of the above, it is neither possible to directly extend those synthesis techniques for the correction of PC$^2$R models. Properties which are disruptive with previous classes have been explored to reinforce this idea. The observed results suggest that, in general, adressing the RAP beyond the S$^4$PR subclass introduces new, significantly more complex problems.

On the other hand, the above synthesis techniques are based on constructive principles that guarantee the ability to systematically deploy them in those application domains for which they are specifically designed (FMSs, multiprocessor interconnection networks, etc.). However, the nature of these solutions may present problems under certain conditions as to their deployment in the context of multithreaded software systems.

This should not lead to the false conclusion that all hope is vain in the realm of multithreaded software. In this section we show that the software engineer can take advantage of all the baggage learned from a RAS perspective. On the other hand, it is important to remember that, ultimately, there is always a solution that achieves liveness for an RAS. When the model is too complicated or it is not possible to achieve analytical results satisfactorily, it is still possible to act in a similar way to the classic methods based on the Banker's Algorithm [Dij82, ETGVC02]: the execution of the troubling processes (or even all processes) can be sequentialised.

In relation to this ultimate solution, the engineer can hope for an improvement in the concurrency level by taking advantage of the knowledge obtained from the RAS abstraction of the software system. As we shall see in the next subsections, there are different types of approaches to the problem. Their application depends largely on the nature of the software system abstracted, and it corresponds to the engineer, ultimately, to decide on the suitability of a particular tool.

The most obvious kind of approach lies on the application of the classic techniques based on control theory, i.e., on the addition of monitors places that restrict concurrency in the model. One possible implementation of such monitors places requires the modification of the original code by inserting calls to allocate/release primitives over a new set of semaphores which are shared between the conflicting processes. The logic that manipulates the semaphores leverages a reduction in concurrency to ensure that the whole system is maintained in a safe condition with respect to the overall resource allocation state. The applicability of such a technique depends on various factors, some of which are discussed later in this section.

A second kind of approach lies in increasing the number of resources in those situations which are prone to deadlock. This practice can actually free the designer from the need to modify the source code. However, this increase should be done in a controlled way since, depending on the implementation, a structural solution to the problem may not be provided. In the latter case, the problem may be reproduced as soon as the system is resized. Nevertheless, it is often possible to privatise the use of these new resources, in the spirit of the method described in Sect. 3.2.4 for the routing problem in interconnection networks, thus bringing a structural solution without degrading the system concurrency.

On certain occasions, it is even possible to alter the structure of the processes in the model in order to enforce liveness. This type of approach is rarely natural in the field of FMS, where production plans are usually preset. However, it makes more sense in the context of multithreaded programming, where the programming logic can sometimes be altered to achieve, for example, that a process can return to a safe state if the allocation level of some shared resources surpasses a certain threshold.

In short: the state of knowledge in the field of RASs and the nature of software systems often allow us to play with models in a much open way in order to correct the system and ensure liveness with respect to the RAP. The degree of freedom is obviously dependent on the particular characteristics of the system to correct. In order to obtain an optimal solution, the designer should ask himself questions such as: Is the source code available? Is it modifiable? What are the involved resources like? Can they be increased if necessary? Once they are assigned to a process, and before they have served their purpose, can they be released for the sake of reaching a safe state? And so on.

Considering the above, the following section presents a toolset of correction strategies for PC²R nets whose application ensures obtaining live models. The application of these tools can be complementary or not depending on the characteristics of the multithreaded software system which is analysed and/or desired. In short, the goal is not to provide a fixed, systematic methodology, but a sufficient set of tools that are left to the designer so s/he can correct the system the most appropriate way possible.

### 3.4.2 Divide and conquer: Deconstructing a PC²R model

Throughout much of Chap. 2 and Sect. 3.3 it has been shown that the general problem of deciding the liveness of the RAS abstraction of a multithreaded system modelled through the PC²R class is, computationally speaking, a very complex problem. However, it is often possible to address the problem in stages through the observation and manipulation of smaller RAS subsystems which are connected in some particular simple way through a subset of shared resources. These subsystems can be identified as subnets of the PC²R model. These subnets may belong to simpler subclasses such as S⁴PR or SOAR². As seen in Sect. 3.2, there exist well-known techniques to enforce liveness on this kind of models. This lets us tackle the liveness problem following a sort of *divide-and-conquer* heuristic which often alleviates the inherent computational complexity of the problem.

Henceforth we present the basic construction principles on which such a methodology is based. The aim is to provide a framework that will enable the correction of the RAP in a multithreaded system modelled through a PC²R net with a 1-acceptable initial marking. As already discussed in Chap. 2, the requirement of the initial marking being 1-acceptable implies that a single instance of each thread class can be entirely executed in isolation, for every possible execution path, without requiring the intervention of any other process as far as the allocation of resources refers. Note that this is often a desirable property in the field of multithreaded software.

As the class of PC²R nets is introduced in Sect. 2.3.2, it is said that a PC²R net is a modular model that can be obtained from merging a non-empty set of smaller PC²R models by fusion of the resource places they share. It is this vision that we exploit in this section, but taken from the opposite end. Given a PC²R model, we can look at it as a set of PC²R subsystems that are connected through a set of shared resource places between them. Throughout this section, we will refer to the latter type of resource places as *interface* resource places. Similarly, every other resource place is considered as *internal* to a particular subsystem. Figure 3.27 summarises the vision of a system from this perspective.

Conceived in this manner, the set of subsystems in which a PC²R model is decomposed is, in general, not absolute but arbitrary. In other words, for the same PC²R model (assuming that it has more than one process subnet) several different partitions can be observed, including the most trivial case: the one considering one single subsystem in which all resource places are internal to it; i.e., the whole PC²R net. The selection criteria for one or another possible partition into subsystems depend heavily on the nature of the abstracted multithreaded software system and the will of its designer. It is beyond the scope of this thesis to deepen in a particular application approach; indeed, they can lead to very disparate families of correction techniques.

For example, the system designer may wish to optimise the concurrence of a par-

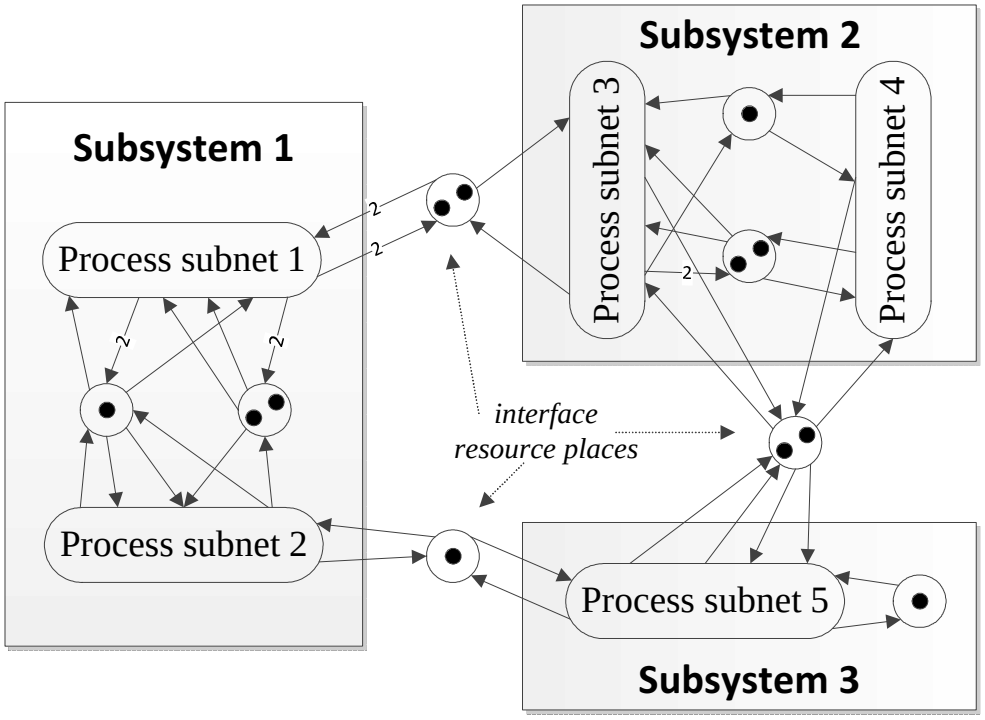



**Figure 3.27**: Decomposition of a PC$^2$R net model into subsystems with shared resources

ticular set of (types of) threads, which suggests grouping them into a single subsystem to be analysed and corrected. This may be due to those threads being strongly coupled in terms of their functionality or being critical with respect to the overall system performance (e.g., they implement business logic). From a different perspective, it should be noted that certain correction techniques are closely linked to the concept of locality, and that the natural implementation of the resulting deadlock correction mechanisms (e.g., monitors, semaphores) do not apply to systems in which the threads are distributed on different machines. In such cases, the designer may usually feel inclined to adjust the partition into subsystems to the physical constraints of the system deployment. In essence, the division into subsystems is a process that should be generally directed or assisted by the system designer, and can strongly determine the structure and control logic of the resulting system. Note also that the process described above is potentially recursive in the style of divide-and-conquer algorithms, though this again depends on the designer judging the particular conditions and properties observed or desired in the system and its environment.

The principle of *modularity* of a PC$^2$R model is thus the first basic principle in

which the methodology is based. The second principle is the fact that a *mutual exclusion* between threads can always be enforced. This grants a last resort to ensure the liveness of a $PC^2R$ model with a 1-acceptable initial marking. As seen below, this second principle is concomitant with the first one: this solution can be locally applied to address $PC^2R$ subsystems that are elusive from other techniques or to coordinate the sharing of resources between different subsystems in conflict.

Indeed, suppose we have a set of subsystems which are live and reversible. Note that these properties are observed considering each subsystem in isolation but also considering their individual connection to the interface resource places. In this case, the introduction of a mutual exclusion restriction between these subsystems from the initial marking obviously ensures that the system resulting from its composition is, on the whole, live and reversible as well.

This leads to a third principle for the analysis and correction of such models: that of *locality*. The designer can focus on obtaining locally live and reversible subsystems to address the final problem of obtaining a globally live and reversible system. As discussed in the paragraph above, the second principle grants that this last bit is ultimately possible. This does not necessarily mean that the process should always end in the sequencing of the subsystems through mutual exclusion once all those have been individually corrected. On the contrary, we have observed that the solution of the problems of non-liveness in them often makes the overall system live. Consequently, it is not necessary to sequentialise their execution in those cases. Recall that, in Theorem 3.15, a tool was provided to verify whether a $PC^2R$ net with a 1-acceptable initial marking is live. Of course, in order to apply it globally on the entire net, all the corrected subsystems must remain as members of that class of net systems after entering the resulting control logic. In case of failure of the test condition (or in case you cannot apply the tool as the whole net falls outside the class of $PC^2R$ nets with a 1-acceptable initial marking) the correction process could finish with the sequentialisation of the larger live and reversible subsystems.

This type of approach is novel, to the best of our knowledge, in the field of RASs, and it is powerfully versatile beyond the scope of this PhD thesis. Suppose, for example, that a particular subsystem was a marked graph (an augmented marked graph considering the interface resource places [CX97]). Observe that in the methodology described above there is no assumption on the nature of the analysis and synthesis techniques which are locally applied. Considering that the subsystem is reasonably small-sized, it is possible to apply synthesis techniques based on the exploration of the state space of that specific subsystem without the need to do so on the whole system. As far as the resulting local subsystem is live and reversible, and its interface resource places are 1-acceptable with respect to the rest of the net, a similar methodology can still be applied.

However, it is previously stated that all the subsystems must be reversible for this

strategy to work. Although liveness and reversibility are paired for RAS net subclasses like S⁴PR, Chapter 2 shows that both properties are decoupled in general for PC²R nets, even when the initial marking is 1-acceptable and there is no resource lending As a result, verifying the reversibility of a PC²R subsystem can be a complicated problem.

At this point, we retake the testing condition for liveness established by Theorem 3.15. While this is a sufficient (but not necessary) condition, in the following pages it is shown that this same testing condition also verifies that the net system is reversible. To this end, we first prove the following theorem, which is analogous (both in wording and in its demonstration) to Theorem 3.10, but replacing the non-liveness property by that of non-reversibility:

**Theorem 3.32.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, be a PC²R net system with a 1-acceptable initial marking. If $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-reversible (i.e., if $\mathbf{m_0}$ is not a home state) then $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that the set of $\mathbf{m}$-process-enabled transitions is non-empty, and every $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled.*

*Proof.* If $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-reversible then there exist a firing sequence $\sigma$ and a reachable marking $\mathbf{m_1} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m_1}$ but $\mathbf{m_0} \notin \mathrm{RS}(\mathcal{N}, \mathbf{m_1})$.

Let $\mathcal{B}_\mathcal{N}$ be the set of the elementary iteration blocks of $\mathcal{N}$, and let $T_{\text{trigger}}$ be the set of transitions that trigger an elementary iteration block of $\mathcal{N}$, i.e. $T_{\text{trigger}} = \bigcup_{(p, P_{\text{SM}} T_{\text{SM}}) \in \mathcal{B}_\mathcal{N}} p^\bullet \cap T_{\text{SM}}$. Now we start at $\langle \mathcal{N}, \mathbf{m_1} \rangle$ and, as far as there exist enabled transitions in the set $T \backslash T_{\text{trigger}}$, we select one of them (any) and fire it. We successively repeat this operation until no transition in the set is firable. Since the subnet induced by $P, T \setminus T_{\text{trigger}}$ is acyclic and every path ends in $P_0$, this eventually happens, and a marking $\mathbf{m}$ is reached such that either no token remains in the process places, i.e. $\mathbf{m}[P_{\text{S}}] = \mathbf{0}$, or for each token remaining, the output transitions of its process place are disabled because of the lack of some resource(s).

In the first case, $\mathbf{m_0} = \mathbf{m}$, but this is impossible since $\mathbf{m_0} \notin \mathrm{RS}(\mathcal{N}, \mathbf{m_1})$. Therefore, only the second case is possible, i.e., the set of $\mathbf{m}$-process-enabled transitions is non-empty, and every $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled. □

The live and reversible net system in Fig. 3.21 proves that the reverse of Theorem 3.32 is not true in general for PC²R nets. Let us remind that this PC²R net with a 1-acceptable initial marking has one reachable marking that holds the condition in Theorem 3.32: the marking is $[\mathrm{A1}, \mathrm{B1}, \mathrm{C0}, \mathrm{R3}]$.

On the other hand, Corollary 2.43 states that, given a PC²R net system with an 1-acceptable initial marking, if it is non-live then it is also non-reversible. The non-live net system depicted in Fig. 2.13 serves as an example of this. Observe that the marking $[\mathrm{A1}, \mathrm{B1}, \mathrm{BOWL}]$ holds the condition in Theorem 3.32.

However, Property 2.39 states that the reverse of Corollary 2.43 is false; even for the S⁵PR subclass. The net system in Fig. 2.16 is non-reversible yet it

is live. The reachability graph depicted in Fig. 2.17 reveals several markings which hold the condition in Theorem 3.32 (e.g., the marking $\mathbf{m} \equiv [\mathrm{A0}, \mathrm{A4}, \mathrm{B0}, \mathrm{B1}, \mathrm{R3}, \mathrm{R4}, \mathrm{R5}, \mathrm{R6}, \mathrm{R9}, \mathrm{R10}, \mathrm{R11}, \mathrm{R13}, \mathrm{R14}]$, in which the set of $\mathbf{m}$-process-enabled transitions is $\{\mathrm{T5}, \mathrm{T26}\}$, but all of them are $\mathbf{m}$-resource-disabled, and only T14 is firable).

Theorem 3.32 lets us prove that the test condition expressed by Theorem 3.15 is also useful for testing the system reversibility. In other words, the theorem can be reformulated in the following way:

**Theorem 3.33.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an $PC^2R$ net system with a 1-acceptable initial marking, and $T_{\text{trigger}}$ be the set of transitions that trigger an elementary iteration block of $\mathcal{N}$, i.e., $T_{\text{trigger}} = \bigcup_{(p, P_{\text{SM}}, T_{\text{SM}}) \in \mathcal{B}_{\mathcal{N}}} p^{\bullet} \cap T_{\text{SM}}$. If the net is non-live or non-reversible, then there exists a marking $\mathbf{m} \in \text{PRS}(\mathcal{N}, \mathbf{m_0})$ such that the following set of inequalities has, at least, one solution:*

$$\begin{aligned} &\min \textstyle\sum_{p \in P \setminus P_0} \mathbf{m}[p] \\ &s.t. \quad \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \\ &\qquad \mathbf{m} \geq \mathbf{0}, \boldsymbol{\sigma} \in \mathbb{N}^{|T|} \\ &\qquad \text{System } (3.3) \end{aligned}$$

*Proof.* If the net is non-live, Theorem 3.15 grants that the condition holds. On the other hand, if $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-reversible, for analogous reasons than stated for Theorem 3.10, there exists a marking $\mathbf{m} \in \text{RS}(\mathcal{N}, \mathbf{m_0})$ such that the above set of inequalities has, at least, one solution. Obviously, $\mathbf{m} \in \text{PRS}(\mathcal{N}, \mathbf{m_0})$. □

Summing up, we suggest below a methodological approach designed to correct multithreaded systems modelled through $\text{PC}^2\text{R}$ nets with a 1-acceptable initial marking. This is based on the dissection of the model in small sub-models that can be analysed and corrected with the help of certain tools, and the final composition of a live model. In the next subsection, a basic set of tools for dealing with that type of nets is presented.

### 3.4.3 Opening the RAS toolbox: The set of rules

Onwards we present a set of tools that can be used to correct (subsystems of) a $\text{PC}^2\text{R}$ model that is not live or whose liveness cannot be verified. This basic toolbox relies on the acquired knowledge on RASs and therefore one of the strengths lies on the fact that this is accomplished avoiding an exhaustive exploration of the state space. Obviously the application of any of these tools is unnecessary if the whole model passes the liveness test described in Subsection 3.3.2. The construction of the resulting model must be controlled by the engineer to ensure that the corresponding modifications can be applied in the actual software system.

**Rule 1: Process splitting**

The Example 2.1 developed throughout Chapter 2 illustrates the fact that resource sharing between iterative processes may produce intricate relationships between them. In the case of the net system of Fig. 2.13 corresponding to Example 2.1, liveness cannot be verified with the current tools. First, there exists a marking ($\mathbf{m} \equiv [\mathrm{A1}, \mathrm{B1}, \mathrm{BOWL}]$) such that every $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled, and therefore the test condition in Theorem 3.33 cannot state that the system is live. On the other hand, we have proved in Sect. 3.3 that no siphon becomes insufficiently marked for any reachable marking of this net system. Thus, we can neither state that the system is non-live according to Theorem 3.13.

One possible solution to this problem is to transform the model so that it is easier to analyse. Through the transformation rules described in Sect. 2.5.5, the iterative process subnets of a PC²R net can be split into simpler process subnets without internal cycles. In the resulting net, each process subnet represents an elementary iteration block considered independently. The idle place of every new process subnet must have an initial marking that ensures its implicitness (e.g., equal to the initial marking of the idle place of the iterative process subnet that originated it). Figure 3.28 depicts the result of splitting all the process subnets in the net of Fig. 2.13. The reachability graph is isomorphic to that of Fig. 2.24 since the new idle places A7 and B7 are implicit from the initial marking[3]. In general, the transformation introduced by Rule 1 preserves the reachability space of the original net since all the new places are SIPs and the initial marking makes them implicit.

One significant consequence of applying this transformation rule is that the resource places aggregated to split the process subnets (e.g., the places R_A1 and R_B1 in Fig. 3.28) do not have a 1-acceptable initial marking, but a 0-acceptable one. For this reason it is not possible to count on these new resource places as interface places to partition the system and later sequentialise as described in Subsection 3.4.2. The fact that the initial marking is not 1-acceptable induces a certain order relation between threads regarding the start of their execution: some have to be triggered before others so that the latter can be started. Therefore, forcing a mutual exclusion between threads is no longer a valid strategy to make the system globally live, since some threads cannot be executed in isolation.

However, the transformation still allows to analyse and correct deadlocks which appear locally in certain subsystems, even if some of their interface resource place with other subsystems have not a 1-acceptable initial marking. In many cases, the correction of these subsystems allow to obtain a globally live system. This is the case of the system of Fig. 3.28: Later, we will see how it is possible to apply Rule 4 after

[3]Note that places A7 and B7 are required since, unlike SPQR nets, every process subnet in a PC²R net must have an idle place

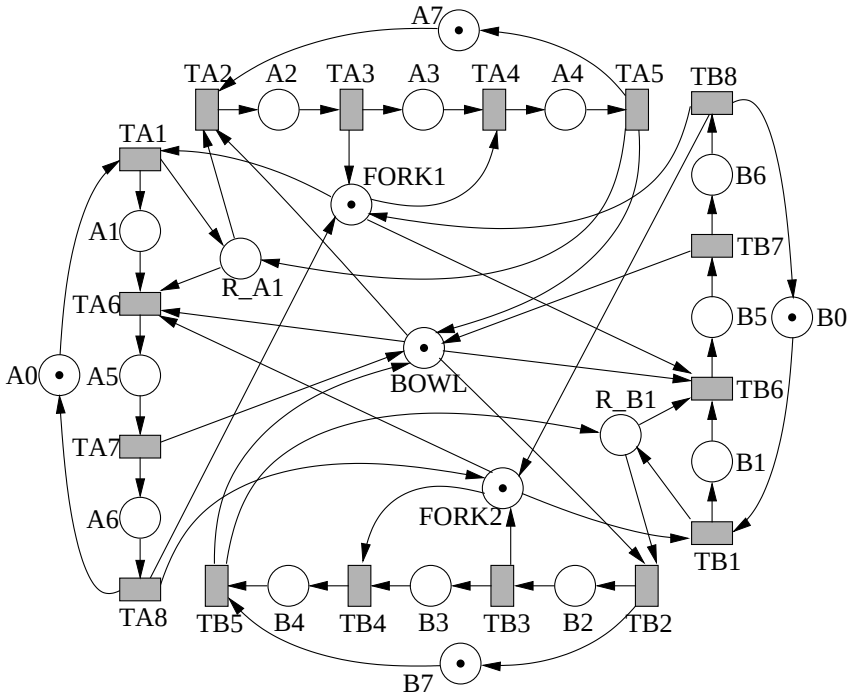



**Figure 3.28**: Process splitting: Transforming the net in Fig. 2.13

applying Rule 1 in order to finally obtain a live system.

One should note, moreover, that it is not always necessary to use Rule 1 to break all loops that appear within a process subnet, but it is only necessary when the tools provided (e.g., Theorem 3.33) cannot decide whether the system is live, or when the tools available do not allow to correct a deadlock.

In order, for the engineer or designer, to bring the changes from model to code, (s)he should be allowed to refactor a fragment of code contained within a `loop`-like statement into an independent thread class. This is often an intricate process because it must respect the context of execution of threads during the execution of the loop. The latter may be achieved through selective replication (e.g,, by program slicing techniques [Wei84, Tip95]) of parts of the original code preceding the loop which affect variables altered or evaluated within that loop.

On the other hand, it is necessary to introduce a mechanism to regulate the start of execution of an instance of the new thread class. One possible implementation goes through inserting a monitor that would be associated with each thread. Then, each new resource place added in the net (e.g., R_A1 and R_B1 in Fig. 3.28) leads to a new boolean condition variable in the monitor, which is assigned the truth value in

the entry point of the loop. Launching a new process instance will occur if a guard is evaluated positively that depends on the condition of entry to the loop and on the value of the new condition variable in the monitor. It is important to note that the designer must verify that both the caller thread (main code) and the thread invoked (nested loop) can physically reside on the same machine and share the same address space so that one can implement this synchronization mechanism between them.

The introduction of these changes in the code allows to consider the new resource places (e.g., places R_A1 and R_B1 in Fig. 3.28) as places that have a physical counterpart in the code, and therefore may be subject to the application of other rules of transformation to simplify not only the analysis but also the correction of the model. In any case, the designer or engineer should be the one who ultimately must decide if (s)he can modify the source code and if (s)he considers reasonable to partition the code as indicated.

Furthermore, changes in the model can also be considered as purely instrumental to lighten model analysis, in which case no modifications on the abstracted code would be required. In this case, the engineer handling the model should be aware of the nature of the new resource places: Certain correction techniques working on this set of places may not be implementable because of them being fully 'virtual' places without an actual counterpart on the code.

**Rule 2: 'Interlaced' p-semiflow cancellation**

Throughout Subsection 3.3.3 we have seen that the emergence of minimal p-semiflows containing more than one resource place is a new scenario that appears in the context of the $\mathrm{PC^2R}$ net class. This scenario often comes paired to the materialisation of non-liveness situations. For this reason it seems desirable to have techniques to break this type of situation. We discuss next a transformation rule that works in this direction. In addition, this will allow us to bring our model closer to simpler net subclasses; i.e., those (like $\mathrm{S^4PR}$ or $\mathrm{SOAR^2}$) that we are able to analyse and correct with the techniques available in the literature.

As we shall see, this scenario arises only when there exists a thread for which the resource usage is permanently interlaced; i.e., it never simultaneously releases all the resources. Obviously this never happens to those systems modelled by subclasses such as $\mathrm{S^5PR}$ or $\mathrm{S^4PR}$, since in such cases the idle place represents a state of minimal resource usage. The transformation rule proposed below, in essence, seeks to induce a de facto ordering in the allocation of those sets of resources that are permanently interlaced.

In net terms, a set of resource places $R \subseteq P_{\mathrm{R}}$ of a $\mathrm{PC^2R}$ net, $|R| > 1$, is supported by a minimal p-semiflow $\mathbf{y_R}$, $R \subseteq \|\mathbf{y_R}\|$, only if there exists a process subnet such that every place in the process subnet is a holder place of some resource place in $R$,

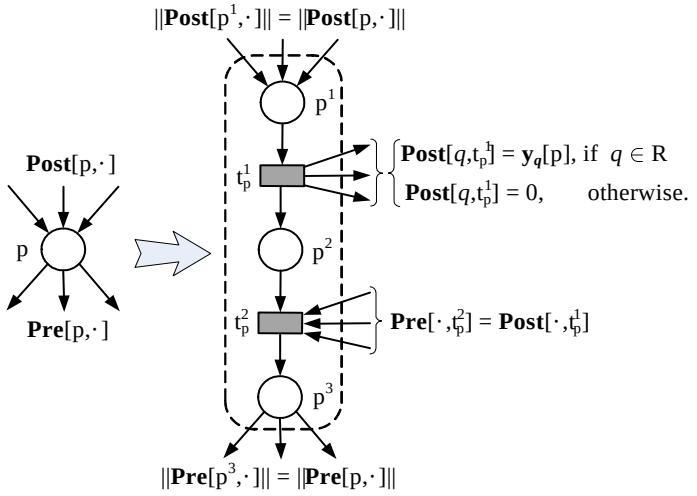



**Figure 3.29**: Rule 2: P-semiflow cancellation

i.e., every such place belongs to at least one minimal p-semiflow $\mathbf{y_r}$ induced by some $r \in R$. This is a condition that stems directly from Lemmas 3.16 and 3.17. Thus, it is possible to construct $\mathbf{y_R}$ from a linear combination in the form described by Lemma 3.16 in which at least the index $K_i$ corresponding to that process subnet is positive. This last condition is necessary to ensure that the support of $\mathbf{y_R}$ is minimal (otherwise, it must be supported by some $\mathbf{y_r}$, with $r \in R$).

Considering the above, one possible strategy can be proposed. In case that a minimal p-semiflow supports a set of multiple resource places $R$, the net would be transformed to enforce the existence of at least one place in each process subnet such that it is no holder place of a resource place in $R$. This can be accomplished in several ways, e.g., splitting one arbitrary place $p$ of the process subnet (where $p \in P_i \cup \{p_{0_i}\}$) into three places and two transitions connected in the form presented in Fig. 3.29. The new place $p^2$ is no holder place of a resource place in $R$, and therefore no minimal p-semiflow supports every place in $R$ plus at least one place in the process subnet. Such a transformation is applied to every process subnet, finally obtaining a net in which no minimal p-semiflow supports every place in $R$.

Figure 3.30 depicts the result of applying the rule of transformation 2 on the net of Fig. 3.23. This rule has been applied to cancel the p-semiflow $2 \cdot \mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + 2 \cdot \mathbf{m}[\mathrm{A1}] + 5 \cdot \mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{B2}] + 2 \cdot \mathbf{m}[\mathrm{C2}] = 6$. This has been accomplished by expanding places A0, B0 and C2.

Note that in this example we have chosen arbitrarily that set of places for they are those in which fewer resources need to be freed. As a general rule, it is sufficient

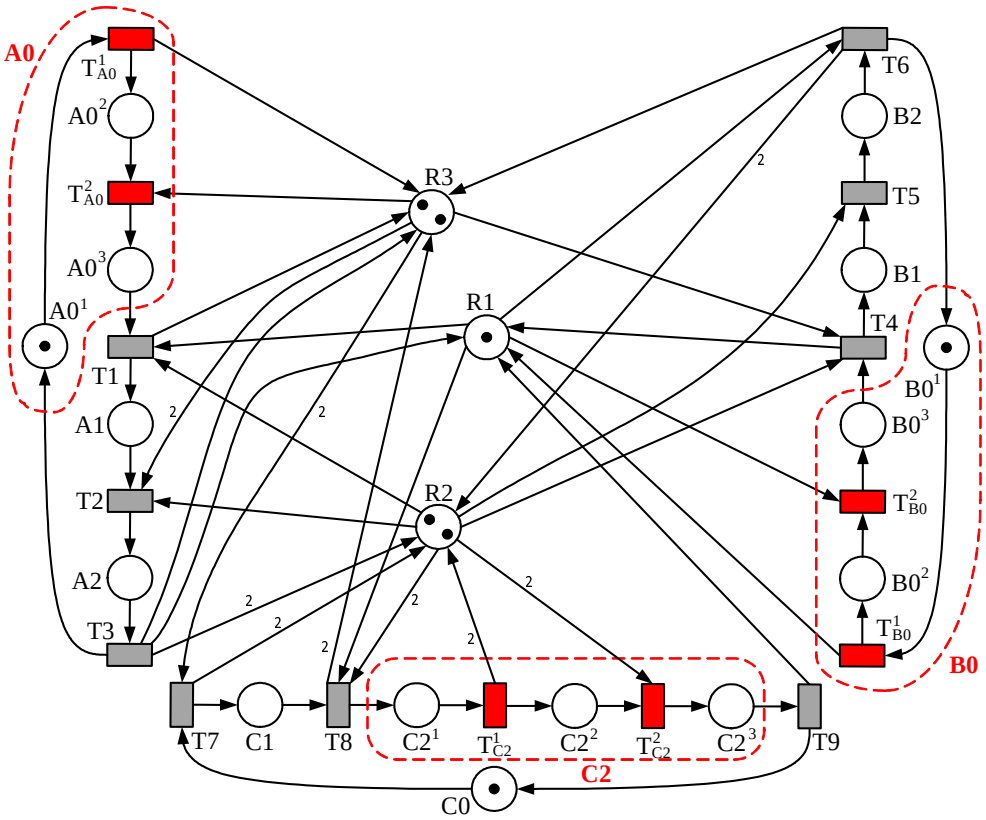



**Figure 3.30**: The net of Fig. 3.23 after applying Rule 2 on A0, B0 and C2

to choose any place, as long as one is chosen for each process subnet. In any case, the choice of places to split must be directed by the designer or system engineer, as he must decide at what point of each thread's code it is acceptable to return some of the allocated resources knowing that they will be requested back immediately after. Also in this decision process, the designer or engineer must consider the nature of the original resource types that are modelled with each resource place in $R$. Depending on their nature, it may be impossible to free some allocated resource in certain states of execution of a thread. Even, it may be impossible to alter the way in which a certain resource type is used throughout some threads considered in their entirety. This is the case, for example, when it comes to new resource places that come from the application of Rule 1 but which have no actual counterpart in the thread's code.

Another interesting observation on the aforementioned example is the fact that the original net has three more minimal p-semiflows supporting several resource places.

Specifically, the following three (other than mentioned above):

$$\begin{aligned}
\mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{R3}] + 2 \cdot \mathbf{m}[\mathrm{A2}] + 2 \cdot \mathbf{m}[\mathrm{C1}] + \mathbf{m}[\mathrm{C2}] &= 3 \\
\mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{R2}] + 2 \cdot \mathbf{m}[\mathrm{A1}] + 3 \cdot \mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{B2}] + 2 \cdot \mathbf{m}[\mathrm{C0}] + 3 \cdot \mathbf{m}[\mathrm{C2}] &= 5 \\
\mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{R3}] + 3 \cdot \mathbf{m}[\mathrm{A2}] + 2 \cdot \mathbf{m}[\mathrm{B1}] + 3 \cdot \mathbf{m}[\mathrm{B2}] &= 4
\end{aligned}$$

It is noteworthy that none of these three minimal p-semiflows persists in the net of Fig. 3.30. In this net, all minimal p-semiflows support no more than one resource place. This means that as the minimal p-semiflow supporting the three resource places is cancelled, the other three minimal p-semiflows supporting more than one resource place are cancelled as well. In general, the selection of the first p-semiflow to be cancelled is important because it can make others also cancelled, although again this selection may be ruled by the nature of the resources abstracted and the restrictions imposed by the software designer or engineer.

Finally, note that the resulting net depicted in Fig. 3.30 is live after applying Rule 2. The reachability graph is not depicted this time due to its size (it has almost one hundred states). The application of Rule 2 does not always returns a live net, however. On the contrary, sometimes the resulting net is not live, yet then other correction rules can be applied to finally enforce liveness in the system. Indeed, Rule 2 works in the direction of narrowing the gap between $\mathrm{PC}^2\mathrm{R}$ nets and those subclasses of nets in which liveness can be seized by structural-based techniques (e.g., $\mathrm{S}^4\mathrm{PR}$ or $\mathrm{SOAR}^2$).

**Rule 3: Increasing the number of available resources**

In Subsection 2.3.4 it was shown that, given a $\mathrm{PC}^2\mathrm{R}$ net, if we are able to increase the number of copies of resources up to some degree then all deadlock problems disappear because the resource places become implicit places and can be removed. On the other hand, an insufficient number of resources can lead to deadlocks.

In many multithreaded software systems, increasing the number of copies of some resources is a suitable solution. However, the feasibility of this approach depends heavily on the nature of the resource types abstracted. If possible, a clever increase on the number of copies of a certain resource type can resolve many adverse situations without reducing the system concurrency. Nevertheless, if liveness is merely accomplished by increasing the number of instances in the resource place then the solution is not structural, but interim. This means that if the initial marking of the idle places is subsequently increased (i.e., if we increase the upper limit of concurrent threads of the same type) the net system can become non-live again.

On the other hand, the number of copies of a resource type can be cleverly increased while privatising its usage by threads of a different type or in a different execution state. By proceeding this way, liveness can be enforced structurally in such a way that rising the upper limit of concurrent threads can no longer violate the liveness property.

The pruning graph of s-regs, which is revisited in Sect. 3.2.4, can be used to accomplish this aim. The technique is originally presented for the subclass of SOAR$^2$ nets, in which, essentially:

1. The process subnets have no internal cycles.

2. The threads use the resources in a *request-before-release* basis, i.e., there is no resource lending.

3. The threads release the resources in the same order in which they request them.

4. Each thread requests/releases only one copy of a resource at a time, i.e. $\forall t \in T : \|\mathbf{C}[P_{\mathrm{R}}, t]\| = 1$.

5. The resource places are binary.

Although in the whole this seems overly-restrictive for modelling many multi-threaded software systems, the technique can still be practical to tackle those non-live subsystems of a PC$^2$R net which fall within the SOAR$^2$ subclass. In this regard, it is worth stressing that:

1. Internal cycles in the process subnets of a PC$^2$R net can be split into new process subnets without internal cycles by repeatedly applying Rule 1.

2. Rule 2 can be used to enforce the existence of a place in each process subnet such that no copy of resource is allocated. Subsequently, this place can be viewed as the idle place of the process subnet as long as the process subnet has no internal cycle.

3. Often in software, ordered release can be enforced (or obliged to the system designer) without affecting thread functionality.

4. In the context of software, most operations on resources request or release only one copy of a resource (yet there exist exceptions, as discussed in Sect. 2.2).

5. Different copies of a single-valued resource type can be modelled by means of a standalone binary resource place per copy.

As in the application of the previous rules, the privatisation of resources must be directed by the application context and/or supervised by the software designer/engineer. To decide on the rule feasibility, not only must the nature of the abstracted resources be considered, but also the kind of operations applied on the resources when allocated. In order to illustrate this, let us consider, e.g., the case of a file which is accessed in mutual exclusion by two threads. If both threads work with it in a read-only basis, the file can be duplicated so that the resource usage is privatised, assigning one copy of the file per thread. But if both request the file for writing on it, then the privatisation of copies can provoke the coexistence of two inconsistent versions of the same file.

On the other hand, the privatisation of resources can be much convenient as a correction strategy when threads are deployed in a distributed environment, considering that the deployment of a centralised control artifact such as a monitor place can be cumbersome, and potentially inefficient, in such contexts, due to intensive message-passing.

**Rule 4: Siphon control through virtual resources**

The addition of virtual resources which act as monitor places preventing the eventual emptying of siphons has been a productive approach to liveness enforcement in the context of FMSs modelled through Petri nets. Such kind of policy is linked to the classic approach in control theory: concurrency is restricted by forbidding those event sequences that lead to the death of some transition; being the death captured by a fatal leak of tokens in some siphon.

The discussion in Sect. 3.3 proves, however, that this strategy is at least partially flawed in the context of multithreaded software systems: Insufficiently marked siphons are not enough to characterise non-liveness for systems modelled through the $PC^2R$ class.

Throughout most of Sect. 3.2, correction techniques which deal with preventing siphons from becoming insufficiently marked siphons have been revisited. These are presented for the $S^4PR$ subclass, yet Theorem 3.13 proves that insufficiently marked siphons are still sufficient (though not necessary) for non-liveness in the more general $PC^2R$ class.

Even in those cases in which the $PC^2R$ net model of the multithreaded software system does not fall within the $S^4PR$ subclass, it is often possible to take advantage of those techniques to correct certain subsystems of it. At this point, it is worth noting that the repetitive application of Rules 1 and 2 can bring the model (or parts of it) close to the $S^4PR$ class. As an example, we retake the net of Fig. 3.28, which is the result of applying Rule 1 twice on the net of Fig. 2.13.

Figure 3.31 depicts a partition of the model into three subsystems, numbered

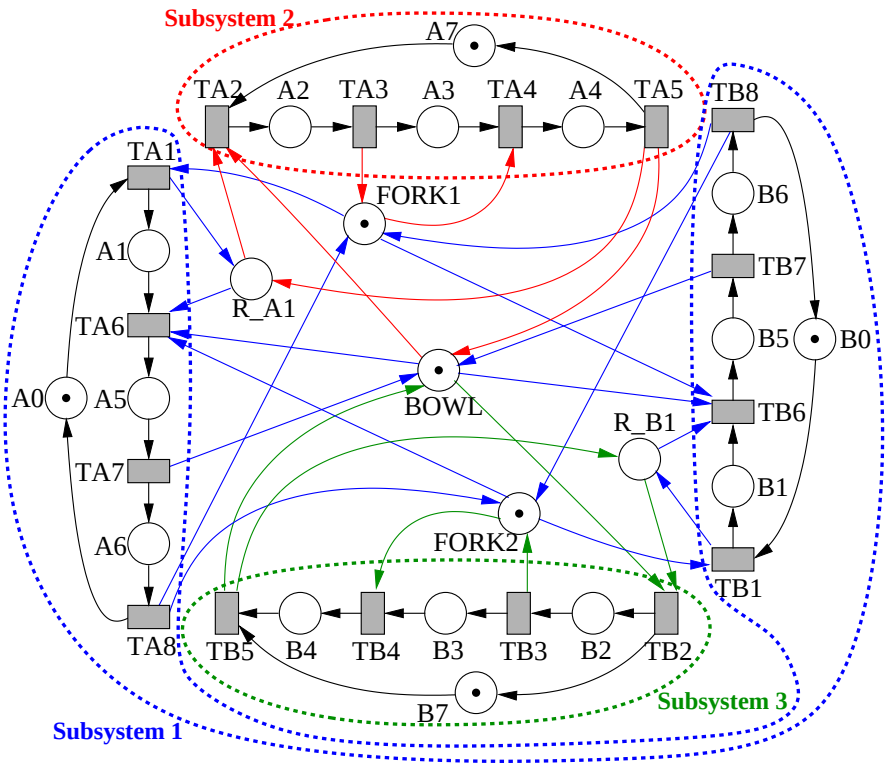



**Figure 3.31**: Decomposition of the net in Fig. 3.28 in subsystems

from 1 to 3. No resource place is internal to a particular subsystem. In other words, from this view, all the resource places are interface resource places: places R_A1 and FORK1 connect Subsystems 1 and 2, places R_B1 and FORK2 connect Subsystems 1 and 3, and place BOWL connects the three of them.

Subsystem 1 (considered along with the arcs from/to the interface resource places) is illustrated in Fig. 3.32[4]. A quick look reveals that the subsystem is not exactly a $S^4PR$ net subsystem. Note, however, that the two resource places R_A1 and R_B1 are implicit (since $\mathbf{C}[\text{R\_A1}, T] = \mathbf{C}[\text{A1}, T]$, $\mathbf{m_0}[\text{R\_A1}] = \mathbf{m_0}[\text{A1}]$, $\mathbf{C}[\text{R\_B1}, T] = \mathbf{C}[\text{B1}, T]$ and $\mathbf{m_0}[\text{R\_B1}] = \mathbf{m_0}[\text{B1}]$). Therefore, those places can be removed to analyse the properties of liveness and reversibility, and the resulting subnet is a $S^4PR$ net with an acceptable initial marking.

Applying Algorithm 3.1, siphon $D = \{\text{FORK1}, \text{FORK2}, \text{A5}, \text{A6}, \text{B5}, \text{B6}\}$ is obtained. This minimal siphon becomes insufficiently marked at the reachable marking $\mathbf{m} = [\text{A1}, \textit{(R\_A1,)}\ \text{B1}, \textit{(R\_B1,)}\ \text{BOWL}]$, which can be reached after firing transitions

[4]Please ignore resource place CP1 for the moment.

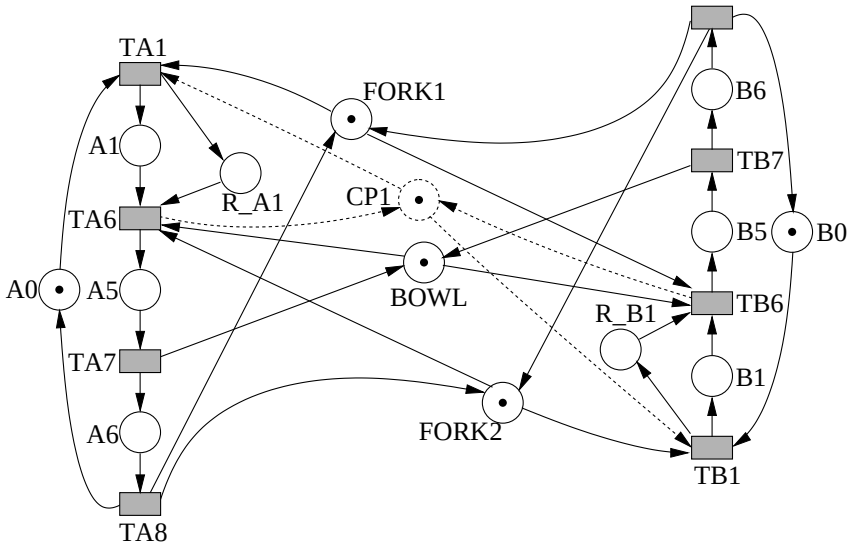



**Figure 3.32**: Subsystem 1 of the net system in Fig. 3.31. Monitor place CP1 prevents the unique bad siphon from becoming insufficiently marked

TA1 and TB1 from the initial marking. Algorithm 3.1 computes the monitor place CP1 depicted in Fig. 3.32, which controls siphon $D$, and the net becomes live (and therefore also reversible, as it is a $S^4PR$ net).

Finally, Fig. 3.33 reproduces the result of inserting the monitor place CP1 into the net of Fig. 3.28. The resulting net is live. Since the initial marking is not 1-acceptable for the resource places R_A1 and R_B1, this cannot be directly verified through the condition in Theorem 3.33. But considering that A1, B1, A7 and B7 are implicit places, we can again remove them, undoing the transformations introduced by the previous application of Rule 1. The result is a $PC^2R$ net with a 1-acceptable initial marking to which we can apply Theorem 3.33, concluding that the resulting net system is live.

**Rule 5: Resource-oriented sequentialisation**

Last but not least, if the liveness property of a certain subsystem cannot be granted, the corresponding threads can be sequentialised. In terms of the model, this can be accomplished by means of a virtual resource which induces a mutual exclusion between all threads in the system. Assuming that the initial marking of the subsystem is 1-acceptable, then every thread can be executed from the initial marking in isolation. Therefore, the resulting net system is live and reversible. However, note that these threads can still concur with other threads belonging to other subsystems (unless the

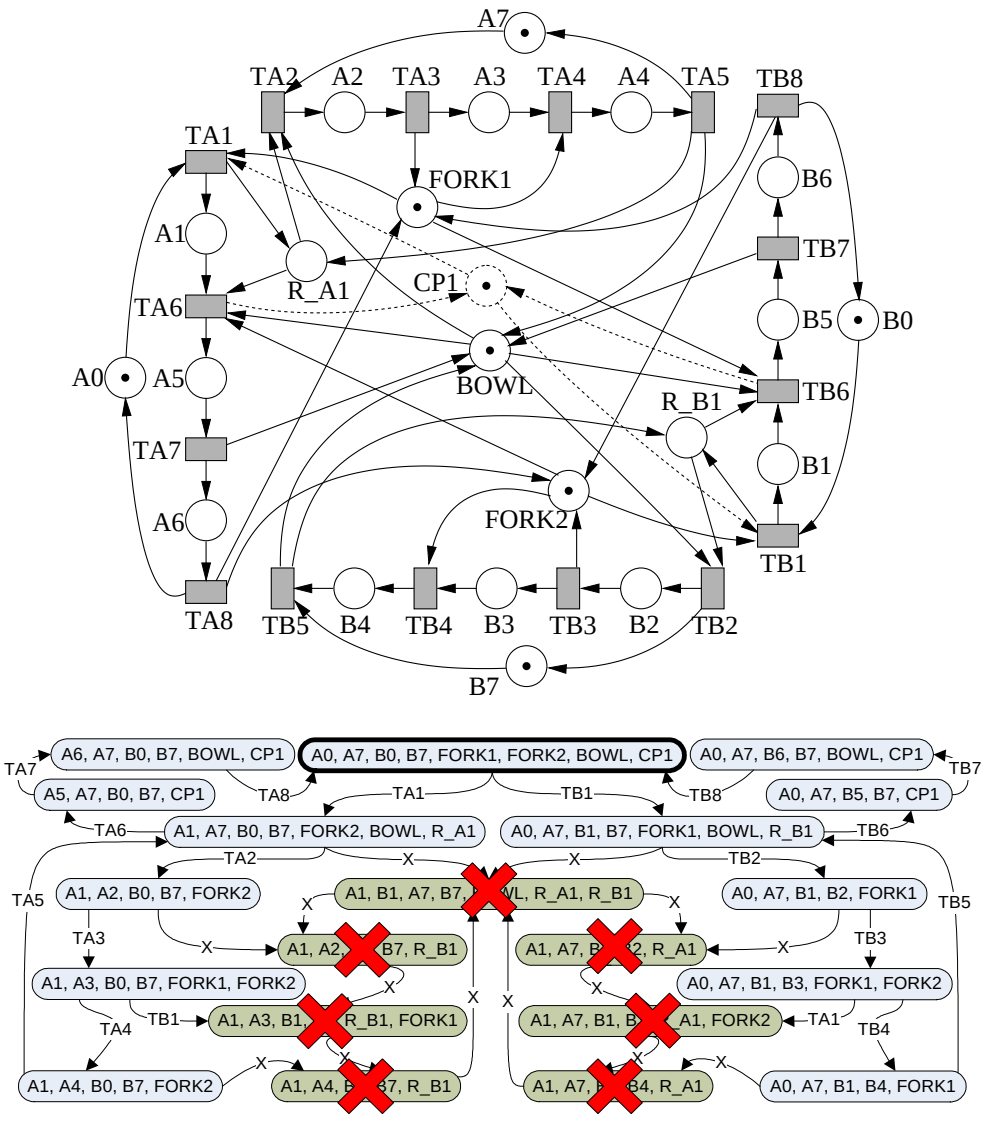



**Figure 3.33**: Controlled version of the net in Fig. 3.28

execution of the subsystems is sequentialised as well).

### 3.4.4 Fitting the jigsaw together

Throughout this chapter, a fresh look and beyond on the current techniques for liveness enforcing on RASs has been portrayed, while showing that, considering the state-of-the-art, there are pitfalls that prevent many interesting RASs from being actually solved. On the other hand, even in the case of highly complex systems, it should be reminded that there is always a solution of last resort which consists in the sequencing of all threads; or, rather, of all those likely to be a source of non-liveness[5].

It has been shown that finding a general characterisation of liveness for the problem of resource allocation in systems abstracted from the multithreaded software world is a very difficult task. Desperate times call for desperate measures: Being guaranteed a solution of last resort, this section has been devoted to providing a battery of techniques which can improve the results that would be obtained by applying the sequentialisation technique. These techniques are based on the acquired knowledge on this kind of systems.

The fundamental principle underlying the application of these techniques is the possibility to observe a complex system as the merge of multiple simpler subsystems. This opens the possibility of start correcting the latter ending up in a global solution that ensures that no issues regarding liveness emerge due to competition for resources between threads.

Starting with the class of $PC^2R$ models (resulting from the abstraction of the resource allocation scheme of a multithreaded software system), five transformation and correction rules have been introduced, which can be applied globally or locally to the subsystems of these models. Their application is dependent on the way the designer or engineer envisions the system, as well as on the nature of the resources involved, and on the fact of the source code of threads being modifiable.

Using this toolbox, some of the more complex examples that had been presented throughout the thesis have been corrected. For example, the net of Fig. 3.30 corrects the liveness problems that were detected in the net of Figure. 3.23 by means of the application of Rule 2. The solution provided by Rule 2 is local in essence, and involves the release of some allocated resources in certain specific points of execution of the threads. Since no thread needs to know the global resource allocation status, or needs access to any centralised monitoring mechanism, this kind of solution is well-suited for being implemented in multithreaded distributed systems.

Besides, the net in Fig 2.13, which models Example 2.1 for the trivial case with two

[5]This solution of last resort works as long as all threads can be executed entirely in isolation, which in terms of the Petri net model is captured by the condition of the initial marking being 1-acceptable.

philosophers, has been corrected, obtaining a live net through successive application of Rules 1 and 4. The net resulting after this process is shown in Fig. 3.34. In this case, we have managed to enforce liveness by introducing a monitor place (CP1) that prevents a philosopher from taking the first fork if his colleague is preparing to eat. In this case it is a global solution that would be, for example, implementable in a centralised system by introducing an $n$-ary semaphore `cp1` (where $n$ is the number of philosophers) which is acquired atomically along with the first fork (i.e., replacing in Algorithm 2.1 the first operation `wait(fork[i])` for an operation `wait(fork[i], cp1)`) and released after taking the second fork (i.e., introducing `signal(cp1)` just above the sentence *Serve spaghetti*).

In general, the proposed strategy for the correction of a system modellable through $PC^2R$ nets for which liveness cannot be verified consists in identifying those troublesome subsystems, in order to correct them first. Once a portion of the model is selected for being studied, it is proposed, if possible, that all interlaced p-semiflows in it are cancelled (through Rule 2), and that some of its iterative processes are split into simpler process subnets (namely, those iterative process interacting with the set of resources that prevent each **m**-process-enabled transition from being enabled) through Rule 1. Both rules approach the (sub-)models to those net classes for which there exist well-established synthesis results.

Once the (sub-)model is simplified, and in case that system liveness cannot be detected yet, Rules 3 or 4 can be repeatedly applied. Rule 3 increases and privatises resource usage, leading to distributed type solutions, while Rule 4 restricts system behaviour by introducing a mutual exclusion between troublesome code sections, leading to centralised type solutions. Ultimately, and in case that no satisfactory solution is found, the execution of those threads causing problems can always be completely sequentialised through Rule 5. Once problems are solved for the simplest subsystems, larger parts of the system can be analysed and corrected, following a similar procedure, until a globally live system is obtained.

## 3.5 Conclusions

The concept of insufficiently marked siphon and its monitorisation through virtual resources form the basis in which a plethora of synthesis techniques of live RASs based on Petri nets have been traditionally grounded. In this chapter, an attempt is made to draw the boundary between the classes of RAS models from which it is not possible to apply the same correction strategies and concepts for RASs.

Throughout the chapter, synthesis methods have been reviewed while bridges have been established with the most general classes of Petri net models for RASs. In parallel, it has been shown that these methods are deeply rooted in liveness characterisations based on the concept of insufficiently marked siphon. Such characterisations are

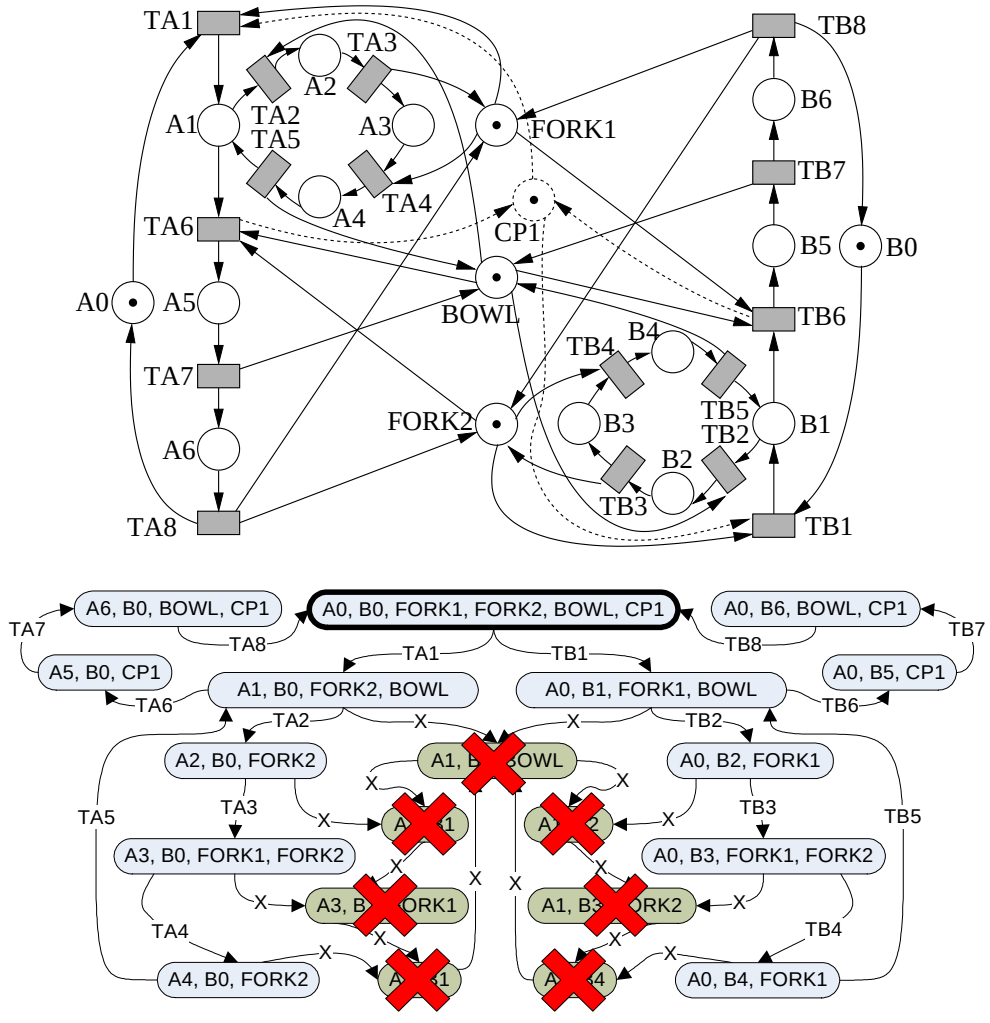



**Figure 3.34**: Controlled version of the net in Fig. 2.13

half behavioral, half structural. The correction methods proposed in the literature usually strive for a structural solution to prevent any anomalous situation regarding liveness from ever happening.

Hopefully, it has been made evident that such techniques cannot be extended directly to the class of $PC^2R$ models, which is aimed to approaching moderately complex multithreaded software systems from a RAS perspective. To this end, it has been proven that the previous liveness characterisations result in only necessary or sufficient conditions for the most general class of $PC^2R$ nets, whereas new properties emerge with respect to previous subclasses. The detailed study of these properties provides the basis to extend those techniques in the future, while evidences that, beyond the subclass of $S^4PR$ nets, liveness is further complicated to approach from a RAS perspective based on such kind of models.

The last part of the chapter attempts to shed some light on this not-so-black prospect. The work previously carried out during this first part of the thesis crystallises in the proposal of a basic methodology that can address the correction of systems modelled through the $PC^2R$ class. The proposal involves the introduction of a toolbox to address the problems of non-liveness, departing from a modular view of the system and taking advantage of the acquired knowledge on this type of systems. This methodology should be adapted to the production context in which the system is implemented, as well as to the nature of the threads and resources in it: A task that should be addressed by the engineer or designer of the system in order to adapt it to the needs and constraints imposed by the problem domain.

# Chapter 4

# Reconstructing the Gadara approach

## Summary

In this chapter, we explore Gadara nets as a significant subclass of $PC^2R$ nets in which the siphon-based characterisation still applies. Some limitations which Gadara nets present for the modelling and automated correction of concurrent software are unveiled. It is also proved that these Petri nets are close to a restricted subclass of $S^4PR$ (a classic, widely-exploited class in the context of FMSs) and provide some related equivalence results. Last but not least, novel formal proofs of the theorems characterising non-liveness in Gadara nets are presented.

## 4.1 Introduction

The $S^4PR$ subclass is a particularly significant Petri nets subclass aimed for the study of RASs. The discussion in Chap. 1 evidenced that, after abstracting details regarding resource allocation, many interesting real world systems from diverse application domains converge in a category of RASs that can be modelled through this particular subclass. Such is the case for many FMSs, for which $S^4PR$ originally emerged as a family of models capable of abstracting versatile plant configurations. On the other hand, $S^4PR$ is the most general subclass of the $S^nPR$ family for which there exists a known structural (siphon-based) liveness characterisation. In this sense, the $S^4PR$ subclass has traditionally established the limit from which structural techniques can be successfully exploited from an RAS perspective.

Consequently, adhering to the family of models induced by the $S^4PR$ specification is a key issue in order to profit from these preexisting analytical results without significant shifts, and successfully approach new application domains from an RAS perspective. Indeed, the assumption of model syntax restrictions over the $S^4PR$ class (albeit with a physical meaning) is a common practice in many application contexts, as discussed in Chap. 1. On the other hand, the complexity of multithreaded software systems led us in Chap. 2 to the introduction of the $PC^2R$ class: a new Petri net class which not only does not restrict, but generalises previous models in order to properly tackle the problem in this domain. This ultimately complicates the finding of a general structural characterisation, as discussed in the previous chapters.

On the contrary, Gadara nets [WLR+09] were introduced as an attempt to import the strengths of liveness enforcing techniques usually applied in FMS into the software domain. However, adding support for internal cycles to the control flow of the processes was esteemed necessary. This was accomplished at the expense of introducing syntactical restrictions to the model with some physical meaning. Unfortunately, those strong restrictions prevent these techniques from being successfully applied in most multithreaded software systems.

In this chapter we approach Gadara nets from different points of view, in order to show their relation to $PC^2R$ nets, consolidate some results which were not formally proved yet, as well as illustrate the strengths and drawbacks that such models present in order to tackle the RAP in multithreaded software systems.

In Section 4.2, Gadara nets, as well as some of their limitations for modelling multithreaded software, are reviewed. In Sect. 4.3, a formal proof of the existence of a structural liveness characterisation for Gadara nets is presented. It is worth noting that, at the time this proof was presented [LGC11], no formal proof was, as far as we know, published yet. Finally, in Sect. 4.4, the equivalence between Gadara nets and a restricted subclass of $S^4PR$ is proven with respect to their correction through net state equation-based structural methods such as that reviewed in Sect. 3.2.2.

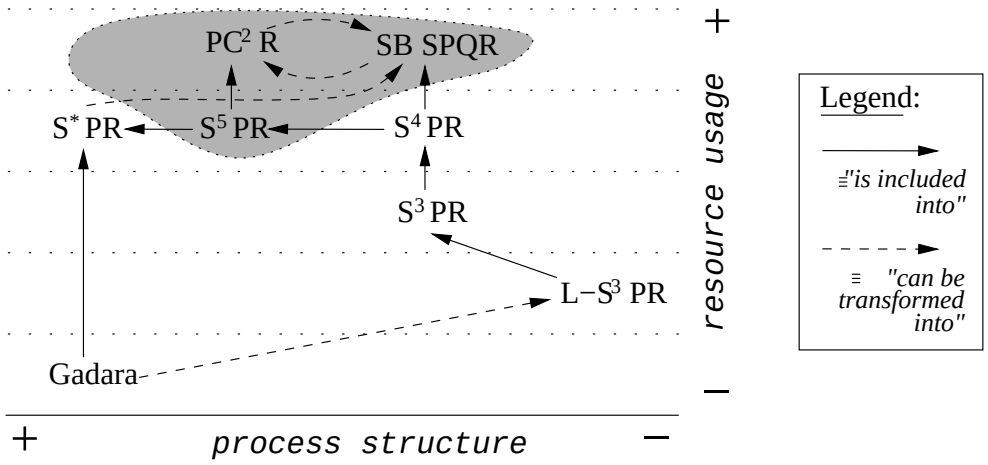



**Figure 4.1**: Inclusion relations between Petri net classes for RASs (second update of Fig. 1.5)

## 4.2 The Gadara approach

Gadara nets belong to the family of Petri nets conceived for modelling RASs. They are modular nets that generalise the $S^4PR$ class in allowing general state machines but constrain the $S^4PR$ class in forbidding the allocation of resources in conflicting transitions inside the state machines (i.e., there is no inclusion relation between these two net classes). A more technical constraint is related to the weights of the minimal p-semiflows associated to resources, which are equal to one. This means that an active thread at most uses one copy per type of resource.

Figure 4.1 completes the whole picture presented in Chap. 2 by portraying the inclusion relations of Gadara nets with the rest of Petri net classes for Sequential RASs.

The formal definition, as originally presented [WLR+09], follows.

**Definition 4.1.** *[WLR+09] Let $I_{\mathcal{N}}$ be a finite set of indices. A Gadara net is a connected ordinary pure P/T net $\mathcal{N} = \langle P, T, F \rangle$ where:*

1. *$P = P_0 \cup P_S \cup P_R$ is a partition such that:*
   - *(a)* [idle places] $P_0 = \bigcup_{i \in I_{\mathcal{N}}} \{p_{0_i}\}$.
   - *(b)* [process places] $P_S = \bigcup_{i \in I_{\mathcal{N}}} P_i$, *where:* $\forall i \in I_{\mathcal{N}} : P_i \neq \emptyset$ *and* $\forall i, j \in I_{\mathcal{N}} : i \neq j,\ P_i \cap P_j = \emptyset$.
   - *(c)* [resource places] $P_R = \{r_1, ..., r_n\}, n > 0$.

2. $T = \bigcup_{i \in I_{\mathcal{N}}} T_i$, *where* $\forall i \in I_{\mathcal{N}}, T_i \neq \emptyset$, *and* $\forall i, j \in I_{\mathcal{N}}, i \neq j, T_i \cap T_j = \emptyset$.
3. *For all* $i \in I_{\mathcal{N}}$ *the subnet generated by restricting* $\mathcal{N}$ *to* $\langle \{p_{0_i}\} \cup P_i, T_i \rangle$ *is a strongly connected state machine. This is called the* $i$*-th* process subnet.
4. *For all* $p \in P_{\mathrm{S}}$*: if* $|p^{\bullet}| > 1$, *then* ${}^{\bullet}(p^{\bullet}) = \{p\}$.
5. *For each* $r \in P_{\mathrm{R}}$, *there exists a unique minimal p-semiflow* $\mathbf{y_r} \in \mathbb{N}^{|P|}$ *such that* $\{r\} = \|\mathbf{y_r}\| \cap P_{\mathrm{R}}$, $\|\mathbf{y_r}\| \cap P_0 = \emptyset$, $\|\mathbf{y_r}\| \cap P_{\mathrm{S}} \neq \emptyset$ *and* $\mathbf{0} \lneq \mathbf{y_r} \lneq \mathbf{1}$.
6. $P_{\mathrm{S}} = \bigcup_{r \in P_{\mathrm{R}}} (\|\mathbf{y_r}\| \setminus \{r\})$.

From the previous definition, it is clear that the novelty with respect to $S^4PR$ is in points 3, 4 and 5. The first one allows state machines with internal cycles, because an important constraint has been removed: the constraint imposing that all circuits in an $S^4PR$ net contain the idle place. This implies that, formally, Gadara is not a subclass of $S^4PR$. Point 4 constrains the definition of $S^4PR$ imposing that resources cannot be allocated in the transitions belonging to a conflict set. A similar situation is given in point 5 where p-semiflows are 0,1-valued. These two points can lead to think that Gadara is a subclass of $S^4PR$, but point 3 denies this possibility.

Later we will see that, from a behavioural point of view, Gadara nets merely address a subclass of the behaviours described by $S^4PR$ nets thanks to the entanglement of the generalisation of point 3 and the constraints of points 4 and 5. This enables the simulation of any Gadara net by means of a $S^4PR$ net obtained from it. This is epitomised in a set of transformation rules presented in Subsection 4.4.

The next definition was originally included as an extra condition to Definition 4.1 [WLR+09]. For coherence reasons with our previous works, it has been extracted, neatly separating the net structure and the initial marking. Note that this definition presents the other fundamental difference with the class of $S^4PR$ systems: in Gadara systems, resource places are binary. This is the deep reason related to the {0-1}-valued p-semiflows.

**Definition 4.2.** *[WLR+09] Let* $\mathcal{N} = \langle P, T, F \rangle$ *be a Gadara net. An initial marking* $\mathbf{m_0}$ *is acceptable for* $\mathcal{N}$ *iff* $\mathbf{m_0}[P_0] \geq \mathbf{1}, \mathbf{m_0}[P_{\mathrm{S}}] = \mathbf{0}, \mathbf{m_0}[P_{\mathrm{R}}] = \mathbf{1}$.

Figure 4.2 depicts a Gadara net with an acceptable initial marking. As will be seen later on, the non-liveness of a Gadara net is characterised by the existence of a structural artifact, a *bad siphon*, that eventually gets *insufficiently marked* or empty. This can be prevented by inserting a monitor place which restricts the system behaviour:

**Definition 4.3.** *[WLR+09] Let* $\mathcal{N} = \langle P, T, F \rangle$ *be a Gadara net. A controlled Gadara net is a connected generalised pure P/T net* $\mathcal{N}_c = \langle P \cup P_{\mathrm{C}}, T, F \cup F_{\mathrm{c}}, W_{\mathrm{c}} \rangle$ *such that, in addition to all conditions in Definition 4.1 for* $\mathcal{N}$*, we have:*

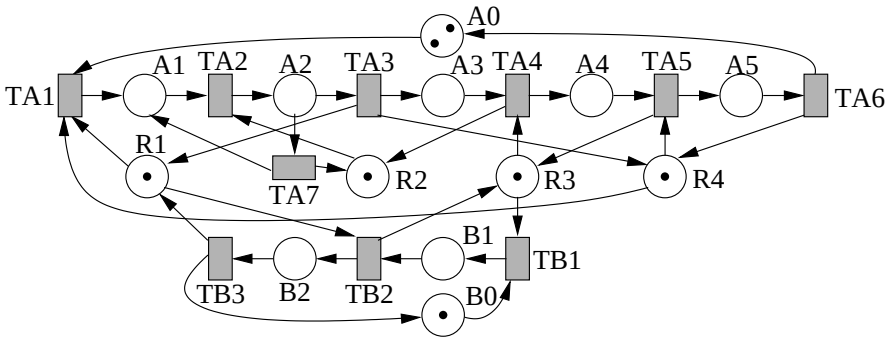



**Figure 4.2**: Non-live Gadara net system

7. *For each* $p_c \in P_C$*, there exists a unique minimal p-semiflow* $\mathbf{y}_{\boldsymbol{p_c}} \in \mathbb{N}^{|P \cup P_C|}$ *such that* $\{p_c\} = \|\mathbf{y}_{\boldsymbol{p_c}}\| \cap P_C$*,* $\|\mathbf{y}_{\boldsymbol{p_c}}\| \cap P_R = \emptyset$*,* $\|\mathbf{y}_{\boldsymbol{p_c}}\| \cap P_0 = \emptyset$*,* $\|\mathbf{y}_{\boldsymbol{p_c}}\| \cap P_S \neq \emptyset$ *and* $\mathbf{y}_{\boldsymbol{p_c}}[p_c] = 1$*.*

Observe that a controlled Gadara net is not a Gadara net in general. This is because the condition that is imposed to the p-semiflows of the control places allow p-semiflows which are not necessarily {0,1}-valued. This is a major problem if we are trying to design synthesis techniques of an iterative nature similar to those presented for $S^4PR$ nets but working only in the context of the Gadara nets (remember that iterative synthesis methods of $S^4PR$ nets cannot be applied to Gadara or controlled Gadara nets because of the internal cycles of the state machines):

**Definition 4.4.** *[WLR+ 09] Let* $\mathcal{N}_c = \langle P_0 \cup P_S \cup P_R \cup P_C, T, F \cup F_c, W_c \rangle$ *be a controlled Gadara net. An initial marking* $\mathbf{m_0}$ *is acceptable for* $\mathcal{N}_c$ *iff* $\mathbf{m_0}[P_0] \geq \mathbf{1}$*,* $\mathbf{m_0}[P_S] = \mathbf{0}$*,* $\mathbf{m_0}[P_R] = \mathbf{1}$ *and for every* $p_c \in P_C, p \in P_S : \mathbf{m_0}[p_c] \geq \mathbf{y}_{\boldsymbol{p_c}}[p]$*.*

The previous definition points out that the initial marking of controlled Gadara nets violates, in general, the binary condition of the initial marking of Gadara nets.

The net of Fig. 4.2 has three bad siphons. The minimal siphon $D = \{R1, R2, R3, R4, A2, A5, B2\}$ is empty at the reachable marking $\mathbf{m} = \{A1, B1, A3\}$. This siphon can be controlled by aggregating a control place $p_c$ which would have arcs from TA1 and TA2 with the following non-unitary weights: $\mathbf{C}[p_c, TA2] = -\mathbf{C}[p_c, TA1] = 2$. Those non-unitary arc weights are due to the fact that A1 belongs to the support of the minimal p-semiflow of two different resource places, $\mathbf{y_{R1}}$ and $\mathbf{y_{R4}}$. Out of curiosity, there exists another minimal siphon, $D' = \{R1, R2, R3, A2, A4, B2\}$ which is also empty at $\mathbf{m}$. If $D'$ is controlled then it is obtained a control place with unitary arcs only, and the resulting controlled net is live. This, of course, does not always happen. As a result, $\mathbf{y}_{\boldsymbol{r}} \in \{0,1\}^{|P \cup P_C|}$ but $\mathbf{y}_{\boldsymbol{p_c}} \in \mathbb{N}^{|P \cup P_C|}$, in general.

Please note that, from now onwards, the term Gadara nets will be used for referring to controlled Gadara nets.

As discussed in Chap. 2, very complex phenomena can appear when internal cycles are allowed in the control flow of the processes. Observe that controlled Gadara nets respond to a very general class of nets with internal cycles, and that the claim is true even in safe nets with no resource lending [LGC12] or overlapped (i.e., not nested) internal cycles, as the net system in Fig. 2.13 reveals. In this case, no *bad* siphon ever becomes insufficiently marked, even when the net is non-live. Thus, the classic structural characterisation [TGVCE05] does not work in the general context.

The "good behaviour" of Gadara nets originates from the fact that conflicts induced by process places are free-choice. This seems to approximate these models to the kind of systems with linear processes, such as the L-$S^3$PR class [EGVC98]. This modelling assumption can however be overrestrictive for modelling software systems: some kind of software cannot be modelled with Gadara nets, due to the usage of non-blocking allocation primitives, which are supported by (e.g.) POSIX locks. A similar argument can be applied when conditional statement expressions must be evaluated atomically. Additionally, general, non-binary semaphores are not supported, and the case of `signal` operations preceding `wait` operations is neither considered. These uncovered aspects in the modelling of real software restrain an automated translator to Petri nets from working, unless the kind of programs that the engineer can construct is constrained.

## 4.3 Liveness characterisation

In 2009, Yin Wang et al. [WLR+09] enunciated a liveness characterisation for Gadara nets based on the existence of an insufficiently marked siphon at a reachable marking (there captured by the equivalent concept of *resource-induced deadly marked siphon*). In 2011, we presented a formal proof of that characterisation [LGC11], which had not been dealt before. In fact, no formal proof of their claim seems to have been published by these authors until very recent times [LWC+13]. Instead, it was originally stated that the same proof strategy to that followed for $S^4$PR nets can be extended for Gadara nets [WLR+09], albeit this is a dubious claim. Please mind that conflicts induced by process places are not free-choice, in general, for $S^4$PR nets. This is a restriction imposed by Gadara nets that must be taken into account in the proof: otherwise, it would be also generalisable for nets like the one in Fig. 2.13. Hence, the liveness theorem needed further proof in order to be unequivocally validated.

Our original proof is reproduced next. Indeed, it must be noted that the liveness theorem is in fact proved on a superclass of controlled Gadara nets. This superclass is defined next.

**Definition 4.5.** *An extended Gadara (e-Gadara) net is a connected* generalised *pure P/T net* $\mathcal{N} = \langle P, T, F, W\rangle$ *(or, equivalently,* $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$*) following Definition 4.1 except for condition 5, which is generalised as follows:*

5. *For each* $r \in P_{\mathrm{R}}$*, there exists a unique minimal p-semiflow* $\mathbf{y}_{\boldsymbol{r}} \in \mathbb{N}^{|P|}$ *such that* $\{r\} = \|\mathbf{y}_{\boldsymbol{r}}\| \cap P_{\mathrm{R}}$*,* $\|\mathbf{y}_{\boldsymbol{r}}\| \cap P_0 = \emptyset$*,* $\|\mathbf{y}_{\boldsymbol{r}}\| \cap P_{\mathrm{S}} \neq \emptyset$ *and* $\mathbf{y}_{\boldsymbol{r}}[r] = 1$*.*

**Definition 4.6.** *Let* $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$ *be an e-Gadara net. An initial marking* $\mathbf{m_0}$ *is acceptable for* $\mathcal{N}$ *iff* $\mathbf{m_0}[P_0] \geq \mathbf{1}$*,* $\mathbf{m_0}[P_{\mathrm{S}}] = \mathbf{0}$ *and* $\forall r \in P_{\mathrm{R}}, p \in P_{\mathrm{S}} : \mathbf{m_0}[r] \geq \mathbf{y}_{\boldsymbol{r}}[p]$*.*

Please note that the control places are included in $P_{\mathrm{R}}$ in Definition 4.5 (therefore, no subset $P_{\mathrm{C}}$ is defined). Note that this can be done since the minimal p-semiflows induced by the resource places in an e-Gadara net are more general than those induced by both the resource places and the control places of controlled Gadara nets. In other words, the class e-Gadara is essentially a generalisation of that of controlled Gadara nets. As a result, Definition 4.4 is consistent with Definition 4.6.

Some more definitions follow which will be instrumental both for the liveness theorems enunciations and proofs.

**Definition 4.7.** *Let* $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$ *be an e-Gadara net. The* set of holders *of* $r \in P_{\mathrm{R}}$ *is the support of the minimal p-semiflow* $\mathbf{y}_{\boldsymbol{r}}$ *without the place* $r$*:* $\mathcal{H}_r = \|\mathbf{y}_{\boldsymbol{r}}\| \setminus \{r\}$*. This definition can be extended to sets of resources* $A \subseteq P_{\mathrm{R}}$ *in the following way:* $\mathcal{H}_A = \bigcup_{r \in A} \mathcal{H}_r$*.*

**Definition 4.8.** *Given a marking* $\mathbf{m}$ *in an e-Gadara net, a transition* $t$ *is said to be:*

- $\mathbf{m}$*-process-enabled (*$\mathbf{m}$*-process-disabled) iff it has (not) a marked input process place, i.e.* $t \in (\|\mathbf{m}\| \cap P_{\mathrm{S}})^{\bullet}$ *(i.e.,* $t \notin (\|\mathbf{m}\| \cap P_{\mathrm{S}})^{\bullet}$*).*

- $\mathbf{m}$*-resource-enabled (*$\mathbf{m}$*-resource-disabled) iff its input resource places have (not) enough tokens to fire it, i.e.,* $\mathbf{m}[P_{\mathrm{R}}, t] \geq \mathbf{Pre}[P_{\mathrm{R}}, t]$ *(i.e.,* $\mathbf{m}[P_{\mathrm{R}}, t] \ngeq \mathbf{Pre}[P_{\mathrm{R}}, t]$*).*

Before proceeding with liveness Theorem 4.13 and 4.14, two instrumental and easy lemmas will be addressed.

**Lemma 4.9.** *Every e-Gadara net is consistent.*

*Proof.* The proof is analogous to that provided for $\mathrm{PC}^2\mathrm{R}$ nets (Lemma 2.22), but included for completion. The process subnets of $\mathcal{N}$ are strongly connected state machines and therefore each one is consistent, i.e., every transition $t$ of $\mathcal{N}$ is covered by at least a t-semiflow of the state machine containing $t$. It will be proved that these t-semiflows are also t-semiflows of the net $\mathcal{N}$. Indeed, if $\mathbf{x}$ is a t-semiflow of $\mathcal{N}$ without resources it is enough to prove that $\forall r \in P_{\mathrm{R}} : \mathbf{C}[r, T] \cdot \mathbf{x} = 0$. Taking into account

Definition 4.5, point 5, $\mathbf{C}[r,T] = -\sum_{p\in\|\mathbf{y_r}\|\setminus\{r\}} \mathbf{y_r}[p]\cdot\mathbf{C}[p,T]$, and therefore,

$$\mathbf{C}[r,T]\cdot\mathbf{x} = -\left(\sum_{p\in\|\mathbf{y_r}\|\setminus\{r\}} \mathbf{y_r}[p]\cdot\mathbf{C}[p,T]\right)\cdot\mathbf{x} = -\sum_{p\in\|\mathbf{y_r}\|\setminus\{r\}} \mathbf{y_r}[p]\cdot\mathbf{C}[p,T]\cdot\mathbf{x} = 0.$$

Thus, $\mathcal{N}$ is consistent. □

**Lemma 4.10.** *Let $\langle\mathcal{N},\mathbf{m_0}\rangle$ be an e-Gadara net with an acceptable initial marking. Then, for every $t\in T$, there exists a t-semiflow containing $t$ being realisable from $\mathbf{m_0}$.*

*Proof.* The proof is very similar to that provided for PC$^2$R net systems with 1-acceptable initial markings (Theorem 2.31), although some parts are slightly different. It will be proved that a single token can be extracted from any idle place at $\mathbf{m_0}$ and be freely moved in isolation through its corresponding state machine. Let $M_1$ be the subset of reachable markings such that one and only one process place is (mono-)marked, i.e., $M_1 = \{\mathbf{m}\in \mathrm{RS}(\mathcal{N},\mathbf{m_0}) \mid \exists|p\in P_\mathrm{S} : \mathbf{m}[p]=1, \|\mathbf{m}\|\cap P_\mathrm{S} = \{p\}\}$.

First, every $t\in {P_0}^\bullet$ is enabled at $\mathbf{m_0}$ since ${}^\bullet t\subseteq P_0\cup P_\mathrm{R}$ and, by the definition of acceptable initial marking, $P_0\subset\|\mathbf{m_0}\|$ and $\forall r\in P_\mathrm{R} : \mathbf{m_0}[r]\geq\mathbf{y_r}[q] = \mathbf{Pre}[r,t]$, with $q = t^\bullet\cap P_\mathrm{S}$. By firing $t$ a marking of $M_1$ is reached.

Without loss of generality, let $\mathbf{m}\in M_1$. It will be proved that every $\mathbf{m}$-process-enabled transition $t$ is enabled. If $t^\bullet\cap P_\mathrm{S}=\emptyset$ then $\mathbf{m}[P_\mathrm{R}]\geq\mathbf{0} = \mathbf{Pre}[P_\mathrm{R},t]$. Thus, $\mathbf{m}\xrightarrow{t}\mathbf{m_0}$. Otherwise, let $\{p\} = {}^\bullet t\cap P_\mathrm{S}$ and $\{q\} = t^\bullet\cap P_\mathrm{S}$. Then $\forall r\in P_\mathrm{R} : \mathbf{Pre}[r,t] = \max(\mathbf{y_r}[q]-\mathbf{y_r}[p],0)$. By Definition 4.6, $\forall r\in P_\mathrm{R} : \mathbf{m_0}[r]\geq\mathbf{y_r}[q]$. Then $\mathbf{m}[r] = \mathbf{m_0}[r]-\mathbf{y_r}[p]\geq\mathbf{y_r}[q]-\mathbf{y_r}[p]$. Also, $\mathbf{m}[r]\geq 0$. Thus, $\mathbf{m}[P_\mathrm{R}]\geq\mathbf{Pre}[P_\mathrm{R},t]$; i.e. $\mathbf{m}\xrightarrow{t}\mathbf{m'}$, $\mathbf{m'}\in M_1$.

It has been proven that an isolated token can be carried from $\mathbf{m_0}[P_0]$ to any arbitrary $p\in P$. If $p$ belongs to a circuit, we can take that token and make it travel around the circuit. Since every t-semiflow corresponds to a circuit in a state machine (as proven for the dual case of circuits of marked graphs and p-semiflows [Mur89]), and e-Gadara nets are consistent by Lemma 4.9, the new lemma holds. □

Since a token in a strongly connected state machine can be moved in isolation to any other arbitrary place of the same state machine, the next lemma is pretty intuitive, yet formally proven in the following. Note that $\boldsymbol{\sigma}$ denotes the firing count vector of $\sigma$.

**Lemma 4.11.** *Let $\langle\mathcal{N},\mathbf{m}\rangle$, $\mathcal{N}=\langle P,T,\mathbf{C}\rangle$, be a set of isolated marked strongly connected state machines, and let $P_0\subset P$ be an arbitrary subset of places such that $P_0$ contains one and only one place of each strongly connected state machine. Then there exists at least one firing sequence $\sigma$, $\mathbf{m}\xrightarrow{\sigma}\mathbf{m'}$, such that there exists no t-semiflow $\mathbf{x}\neq\mathbf{0}$ of $\mathcal{N}$, with $\boldsymbol{\sigma}-\mathbf{x}\geq\mathbf{0}$, and $\|\mathbf{m'}\| = P_0$.*

*Proof.* Let $I_{\mathcal{N}}$ be a finite set of indices, and let $\bigcup_{i \in I_{\mathcal{N}}} \mathcal{N}_i$ be the set of isolated strongly connected state machines of $\mathcal{N}$, where $\mathcal{N}_i = \langle \{p_{0_i}\} \cup P_i, T_i, \mathbf{C_i} \rangle$ and $p_{0_i} \in P_0$. For every $p \in P_i$, the path $\pi_p$ is defined as an arbitrary path of minimal length from $p$ to $p_{0_i}$. It will be proved that the union of every path $\pi_p$, for every $p \in P$, is a spanning forest, i.e. it has no circuits. By reduction to absurd: suppose that there exists a circuit containing $p$ and $q$, where $p, q \in P_i$, $p \neq q$. Let $\pi_p = p\,u\,q\,v\,p_{0_i}$ and $\pi_q = q\,w\,p\,z\,p_{0_i}$, where $u, v, w, z \in (P_i \cup T_i)^+$. Since $\pi_p$ ($\pi_q$) is a minimal path, then $|\pi_p| \leq |p\,z\,p_{0_i}|$ ($|\pi_q| \leq |q\,v\,p_{0_i}|$). But then $|\pi_p| > |q\,v\,p_{0_i}| \geq |\pi_q|$ and $|\pi_q| > |p\,z\,p_{0_i}| \geq |\pi_p|$, reaching a contradiction.

Now let us construct the sequence $\sigma$. Let $\sigma_p$ be the projection of the path $\pi_p$ over $T$. Let $\sigma$ be an arbitrary concatenation of the sequences $(\sigma_p)^{\mathbf{m}[p]}$, for every $p \in \|\mathbf{m}\| \cap P_i$. Obviously, $\sigma$ is firable. And, by construction, no subset of transitions contained in $\sigma$ form a circuit. Hence there is no t-semiflow $\mathbf{x}$ with $\boldsymbol{\sigma} - \mathbf{x} \geq \mathbf{0}$, since a t-semiflow is represented by a circuit of a state machine, and viceversa (as proven for the dual case of circuits of marked graphs and p-semiflows [Mur89]). □

**Lemma 4.12.** *Let $\langle \mathcal{N}, \mathbf{m} \rangle$, $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$, be a set of isolated marked strongly connected state machines. Let $p \in P$ be a marked place at $\mathbf{m}$, $\mathbf{m}[p] > 0$, and let $\sigma$ be a firing sequence, $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$, such that $\mathbf{m}'[p] = 0$. Then there also exists a firable sequence $\sigma'$, $\mathbf{m} \xrightarrow{\sigma'} \mathbf{m}'$, with $\boldsymbol{\sigma}' = \boldsymbol{\sigma}$ and $\exists t \in p^\bullet, \sigma'' \in T^* : \sigma' = t\,\sigma''$.*

*Proof.* Since $p \in \|\mathbf{m}\| \setminus \|\mathbf{m}'\|$, there exists at least one transition in $p^\bullet$ which appears once or more times in $\sigma$. Let $\sigma$ be defined as $\sigma = u\,t\,v$, where $u \in (T \setminus p^\bullet)^*$, $t \in p^\bullet$ and $v \in T^*$; i.e., $u$ is the maximal prefix before the first firing of a transition in $p^\bullet$. It will be proved that $\sigma' = t\,u\,v$ is firable from $\mathbf{m}$. It is enough to prove that $t\,u$ is firable, since it implies that a marking $\mathbf{m_2}$ is reached from which $v$ is firable (because it is the same marking $\mathbf{m_2}$ reached when fired the prefix $u\,t$ of $\sigma$). Since $\mathbf{m}[p] > 0$, $t$ can be fired from $\mathbf{m}$, reaching $\mathbf{m_1}$, with $\mathbf{m_1}[p'] \geq \mathbf{m}[p']$, $\forall p' \in P \setminus \{p\}$. Since $\mathbf{m} \xrightarrow{u}$ and every transition $t$ that appears in $u$ holds $p \notin {}^\bullet t$ then $u$ must also be firable from $\mathbf{m_1}$. □

Taking into account the previous results, now we are prepared to prove the following non-liveness characterisation for e-Gadara nets.

**Theorem 4.13.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an e-Gadara net with an acceptable initial marking. $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live iff $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that the set of $\mathbf{m}$-process-enabled transitions is non-empty and each one of these transitions is $\mathbf{m}$-resource-disabled.*

*Proof.* $\Longrightarrow$) Let $\mathbf{m}'$ be a reachable marking such that at least one transition $t$ in $\mathcal{N}$ is dead. Let $\mathcal{N}^{P_S}$ be the net $\mathcal{N}$ without the resource places, and $\mathbf{m}'_{|\mathbf{P}_S}$ ($\mathbf{m_{0|P_S}}$) denote the marking $\mathbf{m}'$ ($\mathbf{m_0}$) restricted to the places of $\mathcal{N}^{P_S}$. Let $\Sigma = \{\sigma \mid \mathbf{m}'_{|\mathbf{P}_S} \xrightarrow{\sigma} \mathbf{m_{0|P_S}}$ and there is no t-semiflow $\mathbf{x}$ with $\boldsymbol{\sigma} - \mathbf{x} \geq \mathbf{0}\}$. By

Lemma 4.11, the set $\Sigma$ is non-empty. Besides, since the unitary vector of dimension $|T|$ is a t-semiflow of $\mathcal{N}^{P_S}$, every $\sigma \in \Sigma$ holds $|\sigma| < \mathrm{K} \cdot |T|$, where $\mathrm{K} = \sum_{p \in P_S} \mathbf{m}[p]$. Consequently, the set $\Sigma$ is finite.

Let $\sigma_1$ be the sequence of $\Sigma$ which has the longest prefix $u$, $\sigma_1 = u\,v$, such that $\mathbf{m}' \xrightarrow{u}$ in $\mathcal{N}$. If $u = \sigma_1$, $\mathbf{m}' \xrightarrow{u} \mathbf{m_0}$. But $t$ would be eventually firable by Lemma 4.10, contradicting the hypothesis that $t$ is dead at $\mathbf{m}'$. Therefore $u \neq \sigma_1$, and $\mathbf{m}' \xrightarrow{u} \mathbf{m}$, $\mathbf{m} \neq \mathbf{m_0}$. Thus, $\mathbf{m}[P_S] \neq \mathbf{0}$. The set of $\mathbf{m}$-process-enabled transitions is non-empty.

Now it is proved that every transition in $(\|\mathbf{m}\| \cap P_S)^\bullet$ is disabled at $\mathbf{m}$. Without loss of generality, an arbitrary $p \in \|\mathbf{m}\| \cap P_S$ is taken. Let $\mathbf{m}_{|\mathbf{P_S}}$ denote the marking $\mathbf{m}$ restricted to $\mathcal{N}^{P_S}$. By Lemma 4.12, there exists $t \in p^\bullet, v'' \in T^*$ such that $v' = t\,v''$ is firable from $\langle N^{P_S}, \mathbf{m}_{|\mathbf{P_S}} \rangle$ with $\mathbf{v} = \mathbf{v}'$. Then the sequence $\sigma_2 = u\,t\,v''$ is firable from $\langle N^{P_S}, \mathbf{m}'_{|\mathbf{P_S}} \rangle$ and belongs to $\Sigma$, since $\boldsymbol{\sigma}_2 = \boldsymbol{\sigma}_1$. But $u\,t$ is not firable from $\mathbf{m}'$ since otherwise $u$ would not be the longest firable prefix of every sequence in $\Sigma$. Since $t$ is $\mathbf{m}$-process-enabled, $t$ must be $\mathbf{m}$-resource-disabled, with ${}^\bullet t \cap P_R \neq \emptyset$. Then, by Definition 4.1, point 4, $|p^\bullet| = 1$. Thus every transition in $p^\bullet$ is $\mathbf{m}$-process-enabled, $\mathbf{m}$-resource-disabled, and so is every transition in $(\|\mathbf{m}\| \cap P_S)^\bullet$.

$\Longleftarrow$) Let $t \in (\|\mathbf{m}\| \cap P_S)^\bullet$. In order to fire $t$ some more tokens are needed in some places belonging to $P_R \cap {}^\bullet t$. Since tokens in the process places cannot progress at $\mathbf{m}$, the marking of such resources can only be changed by activating some idle processes. Let ET be the set of $\mathbf{m}$-process-enabled transitions, let $\mathrm{AP} = {}^\bullet\mathrm{ET} \cap P_S$, and let $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$. It will be proved, by induction over the length of $\sigma$ that: (i) $\|\sigma\| \cap \mathrm{ET} = \emptyset$, and (ii) $\forall p \in \mathrm{AP} : \mathbf{m}'[p] \geq \mathbf{m}[p]$.

Doing so, and since $\mathbf{m}[P_S \setminus \mathrm{AP}] = \mathbf{0}$, it can be deduced that $\forall p \in P_S : \mathbf{m}'[P_S] \geq \mathbf{m}[P_S]$. But then $\forall r \in P_R : \mathbf{m}'[r] = \mathbf{m_0}[r] - \sum_{p \in P_S} \mathbf{m}'[p] \cdot \mathbf{y_r}[p] \leq \mathbf{m_0}[r] - \sum_{p \in P_S} \mathbf{m}[p] \cdot \mathbf{y_r}[p] = \mathbf{m}[r]$. Therefore, no transition of ET can be $\mathbf{m}'$-resource-enabled.

- *Case $\sigma = t$.* Since no transition of ET is enabled at $\mathbf{m}$, then $t \in P_0{}^\bullet$ and then $t \notin \mathrm{ET}$. On the other hand, if $t \notin {}^\bullet\mathrm{AP}$, $\forall p \in \mathrm{AP} : \mathbf{m}'[p] = \mathbf{m}[p]$. If $t \in {}^\bullet\mathrm{AP}$, let $t^\bullet \cap P_S = \{q\} \in \mathrm{AP}$. In this case, $\mathbf{m}'[q] = \mathbf{m}[q] + 1$ and $\mathbf{m}'[p] = \mathbf{m}[p]$ for every $p \in \mathrm{AP} \setminus \{q\}$.
- *General case.* $\mathbf{m} \xrightarrow{\sigma''} \mathbf{m}'' \xrightarrow{t} \mathbf{m}'$, where $\sigma''$, $\mathbf{m}''$ verify the induction hypothesis. But since $\forall p \in \mathrm{AP} : \mathbf{m}''[p] \geq \mathbf{m}[p]$ then $\forall r \in P_R : \mathbf{m}''[r] \leq \mathbf{m}[r]$, so every transition of ET is $\mathbf{m}''$-resource disabled, and $t \notin \mathrm{ET}$. Therefore, $\forall p \in \mathrm{AP} : \mathbf{m}'[p] \geq \mathbf{m}''[p] \geq \mathbf{m}[p]$, and we can conclude.

□

It is worth mentioning that the second half of the proof of Theorem 4.13 is almost literally that presented for $S^4PR$ nets [Tri03]. This is also true for the next theorem:

**Theorem 4.14.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an e-Gadara net with an acceptable initial marking. $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live iff $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ and a siphon $D$ such that $\mathbf{m}[P_\mathrm{S}] > 0$ and the firing of each $\mathbf{m}$-process-enabled transition is prevented by a set of resource places belonging to $D$.*

*Proof.* $\Longleftarrow$) Each $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled and $\mathbf{m}[P_\mathrm{S}] > \mathbf{0}$. Hence $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live.

$\Longrightarrow$) Let $\mathbf{m}$ be a marking such that the set of $\mathbf{m}$-process-enabled transitions is non-empty and each $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled. Let $D$ be constructed, with $D = D_\mathrm{R} \cup D_\mathrm{S}$, as follows: (i) $D_\mathrm{R} = D \cap P_\mathrm{R} = \{r \in P_\mathrm{R} \mid \exists t \in r^\bullet : \mathbf{m}[r] < \mathbf{Pre}[r,t] \wedge \mathbf{m}[{}^\bullet t \cap P_\mathrm{S}] > \mathbf{0}\}$, and (ii) $D_\mathrm{S} = D \cap P_\mathrm{S} = \{p \in \mathcal{H}_{D_\mathrm{R}} \mid \mathbf{m}[p] = 0\}$.

It will be proved that $D_\mathrm{S} \neq \emptyset$ and $D_\mathrm{S} \subset \mathcal{H}_{D_\mathrm{R}}$. Let us suppose that $D_\mathrm{S} = \emptyset$. Let $F$ be a directed path defined as $F = p_0\, t_0\, p_1\, t_1 \ldots p_k\, t_k$ such that $\forall i \in \{1, \ldots, k\} : p_i \in {}^\bullet t_i \cap P_\mathrm{S}$, $p_0 \in {}^\bullet t_0 \cap P_0$ and $\exists j \in \{1, \ldots, k\} : t_j \cap {}^\bullet D_\mathrm{R} \neq \emptyset$. Such a path must exist since the process nets are strongly connected state machines: thus, for every $i \in I_\mathcal{N}$, $t_j \in T_i$, exists a circuit containing $p_{0_i}$ and $t_j$ such that $p_{0_i}$ and $t_j$ appear only once.

Let $t$ be the last transition in the directed path $F$ such that $t \in {}^\bullet D_\mathrm{R}$. Let $r \in t^\bullet \cap D_\mathrm{R}$. Since $P_\mathrm{S} \cap {}^\bullet t \in \mathcal{H}_r$ and $D_\mathrm{S} = \emptyset$, $\mathbf{m}[P_\mathrm{S} \cap {}^\bullet t] > 0$, i.e. $t$ is $\mathbf{m}$-process-enabled. Since $t \notin D_\mathrm{R}{}^\bullet$ (if $t \in D_\mathrm{R}{}^\bullet$, and taking into account that the net is self-loop free and $P_0 \cap \bigcup_{r \in P_\mathrm{R}} \|\mathbf{y_r}\| = \emptyset$, $t$ could not be the last one), then $t$ is also $\mathbf{m}$-resource-enabled and therefore $t$ can fire contradicting the hypothesis that from $\mathbf{m}$ only transitions in $P_0{}^\bullet$ can occur.

If $D_\mathrm{S} = \mathcal{H}_{D_\mathrm{R}}$, since $\mathbf{m}[D_\mathrm{S}] = \mathbf{0}$, $\forall r \in D_\mathrm{R} : \mathbf{m}[r] = \mathbf{m_0}[r]$ which makes impossible for $r$ to prevent the firing of any transition ($\mathbf{m_0}$ is acceptable). Then, $D_\mathrm{S} \subset \mathcal{H}_{D_\mathrm{R}}$.

Let us now prove that $D = D_\mathrm{R} \cup D_\mathrm{S}$ is a siphon. Let $t \in {}^\bullet D$; two cases must be checked.

***First case*** *($t \in {}^\bullet D_\mathrm{R}$).* Let $r \in D_\mathrm{R}$ be such that $t \in {}^\bullet r$. Let $p \in \mathcal{H}_r \cap {}^\bullet t$ (there exists such $p$ because there is an arc from $t$ to $r$). If $\mathbf{m}[p] = 0$, then $p \in D_\mathrm{S}$, and $t \in D_\mathrm{S}{}^\bullet$. Otherwise, since $t$ is disabled, $\exists r' \in (P_\mathrm{R} \cap {}^\bullet t) : \mathbf{m}[r'] < \mathbf{Pre}[r', t]$, i.e. disabling it. Then $r' \in D_\mathrm{R}$, and in consequence, $t \in D_\mathrm{R}{}^\bullet$.

***Second case*** *($t \notin {}^\bullet D_\mathrm{R}$).* Then $\exists p \in D_\mathrm{S} : t \in {}^\bullet p$ and $\exists r' \in D_\mathrm{R} : p \in \mathcal{H}_{r'}$. If $\exists r \in ({}^\bullet t \cap D_\mathrm{R})$, $t \in D^\bullet$ and we can conclude. Let us now suppose that

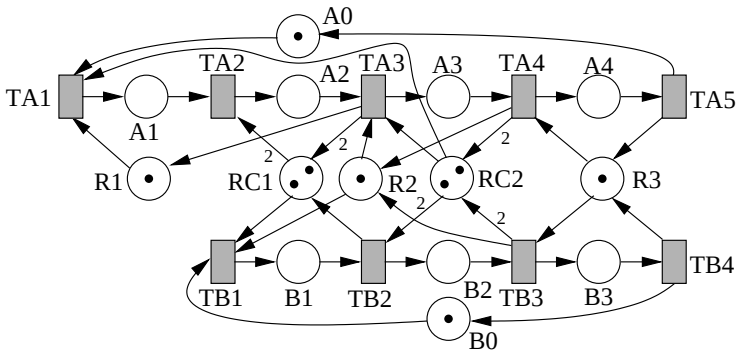



**Figure 4.3**: Net belonging to the controlled Gadara class with no minimal siphon becoming insufficiently marked

$^\bullet t \cap D_{\mathrm{R}} = \emptyset$. In this case, $t$ cannot be $\mathbf{m}$-process-enabled; if it was, by Theorem 4.13, $t$ has to be $\mathbf{m}$-resource-disabled, and then, there would exist $r \in {}^\bullet t \cap D_{\mathrm{R}}$. Let $\{q\} = {}^\bullet t \cap P_{\mathrm{S}}$ (this place exists because $p \in \mathcal{H}_{r'}$ and $^\bullet t \cap D_{\mathrm{R}} = \emptyset$). Since $t$ is not $\mathbf{m}$-process-enabled, $\mathbf{m}[q] = 0$. Moreover, since $p \in \mathcal{H}_{r'}$, $p$ belongs to a minimal p-semiflow containing $r'$ in its support and since $r' \notin {}^\bullet t$, $q$ is also in the support of such p-semiflow, which implies that $q \in \mathcal{H}_{r'}$. Therefore, $q \in D_{\mathrm{S}}$ ($q$ is not marked), and $t \in D_{\mathrm{S}}{}^\bullet$.

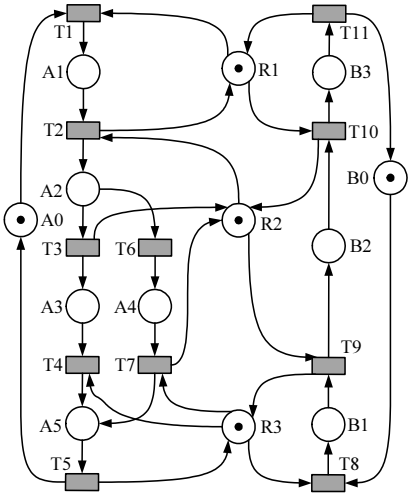
By construction, the firing of each $\mathbf{m}$-process-enabled transition is prevented by some resource places in $D$.

□

A siphon that holds the condition of Theorem 4.14 is said to be a *bad siphon* that becomes *insufficiently marked at* $\mathbf{m}$. Note that minimal siphons are insufficient to characterise non-liveness for controlled Gadara nets. The net in Fig. 4.3 is non-live: the siphon $D = \{\mathrm{RC1}, \mathrm{RC2}, \mathrm{A3}, \mathrm{B2}\}$ becomes insufficiently marked at $\mathbf{m} \equiv [\mathrm{A1}, \mathrm{B1}, \mathrm{RC1}, \mathrm{RC2}, \mathrm{R3}]$, but it is not minimal, since it contains the minimal siphon $D' = \{\mathrm{RC2}, \mathrm{A3}, \mathrm{B2}\}$. $D'$ is not insufficiently marked for any reachable marking. It is worth noting that no siphon, be it minimal or not, is ever fully emptied.

The non-live net system of Fig. 4.4 is an example where an iterative method of synthesis such as those developed by the proponents of Gadara nets [Wan09, LSW+11] cannot be used, because the resulting net falls out of the class after the first iterations.

Figure 4.5 depicts the result of controlling the original net in Fig. 4.4 using an iterative synthesis method. It should be remarked that, if a classical siphon control-based method is to be used, then an iterative procedure is necessary, as the example evidences: there exist markings which inevitably lead to deadlock but no siphon is insufficiently marked until the terminal deadlocks $[\mathrm{A1}, \mathrm{B2}, \mathrm{R3}]$ and $[\mathrm{A5}, \mathrm{B1}, \mathrm{R1}]$ are




*Min. p-semiflows covering a resource place:*

$$\mathbf{y_{R1}} : \mathbf{m}[\mathrm{R1}] + \mathbf{m}[\mathrm{A1}] + \mathbf{m}[\mathrm{B3}] = 1$$
$$\mathbf{y_{R2}} : \mathbf{m}[\mathrm{R2}] + \mathbf{m}[\mathrm{A2}] + \mathbf{m}[\mathrm{A5}] + \mathbf{m}[\mathrm{B2}] = 1$$
$$\mathbf{y_{R3}} : \mathbf{m}[\mathrm{R3}] + \mathbf{m}[\mathrm{A4}] + \mathbf{m}[\mathrm{B1}] = 1$$

*Minimal siphons with some resource places:*

$$D_{\mathrm{R1}} = \{\mathrm{R1}, \mathrm{A1}, \mathrm{B3}\}$$
$$D_{\mathrm{R2}} = \{\mathrm{R2}, \mathrm{A2}, \mathrm{A5}, \mathrm{B2}\}$$
$$D_{\mathrm{R3}} = \{\mathrm{R3}, \mathrm{A4}, \mathrm{B1}\}$$
$$D_{\mathrm{R1R2}} = \{\mathrm{R1}, \mathrm{R2}, \mathrm{A2}, \mathrm{A5}, \mathrm{B3}\}$$
$$D_{\mathrm{R2R3}} = \{\mathrm{R2}, \mathrm{R3}, \mathrm{A2}, \mathrm{A4}, \mathrm{B2}\}$$
$$D_{\mathrm{R1R2R3}} = \{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{A2}, \mathrm{A4}, \mathrm{B3}\}$$

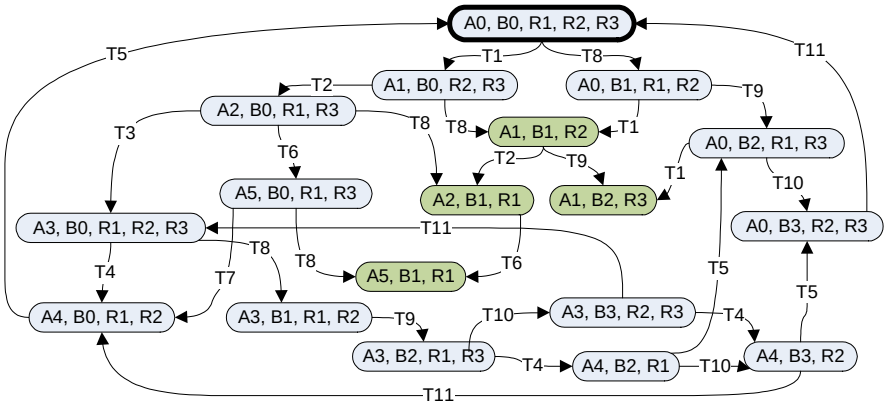
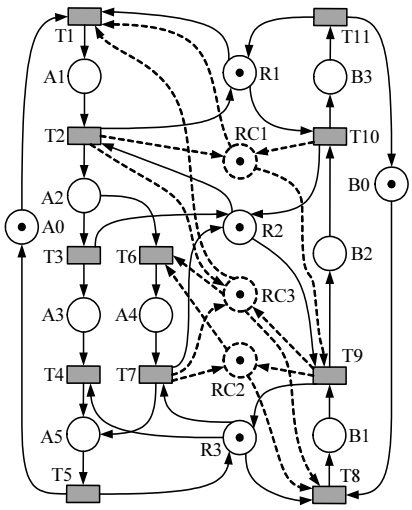



**Figure 4.4**: Gadara net for which some non-minimal siphon must be eventually controlled

removed. Unfortunately, the net obtained after adding control places RC1 and RC2 in the two first iterations is not a Gadara net. Indeed, it does not even belong to the class of controlled Gadara nets, since the controlled place RC2 inputs in the transition T6, which belongs to a process-induced conflict containing two or more transitions. This violates point 4 of the definition of both Gadara and controlled Gadara nets. Even further, the net obtained has a non-minimal siphon ($\{\mathrm{RC1}, \mathrm{RC2}, \mathrm{A1}, \mathrm{A5}, \mathrm{B2}\}$) which does not contain any resource place of the original Gadara net, but only control places instead. And this siphon must be controlled in order to obtain a live system.

The example points out that it is not possible, in the general case, to apply an




*New minimal p-semiflows:*

$$\mathbf{y_{RC1}} : \mathbf{m}[\text{RC1}] + \mathbf{m}[\text{A1}] + \mathbf{m}[\text{B2}] = 1$$
$$\mathbf{y_{RC2}} : \mathbf{m}[\text{RC2}] + \mathbf{m}[\text{A5}] + \mathbf{m}[\text{B1}] = 1$$
$$\mathbf{y_{RC3}} : \mathbf{m}[\text{RC3}] + \mathbf{m}[\text{A1}] + \mathbf{m}[\text{A5}] + \mathbf{m}[\text{B1}] = 1$$

*New minimal siphons:*

None.

*Bad siphons to be controlled:*

Iteration 1: $\{\text{R1}, \text{R2}, \text{A2}, \text{A5}, \text{B3}\} \longrightarrow \text{RC1}$
Iteration 2: $\{\text{R2}, \text{R3}, \text{A2}, \text{A4}, \text{B2}\} \longrightarrow \text{RC2}$
Iteration 3: $\{\text{RC1}, \text{RC2}, \text{A1}, \text{A5}, \text{B2}\} \longrightarrow \text{RC3}$

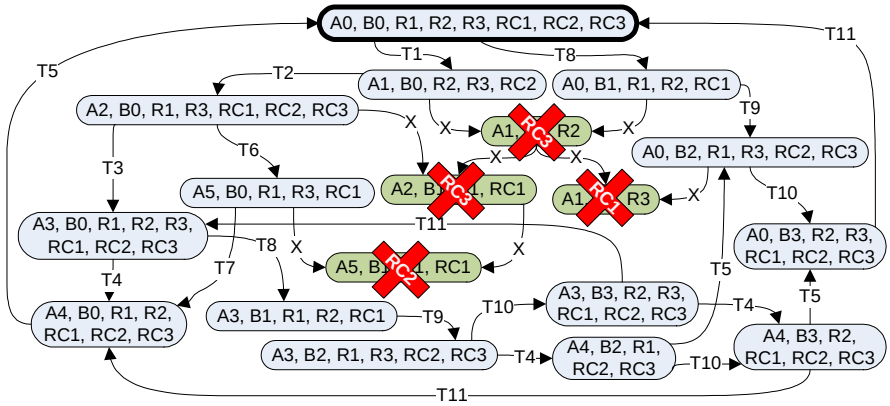



**Figure 4.5**: Controlled Gadara net corresponding to the net in Fig. 4.4

iterative method to cut bad markings maintaining the resulting controlled net at the end of an iteration not only within the Gadara net class, but not even within the controlled Gadara net class. The authors [Wan09] claim that it is still possible to keep it within the class if the designer moves the arcs to previous transitions to those of the conflict set where the new virtual resources input. Nevertheless, the authors do not prove three fundamental aspects related to this post-processing procedure in the synthesis of live models:

1. Is it always possible to do that, and the resulting net remains live?

2. When should be the post-processing performed: after each iteration, at the end of the full process, etc.?

3. The permisiveness of the method must be analysed because, a priori, there always exists a solution that consists in the full sequentialisation of the system in which only a t-semiflow is active at a given moment. In other words, if we start the execution of a t-semiflow of the net, no other t-semiflow will be activated until the initiated is finished.

In other words, it must be proven that the proposed synthesis method for Gadara nets is closed to the class. Observe that this closure property holds for the $S^4PR$ class [Tri03].

## 4.4 Approaching Gadara by means of $S^4PR$ nets

Gadara nets can be transformed into CPR nets (a restricted subclass of $S^4PR$) so that controlling a Gadara net through net state equation-based structural methods [TGVCE05] can alternatively be conducted in the space of the transformed net: as it will be proved onwards, both classes are equivalent at that level.

Paradoxically, the syntactic restriction enforced to retain a structural characterisation places Gadara nets into an instrumental role from the angle of structural liveness analysis and synthesis: the maturity of the techniques introduced for nets [PR01, TGVCE05] suggests working in the transformed space.

First, the subclass of CPR nets will be introduced.

### 4.4.1 A constrained subclass of $S^4PR$ with deterministic processes

**Definition 4.15.** *Let $I_{\mathcal{N}}$ be a finite set of indices. A net of Confluent Processes with Resources (CPR net) is a connected* generalised *pure P/T net $\mathcal{N} = \langle P, T, F, W\rangle$ (or, equivalently, $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$) defined with the same conditions of Definition 4.1 except conditions 4 and 5, which are redefined as follows:*

*4. For all $p \in P_{\mathrm{S}} : |p^{\bullet}| = 1$.*

*5. For each $r \in P_{\mathrm{R}}$, there exists a unique minimal p-semiflow $\mathbf{y}_{\boldsymbol{r}} \in \mathbb{N}^{|P|}$ such that $\{r\} = \|\mathbf{y}_{\boldsymbol{r}}\| \cap P_{\mathrm{R}}$, $\|\mathbf{y}_{\boldsymbol{r}}\| \cap P_0 = \emptyset$, $\|\mathbf{y}_{\boldsymbol{r}}\| \cap P_{\mathrm{S}} \neq \emptyset$ and $\mathbf{y}_{\boldsymbol{r}}[r] = 1$.*

Clearly, CPR nets are a subclass of e-Gadara nets. The corresponding definition of acceptable initial marking is consistent with Definition 4.6 (indeed the conditions are identical) and can be easily derived.

Also, a CPR net is an $S^4PR$ net such that there is no conflict induced by a process place, i.e. $\forall p \in P_S : |p^\bullet| = 1$. Again, it must be noticed that the concept of acceptable initial marking for CPR nets is consistent with that provided for the superclass $S^4PR$ [TGVCE05].

In the same vein, the rest of definitions are inherited from the e-Gadara superclass. In all cases, these definitions collapse perfectly with those given for $S^4PR$ nets.

### 4.4.2 The conflict expansion rule

Next, a rule to transform Gadara nets into CPR nets will be introduced. The free choice constraint in the process subnets of Gadara nets makes that, from the point of view of the allocation of resources, a process first decides the computation path and, after that, the allocation of resources is deterministic. In other words, resources do not participate in the internal choices of the processes. Choices on resources only happen in the competition relations between processes for the resources.

Taking advantage of this behaviour, a transformation for Gadara nets is introduced such that this *a priori* decision about the internal computation path to be carried out when choices appear is dealt from the initial state.

**Definition 4.16.** *Let $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$, $P = P_0 \cup P_S \cup P_R$, be an e-Gadara net such that $\exists p \in P_S : |p^\bullet| > 1$. Let $p_{0_i}$ be the idle place of the process subnet to which $p$ belongs. The net $\mathcal{N}_e = \langle P, T_e \cup \{t\}, \mathbf{C_e}\rangle$ is said to be a* conflict expansion of $p$ in $\mathcal{N}$*, where:*

- $T_e = T \setminus (p^\bullet \cap {}^\bullet P_0)$.
- $\forall t' \in T \setminus p^\bullet : \mathbf{C_e}[P, t'] = \mathbf{C}[P, t']$.
- $\forall t' \in p^\bullet \setminus {}^\bullet P_0, p' \in P \setminus (P_R \cup \{p, p_{0_i}\}), r \in P_R :$
  $\mathbf{C_e}[p', t'] = \mathbf{C}[p', t']$ *(thus: $\mathbf{C_e}[p', t'] \geq 0$),*
  $\mathbf{C_e}[r, t'] = \mathbf{C}[r, t'] - \mathbf{y_r}[p]$ *(thus: $\mathbf{C_e}[r, t'] \leq 0$),*
  $\mathbf{C_e}[p, t'] = 0$, $\mathbf{C_e}[p_{0_i}, t'] = -1$.
- $\forall p' \in P \setminus (P_R \cup \{p, p_{0_i}\}), r \in P_R :$
  $\mathbf{C_e}[p', t] = 0$,
  $\mathbf{C_e}[r, t] = \mathbf{y_r}[p]$ *(thus: $\mathbf{C_e}[r, t] \geq 0$),*
  $\mathbf{C_e}[p_{0_i}, t] = -\mathbf{C_e}[p, t] = 1$.

Figure 4.6 attempts to illustrate the conflict expansion described in Definition 4.16. Note that, in this case, $T_e = \{T1, T2, T3, T4, T5, T6, T7\} = T$ since no conflicting transition outputs to the idle place $p_{0_i}$. Besides, two interesting corollaries are derived from the definition:

**Corollary 4.17.** *$\mathcal{N}_e$ is an e-Gadara net and for every $r \in P_R$ its associated minimal p-semiflow $\mathbf{y_r^e}$ holds $\mathbf{y_r^e} = \mathbf{y_r}$.*

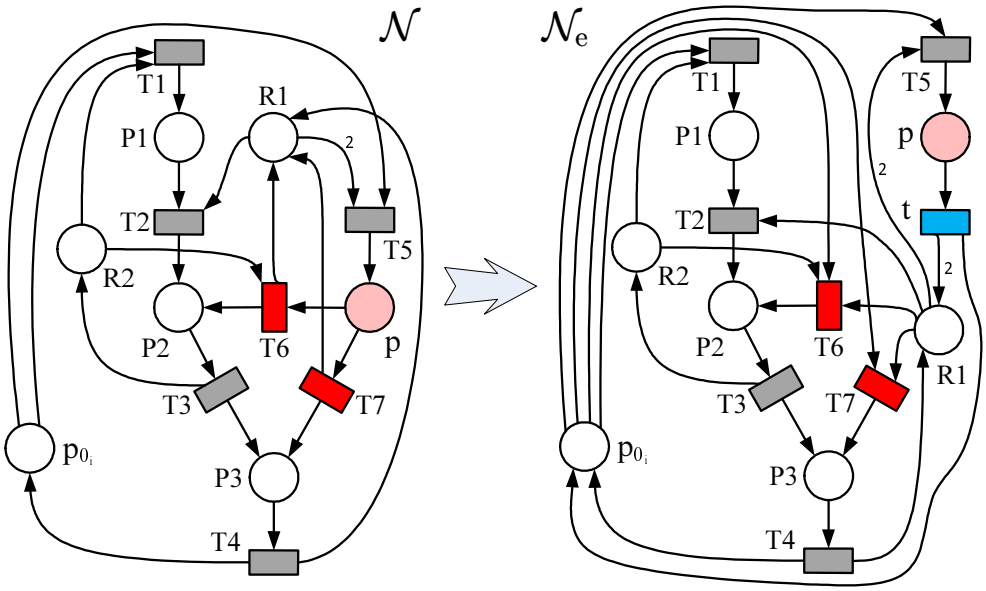



**Figure 4.6**: Example of a conflict expansion in a Gadara net

**Corollary 4.18.** *If there exist no more conflicts in the process subnets of $\mathcal{N}_e$, then $\mathcal{N}_e$ is a CPR net.*

The proof of Corollary 4.17 (which is intuitive but cumbersome to prove) is left to the reader. Corollary 4.18 is straightforward. A consequence of these corollaries is that, starting from an e-Gadara net, a CPR net can always be obtained by way of successively expanding its conflicts.

### 4.4.3 On liveness and siphons preservation

Next, an interesting result regarding (non-)liveness preservation after the net transformation is proved. Since the set of places of $\mathcal{N}$ is equal to the set of places of $\mathcal{N}_e$, markings over $\mathcal{N}$ will be trivially mapped over $\mathcal{N}_e$, and viceversa. This will be assumed for the rest of the chapter, and transitively extended to nets obtained by way of a succession of conflict expansions starting from $\mathcal{N}$.

**Theorem 4.19.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be an e-Gadara net with an acceptable initial marking such that $\exists p \in P_S : |p^\bullet| > 1$, and $\mathcal{N}_e = \langle P_0 \cup P_S \cup P_R, T_e \cup \{t\}, \mathbf{C_e} \rangle$ be an e-Gadara net being the conflict expansion of $p$ in $\mathcal{N}$. $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non-live $\implies$ $\langle \mathcal{N}_e, \mathbf{m_0} \rangle$ is non-live.*

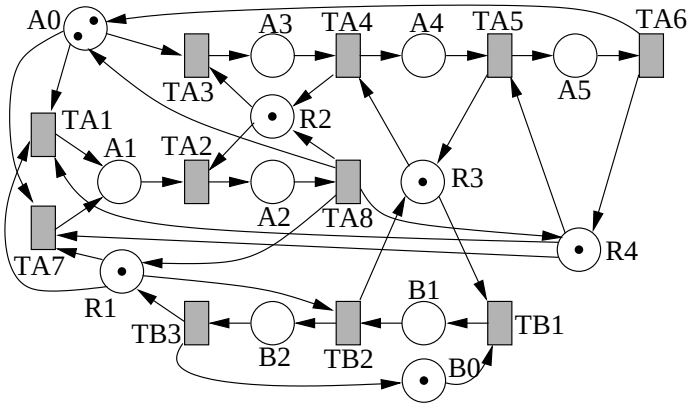



**Figure 4.7**: A CPR net which is the conflict expansion of place A2 in the net of Fig. 4.2

*Proof.* Let $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that $\mathbf{m}[P_\mathrm{S}] > 0$ and every $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled. Such $\mathbf{m}$ must exist by Theorem 4.13. Let $\sigma$ be a firing sequence of $\mathcal{N}$ such that $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$. Let $T' = \{t' \in T \mid \mathbf{C}[p, t'] < 0\}$. A firing sequence $\sigma_\mathrm{e}$ of $\mathcal{N}_\mathrm{e}$ such that $\mathbf{m_0} \xrightarrow{\sigma_\mathrm{e}} \mathbf{m}$ can be constructed by copying $\sigma$ after making some replacements in it, following these two rules: (i) For each occurrence of a transition $u \in T' \setminus T_\mathrm{e}$ in $\sigma$, $u$ is replaced with $t$ in $\sigma'$, and (ii) For each occurrence of a transition $v \in T' \cap T_\mathrm{e}$ in $\sigma$, $v$ is replaced with the sequence $tv$ in $\sigma'$.

The sequence $\sigma'$ must also be firable from $\mathbf{m_0}$, since $\mathbf{C}[P, T \setminus T'] = \mathbf{C_e}[P, T \setminus T']$, and (i) $\forall u \in T' \setminus T_\mathrm{e} : \mathbf{C_e}[P, t] = \mathbf{C}[P, u]$, and (ii) $\forall v \in T' \cap T_\mathrm{e} : \mathbf{C_e}[P, t] + \mathbf{C_e}[P, v] = \mathbf{C}[P, v]$ and $t$ must be firable in $\mathcal{N}_\mathrm{e}$ whenever $v$ is firable in $\mathcal{N}$, since $t$ has the same input process place than $v$ and no input resource place. Thus, $\mathbf{m_0} \xrightarrow{\sigma_\mathrm{e}} \mathbf{m}$.

Finally let $T_\mathrm{mpe}$ be the non-empty set of $\mathbf{m}$-process-enabled transitions of $\mathcal{N}$. For every $u \in T_\mathrm{mpe}$, $u$ is the unique output transition of its input process place in $\mathcal{N}$. Otherwise, $\mathbf{C}[P_\mathrm{R}, u] = \mathbf{0}$ and therefore $u$ would not be $\mathbf{m}$-resource-disabled. Thus, $p$ is not the input place of $u$, $\forall u \in T_\mathrm{mpe}$, and therefore $T_\mathrm{mpe} \cap T' = \emptyset$. Then $\mathbf{C_e}[P, T_\mathrm{mpe}] = \mathbf{C}[P, T_\mathrm{mpe}]$. Thus, $T_\mathrm{mpe}$ is also the set of $\mathbf{m}$-process-enabled transitions of $\mathcal{N}_\mathrm{e}$, and every transition in $T_\mathrm{mpe}$ is $\mathbf{m}$-resource-disabled over $\mathcal{N}_\mathrm{e}$. By Theorem 4.13, $\langle \mathcal{N}_\mathrm{e}, \mathbf{m_0} \rangle$ is non-live. □

The net in Fig. 4.7 is the result of expanding the conflict induced by place A2 in the net of Fig. 4.2. In fact, the net system $\langle \mathcal{N}, \mathbf{m_0} \rangle$ depicted in Fig. 4.2 is a non-live e-Gadara net with an acceptable initial marking. By Theorem 4.19 if the original net system is non-live, the transformed one $\langle \mathcal{N}_\mathrm{e}, \mathbf{m_0} \rangle$ is also non-live. In effect, by sequentially firing transitions TA1, TA3 and TB1 in the net of Fig. 4.7, the marking [A1, A3, B1] is reached, which is a deadlock.

The reverse of Theorem 4.19 is not true in general, since there may exist killing

spurious solutions in a live system $\langle \mathcal{N}, \mathbf{m_0} \rangle$ which are reachable deadlocks in $\langle \mathcal{N}_e, \mathbf{m_0} \rangle$. Nevertheless, Theorem 4.19 allows us to work over the transformed net in order to enforce liveness, since if $\langle \mathcal{N}_e, \mathbf{m_0} \rangle$ is live then $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is live. However, this is only reasonable if the number of siphons to be controlled is not severely increased. The next result is related to this:

**Lemma 4.20.** *Let $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$, $P = P_0 \cup P_S \cup P_R$, be an e-Gadara net such that $\exists p \in P_S : |p^\bullet| > 1$, $\mathcal{N}_e = \langle P, T_e \cup \{t\}, \mathbf{C_e} \rangle$ be a conflict expansion of $p$ in $\mathcal{N}$, and $D \subseteq P$. If $D$ is a siphon of $\mathcal{N}_e$ then $D$ is a siphon of $\mathcal{N}$.*

*Proof.* Let $T' = \{t' \in T \mid \mathbf{C}[p, t'] < 0\}$ be the set of transitions belonging to the conflict induced by $p$ in $\mathcal{N}$ and Prop1 be a parameterised proposition that returns a truth value for each transition of $\mathcal{N}$, where $\text{Prop1}(t_1) \equiv [(\exists p_1 \in D : \mathbf{C}[p_1, t_1] > 0) \implies (\exists p_2 \in D : \mathbf{C}[p_2, t_1] < 0)]$, for every $t_1 \in T$. It is obvious that $D$ is a siphon of $\mathcal{N}$ only if $\forall t_1 \in T : \text{Prop1}(t_1)$. Since $\forall t_3 \in T \setminus T' : \mathbf{C}[P, t_3] = \mathbf{C_e}[P, t_3]$, it is enough to prove that $\forall t_1 \in T' : \text{Prop1}(t_1)$.

Let us prove that $\forall t_1 \in T' \setminus T_e : \text{Prop1}(t_1)$. Remind that the set of transitions in $T' \setminus T_e$ are those transitions belonging to the conflict induced by $p$ in $\mathcal{N}$ which output to the idle place of its process subnet in $\mathcal{N}$ and therefore are removed from the transformed net. Without loss of generality, let $t_1 \in T' \setminus T_e$. If $\nexists p_1 \in D$ such that $\mathbf{C}[p_1, t_1] > 0$, then $\text{Prop1}(t_1) \equiv \text{True}$. Let $p_1 \in D$ such that $\mathbf{C}[p_1, t_1] > 0$. Note that $\forall r \in P_R : \mathbf{C}[r, t_1] = \mathbf{y_r}[p]$, $\mathbf{C}[p_{0_i}, t_1] = 1$, $\mathbf{C}[p, t_1] = -1$, and $\forall p_2 \in P \setminus (P_R \cup \{p_{0_i}, p\}) : \mathbf{C}[p_2, t_1] = 0$. Thus, $\mathbf{C_e}[P, t] = \mathbf{C}[P, t_1]$. Then $\mathbf{C_e}[p_1, t] = \mathbf{C}[p_1, t_1] > 0$. Since $p_1 \in D$ and $p$ is the unique input place of $t$, then $p \in D$. Since $\mathbf{C}[p, t_1] < 0$, $\text{Prop1}(t_1) \equiv \text{True}$.

It will now be proved that $\forall t_1 \in T' \cap T_e : \text{Prop1}(t_1)$, i.e., we pay attention to the transitions belonging to the conflict induced by $p$ in $\mathcal{N}$ which remain in the transformed net. Without loss of generality, let $t_1 \in T' \cap T_e$. Two cases are observed:

**(a)** Suppose that $\nexists p_1 \in D$ such that $\mathbf{C_e}[p_1, t_1] > 0$. Since the unique output place of $t_1$ in $\mathcal{N}_e$ is a process place, if $\mathbf{C}[P_R, t_1] = \mathbf{0}$, then $\nexists p_2 \in D$ such that $\mathbf{C}[p_2, t_1] > 0$, and $\text{Prop1}(t_1) \equiv \text{True}$. If $\mathbf{C}[P_R, t_1] \not\geq \mathbf{0}$, let $r$ be an arbitrary $r \in P_R$ such that $\mathbf{C}[r, t_1] > 0$. Then $\mathbf{y_r}[p] > 0$ and therefore $\mathbf{C_e}[r, t] = \mathbf{y_r}[p] > 0$. Since $p$ is the unique input of $t$, then $p \in D$. Since $\mathbf{C}[p, t_1] < 0$, $\text{Prop1}(t_1) \equiv \text{True}$.

**(b)** Otherwise, $\exists p_1 \in D$ such that $\mathbf{C_e}[p_1, t_1] > 0$. Then $\mathbf{C}[p_1, t_1] \geq \mathbf{C_e}[p_1, t_1] > 0$ and $\exists p_2 \in D$ such that $\mathbf{C_e}[p_2, t_1] < 0$. If $p_2 \in P_0$ then $p \in D$, since $p$ is the unique input place of $t$ and $t$ is an input transition of $P_0$. Since $\mathbf{C}[p, t_1] < 0$, $\text{Prop1}(t_1) \equiv \text{True}$. If $p_2 \notin P_0$, i.e., $p_2 \in P_R$, then $p \in D$, since $\forall r \in P_R : \mathbf{C_e}[r, t] = \mathbf{y_r}[p] \geq -\mathbf{C_e}[r, t_1]$ and $p$ is the unique input transition of $t$. Since $\mathbf{C}[p, t_1] < 0$, $\text{Prop1}(t_1) \equiv \text{True}$.

Since $\forall t_1 \in T : \text{Prop1}(t_1)$, $D$ is a siphon of $\mathcal{N}$. □

**Corollary 4.21.** *The number of siphons of $\mathcal{N}_\mathrm{e}$ is lower than or equal to the number of siphons of $\mathcal{N}$.*

Returning to the example of Fig. 4.6, both nets (original and transformed) have the same set of siphons (note that only the first three are minimal siphons):

$$\begin{aligned}
D_{\mathrm{S}_i} &= \{p_{0_i}, \mathrm{P1}, \mathrm{P2}, \mathrm{P3}, p\} && = \|\mathbf{y_{S_i}}\| \\
D_{\mathrm{R1}} &= \{\mathrm{R1}, \mathrm{P2}, \mathrm{P3}, p\} && = \|\mathbf{y_{R1}}\| \\
D_{\mathrm{R2}} &= \{\mathrm{R2}, \mathrm{P1}, \mathrm{P2}\} && = \|\mathbf{y_{R2}}\| \\
&\{\mathrm{R1}, p_{0_i}, \mathrm{P2}, \mathrm{P3}, p\} && = D_{\mathrm{R1}} \cup \{p_{0_i}\} \\
&\{\mathrm{R1}, p_{0_i}, \mathrm{P1}, \mathrm{P2}, \mathrm{P3}, p\} && = D_{\mathrm{R1}} \cup \{p_{0_i}, \mathrm{P1}\} \\
&\{\mathrm{R2}, p_{0_i}, \mathrm{P1}, \mathrm{P2}, \mathrm{P3}, p\} && = D_{\mathrm{R2}} \cup \{p_{0_i}, \mathrm{P3}, p\} \\
&\{\mathrm{R1}, \mathrm{R2}, p_{0_i}, \mathrm{P2}, \mathrm{P3}, p\} && = D_{\mathrm{R1}} \cup \{\mathrm{R2}, p_{0_i}\} \\
&\{\mathrm{R1}, \mathrm{R2}, p_{0_i}, \mathrm{P1}, \mathrm{P2}, \mathrm{P3}, p\} && = D_{\mathrm{R1}} \cup \{\mathrm{R2}, p_{0_i}, \mathrm{P1}\}
\end{aligned}$$

However, the reverse of Lemma 4.20 is not true (i.e., a siphon of $\mathcal{N}$ is not always a siphon of $\mathcal{N}_\mathrm{e}$). For example, the net in Fig. 4.2 has the (non-minimal) siphon $D = \{\mathrm{R3}, \mathrm{A0}, \mathrm{A4}, \mathrm{A5}, \mathrm{B1}\}$, but this is not a siphon in the transformed net of Fig. 4.7 since $\mathrm{TA8} \in {}^\bullet\mathrm{A0} \subseteq {}^\bullet D$ but $\mathrm{TA8} \notin D^\bullet$.

Nevertheless, there exists a close relation between the siphons of both nets. Indeed, for every siphon in $\mathcal{N}$ there exists another siphon in $\mathcal{N}_\mathrm{e}$ which contains the same resource places. The next proposition is instrumental; the corresponding lemma, in which the aforementioned proposition is used, follows shortly afterwards.

**Proposition 4.22.** *Let $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$, $P = P_0 \cup P_\mathrm{S} \cup P_\mathrm{R}$, be an e-Gadara net such that $\exists p \in P_\mathrm{S} : |p^\bullet| > 1$, $\mathcal{N}_\mathrm{e} = \langle P, T_\mathrm{e} \cup \{t\}, \mathbf{C_e}\rangle$ be a conflict expansion of $p$ in $\mathcal{N}$, and $D \subseteq P$. Let $\mathcal{N}_\mathrm{et} = \langle P, T_\mathrm{e}, \mathbf{C_{et}}\rangle$ be the subnet generated by restricting $\mathcal{N}_\mathrm{e}$ to $\langle P, T_\mathrm{e}\rangle$. If $D$ is a siphon of $\mathcal{N}$ then $D$ is a siphon of $\mathcal{N}_\mathrm{et}$.*

*Proof.* Let $T' = \{t' \in T \mid \mathbf{C}[p, t'] < 0\}$ be the set of transitions belonging to the conflict induced by $p$ in $\mathcal{N}$ and Prop2 be a parameterised proposition that returns a truth value for each transition of $\mathcal{N}_\mathrm{et}$, where $\mathrm{Prop2}(t_2) \equiv [(\exists p_1 \in D : \mathbf{C_e}[p_1, t_2] > 0) \implies (\exists p_2 \in D : \mathbf{C_e}[p_2, t_2] < 0)]$, for every $t_2 \in T_\mathrm{e}$. It is obvious that $D$ is a siphon of $\mathcal{N}_\mathrm{et}$ only if $\forall t_2 \in T_\mathrm{e} : \mathrm{Prop2}(t_2)$. Since $\forall t_3 \in T \setminus T' : \mathbf{C}[P, t_3] = \mathbf{C_e}[P, t_3]$, it is enough to prove that $\forall t_2 \in T' \cap T_\mathrm{e} : \mathrm{Prop2}(t_2)$.

Remind that the set of transitions in $T' \setminus T_\mathrm{e}$ are those transitions belonging to the conflict induced by $p$ in $\mathcal{N}$ which output to the idle place of its process subnet in $\mathcal{N}$ and therefore are removed from the transformed net. Without loss of generality, let $t_2 \in T' \cap T_\mathrm{e}$, and note that $\forall p_1 \in P : \mathbf{C_e}[p_1, t_2] \leq \mathbf{C}[p_1, t_2]$. Suppose that $\nexists p_1 \in D$ such that $\mathbf{C}[p_1, t_2] > 0$. Then $\forall p_1 \in D : \mathbf{C_e}[p_1, t_2] \leq \mathbf{C}[p_1, t_2] \leq 0$, i.e., $\nexists p_2 \in D$ such that

$\mathbf{C_e}[p_2, t_2] > 0$, and Prop2$(t_2) \equiv$ True. Otherwise, $\exists p_2 \in D$ such that $\mathbf{C}[p_2, t_2] < 0$. Then $\mathbf{C_e}[p_2, t_2] \leq \mathbf{C}[p_2, t_2] < 0$, and thus Prop2$(t_2) \equiv$ True. □

Returning to the previously mentioned example, it can be observed that $D = \{\text{R3}, \text{A0}, \text{A4}, \text{A5}, \text{B1}\}$ is a siphon both of the original net in Fig. 4.2 and the transformed net in Fig. 4.7 after removing transition TA8.

**Lemma 4.23.** *Let $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$, $P = P_0 \cup P_\text{S} \cup P_\text{R}$, be an e-Gadara net such that $\exists p \in P_\text{S} : |p^\bullet| > 1$, $\mathcal{N}_\text{e} = \langle P, T_\text{e} \cup \{t\}, \mathbf{C_e}\rangle$ be a conflict expansion of $p$ in $\mathcal{N}$, and $D \subseteq P$. If $D$ is a siphon for $\mathcal{N}$, then $\exists D_\text{e} \supseteq D$, with $D_\text{e} \setminus D \subset P_\text{S}$, such that $D_\text{e}$ is a siphon of $\mathcal{N}_\text{e}$.*

*Proof.* Let $i \in I_\mathcal{N}$ the index of the process subnet of $\mathcal{N}_\text{e}$ to which $p$ belongs. Let $\mathcal{N}_\text{et} = \langle P, T_\text{e}, \mathbf{C_{et}}\rangle$ be the subnet generated by restricting $\mathcal{N}_\text{e}$ to $\langle P, T_\text{e}\rangle$. By Proposition 4.22, $D$ is a siphon of $\mathcal{N}_\text{et}$. However, $D$ is not a siphon of $\mathcal{N}_e$ iff $p \notin D$ and $\exists p_1 \in (\{p_{0_i}\} \cup P_\text{R}) \cap D$ such that $\mathbf{C_e}[p_1, t] > 0$.

If $p_1 = p_{0_i}$ then $D_\text{e} = D \cup P_i$ is a siphon of $\mathcal{N}_\text{e}$, since the $i$-th process subnet is a strongly connected state machine. Otherwise, $r = p_1$ is a resource place, $r \in P_\text{R}$, with $\mathbf{y_r}[p] > 0$. Let $D_\text{t} = (\mathcal{H}_r \setminus D) \cap P_i$. It will proved that $D_\text{e} = D \cup D_\text{t}$ is a siphon of $D$.

Let $T' = \{t \in T \mid \exists p \in D_\text{t}$ such that $\mathbf{C}[p, t] > 0\}$ and Prop2$(t_1) \equiv [(\exists p_1 \in D : \mathbf{C_e}[p_1, t_1] > 0) \implies (\exists p_2 \in D : \mathbf{C_e}[p_2, t_1] < 0)]$. It must be proved that $\forall t_1 \in T :$ Prop2$(t_1)$. Since for every $t_2 \in T \setminus (T' \cup \{t\}) : \mathbf{C_e}[D, t_2] = \mathbf{C_{et}}[D, t_2]$, it is enough to prove that $\forall t_1 \in T' :$ Prop2$(t_1) \wedge$ Prop2$(t)$. Since $p \in \mathcal{H}_r$, then $p \in D$ and Prop2$(t) \equiv$ True.

Finally, it will be proved that $\forall t_1 \in T' :$ Prop2$(t_1)$. Without loss of generality, let $t_1 \in T'$, and let $p_1$ the output process place of $t_1$, $p_1 \in D_\text{t}$. If $\mathbf{C}[r, t] < 0$, then Prop2$(t_1) \equiv$ True since $r \in D$. Otherwise, if $\mathbf{C}[r, t] = 0$ then $\exists p_2 \in P_i$ such that $\mathbf{C}[p_2, t] < 0$. If $p_2 \in D$ then Prop2$(t_1) \equiv$ True. If $p_2 \notin D$ then $p_2 \in \mathcal{H}_r$ (because $\mathbf{C}[r, t] < 0$) and thus $p_2 \in D_\text{t}$. Summing up, in all cases, Prop2$(t_1) \equiv$ True. □

Bringing back our attention to the previous example, we can see that the siphon $D$ of the original net in Fig. 4.2, where $D = \{\text{R3}, \text{A0}, \text{A4}, \text{A5}, \text{B1}\}$, is contained in the siphon $D_\text{e} = \{\text{R3}, \text{A0}, \text{A1}, \text{A2}, \text{A4}, \text{A5}, \text{B1}\}$ of the transformed net in Fig. 4.7, and $D_\text{e} \setminus D = \{\text{A1}, \text{A2}\} \subset P_\text{S}$.

### 4.4.4 Transformation rules between Gadara and CPR nets

Next, the expansion and reduction rules will be introduced, based on Definition 4.16. In order to be able to undo a conflict expansion after having enforced liveness, it is necessary to keep record of the previous steps. The following definition is instrumental for this aim.

**Definition 4.24.** *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$ be an e-Gadara net. Its* Associated Expansion Record (AER) *is a duple $\langle T_\mathcal{N}, \Psi_\mathcal{N}\rangle$, where $T \subseteq T_\mathcal{N}$ and $\Psi_\mathcal{N}$ is a set of triples in $T_\mathcal{N} \times P_S \times \mathcal{P}(T_\mathcal{N})$ such that $\forall (t,p,\tau) \in \Psi_\mathcal{N}$: (i) $\exists p_0 \in P_0 : t \in {}^\bullet p_0, \tau \in p_0{}^\bullet$; (ii) $\{p\} = {}^\bullet t \cap P_S$; and (iii) $\forall (t',p,\tau') \in \Psi_\mathcal{N} : t \neq t', \tau \cap \tau' = \emptyset$.*

$\Psi_\mathcal{N}$ registers which conflicts were previously expanded, and how. $T_\mathcal{N}$ is a record of the whole set of transitions, including those which were removed or created at past conflict expansions. The transformations are formally defined as follows:

**Rule 4.1.** (Expansion Rule)

Input: *An e-Gadara net $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$ such that $\exists p \in P_S : |p^\bullet| > 1$, plus its AER $\langle T_\mathcal{N}, \Psi_\mathcal{N}\rangle$.*

Output: *An e-Gadara net $\mathcal{N}_e = \langle P_0 \cup P_S \cup P_R, T_e \cup \{t\}, \mathbf{C_e}\rangle$ which is the conflict expansion of $p$ in $\mathcal{N}$, plus its AER $\langle T_\mathcal{N} \cup \{t\}, \Psi_{\mathcal{N}_e}\rangle$, with $\Psi_{\mathcal{N}_e} = \Psi_\mathcal{N} \cup \{(t,p,p^\bullet)\}$.*

**Rule 4.2.** (Reduction Rule)

Input: *An e-Gadara net $\mathcal{N}_e = \langle P, T_e \cup \{t\}, \mathbf{C_e}\rangle$ plus its AER $\langle T_\mathcal{N} \cup \{t\}, \Psi_{\mathcal{N}_e}\rangle$, such that there exists $(t,p,\tau) \in \Psi_{\mathcal{N}_e}$ and there also exists an e-Gadara net $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$ with $\mathcal{N}_e$ being the conflict expansion of $p$ in $\mathcal{N}$.*

Output: *The e-Gadara net $\mathcal{N}$ plus its AER $\langle T_\mathcal{N}, \Psi_\mathcal{N}\rangle$, with $\Psi_\mathcal{N} = \Psi_{\mathcal{N}_e} \setminus \{(t,p,\tau)\}$.*

Thanks to Theorem 4.19, liveness can be enforced directly over the transformed CPR net. Once enough control places have been aggregated so as to make it live, the reduction rule can be applied to obtain a live e-Gadara net. However, it is worth mentioning that it may be necessary to move carefully the arcs of some control places before. This is due to the fact that some transitions were uncontrollable in the original net: namely, those belonging to a conflict in a process subnet [WLR+09]. Indeed, any transition belonging to the set $\mathcal{F} = \bigcup\{\tau \subset T_\mathcal{N} \mid \exists p \in P_S, t \in T_\mathcal{N} : (t,p,\tau) \in \Psi_\mathcal{N}\}$ is uncontrollable in the original (i.e., expanded) net. Therefore, the following last rule is introduced in order to accomplish the aforementioned goal:

**Rule 4.3.** (Arc-extension Rule)

Input: *A CPR net $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$ plus its AER $\langle T_\mathcal{N}, \Psi_\mathcal{N}\rangle$ such that $\exists r \in P_R, (t,p,\tau) \in \Psi_\mathcal{N}, t_\tau \in \tau : \mathrm{K} = \mathbf{Pre}[r,t_\tau] - \mathbf{y_r}[p] > 0$.*

Output: *A CPR net $\mathcal{N}_e = \langle P, T, \mathbf{C_e}\rangle$ such that $\forall q \in P \setminus \{r\}, u \in T : \mathbf{C_e}[q,u] = \mathbf{C}[q,u]$, $\forall u' \in T \setminus ({}^\bullet p \cup \{t\}) : \mathbf{C_e}[r,u'] = \mathbf{C}[r,u']$, $\forall u'' \in {}^\bullet p : \mathbf{C_e}[r,u''] = \mathbf{C}[r,u''] - \mathrm{K}$ and $\mathbf{C_e}[r,t] = \mathbf{C}[r,t] + \mathrm{K}$.*

### 4.4.5 Synthesis from the underlying CPR net

Given the above set of transformation rules, it is possible to construct a method for the correction of a non-live e-Gadara net. The correction is performed by adding monitors preventing firing sequences that can empty a bad siphon, and therefore lead

to a deadlock. It is important to note that if the original net is a Gadara net, the resulting net remains within the class of controlled Gadara nets after the addition of the control logic. That is, it is a closed method on the class, which overcomes the main limitation of the techniques of the proponents of Gadara nets [Wan09, LSW+11]; limitations that were discussed at the end of Sect. 4.3.

This technique is divided into three stages. First, it performs the transformation of the e-Gadara net into a CPR net through the successive application of Rule 4.1 to every conflict in the net. Before starting this process, the AER associated to the net is assigned the pair $\langle T, \emptyset \rangle$. At the end of the first stage, the AER contains information related to the expanded conflicts and the set of added and removed transitions. In the example net of Fig. 4.2, only one conflict exists. Therefore, the net obtained at completion of the first stage of the method is the one illustrated in Fig. 4.7. The AER of the transformed net $\mathcal{N}_e$ shall be as follows:

$$\langle \{\text{TA1} - \text{TA8}, \text{TB1} - \text{TB3}\}, \{(\text{TA8}, \text{A2}, \{\text{TA3}, \text{TA7}\})\} \rangle$$

Once the transformed net is obtained, the second step is to make it live. Since it is a CPR net, and thus belongs to a subclass of $S^4PR$, we may use some of the iterative techniques of adding monitors described for $S^4PR$ nets so as to correct the transformed net. It is convenient to mention that the addition of monitor places is closed within the CPR subclass since it does not alter the structure of the processes.

At this point, it is worth stressing those ILP-based iterative synthesis techniques [Tri03, TGVCE05] which were revisited in Chap. 3 as efficient approaches to the control of $S^4PR$ nets. The next result regarding the potential reachability set (PRS) of the transformed net is especially relevant with regard to the application of such techniques in the second stage of our method.

**Lemma 4.25.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an e-Gadara net with an acceptable initial marking such that $\exists p \in P_S : |p^\bullet| > 1$, and $\langle \mathcal{N}_e, \mathbf{m_0} \rangle$ be an e-Gadara net obtained by applying the conflict expansion transformation rule over $\mathcal{N}$. Then $\mathbf{m} \in \text{PRS}(\mathcal{N}, \mathbf{m_0})$ iff $\mathbf{m} \in \text{PRS}(\mathcal{N}_e, \mathbf{m_0})$.*

*Proof.* For every $i \in I_\mathcal{N}$, let $\mathbf{y_{S_i}}$ denote the unique minimal p-semiflow of $\mathcal{N}$ induced by the $i$-th process subnet of $\mathcal{N}$. It is easy to see that $\mathbf{y_{S_i}}$ is also a unique minimal p-semiflow of $\mathcal{N}_e$, induced by the $i$-th process subnet of $\mathcal{N}_e$. On the other hand, by Corollary 4.17, $\mathbf{y_r} = \mathbf{y_r^e}$, for all $r \in P_R$.

Let $\mathbf{B}$ be a matrix of dimensions $(|P_R| + |I_\mathcal{N}|) \times |P|$ of integers such that the rows of $\mathbf{B}$ are the set of vectors $\{\mathbf{y_{S_i}} \mid i \in I_\mathcal{N}\} \cup \{\mathbf{y_r} \mid r \in P_R\}$. Then $\mathbf{B}$ is a non-negative canonical basis of p-semiflows both for $\mathcal{N}$ and $\mathcal{N}_e$.

Finally, since a Gadara net is consistent (by Lemma 4.9) and conservative (by construction), a non-negative canonical basis of p-semiflows ($\mathbf{B}$) generates the same solution space than the net state equation. Hence, $\text{PRS}(\mathcal{N}, \mathbf{m_0}) = \text{PRS}(\mathcal{N}_e, \mathbf{m_0})$. □

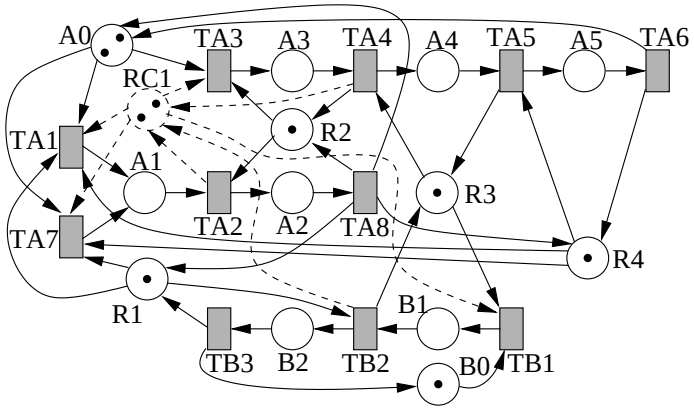



**Figure 4.8**: The CPR net obtained after controlling siphon $D'$ of Fig. 4.7

Many efficient structure-based liveness enforcing techniques rely on the net state equation. Section 3.2.2 reviews one such technique for $S^4PR$ nets [TGVCE05]; note that $S^4PR$ is a superclass of CPR. The result of Lemma 4.25 encourages the application of this kind of techniques over the transformed net, since the space solution of the net state equation is equal on both the original and the transformed net.

Figure 4.7 shows the result of adding a control place to the net of Fig. 4.8 using the aforementioned technique. This place prevents the minimal siphon $D' = \{\mathrm{R1}, \mathrm{R2}, \mathrm{R3}, \mathrm{A2}, \mathrm{A4}, \mathrm{B2}\}$ from becoming empty by firing TA1, TA3 and TB1, and therefore also prevents the deadlocked marking $[\mathrm{A1}, \mathrm{A3}, \mathrm{B1}]$ from being reachable.

Note that, as expected from Lemma 4.20, the siphon above is also a siphon of the original net. In this case, moreover, the deadlocked marking $[\mathrm{A1}, \mathrm{A3}, \mathrm{B1}]$ is also reachable in the original net by firing, for example, the sequence TA1 TA2 TA3 TA1 TB1 from the initial marking.

After adding the control places needed to make the transformed net live, the previous expansion of conflicts recorded by the AER $\langle T_{\mathcal{N}}, \Theta_{\mathcal{N}} \rangle$ must be reversed. In this way, the processes will recover their original structure. Eventually, this is accomplished through repeated application of Rule 4.2. A prior obstacle stems from the fact that the new control places can have outgoing arcs into transitions that originally belonged to a conflict of two or more transitions induced by a process place. Such transitions will belong again to a similar conflict after applying the reduction rule. Recall that these are uncontrollable transitions from the perspective of the implementation techniques developed by the proponents of Gadara nets [WLR+09]. In addition, conflicts cannot have input arcs if the net finally obtained should fall within the class of controlled Gadara nets.

For this reason, each time a new monitor place is added, it must be evaluated

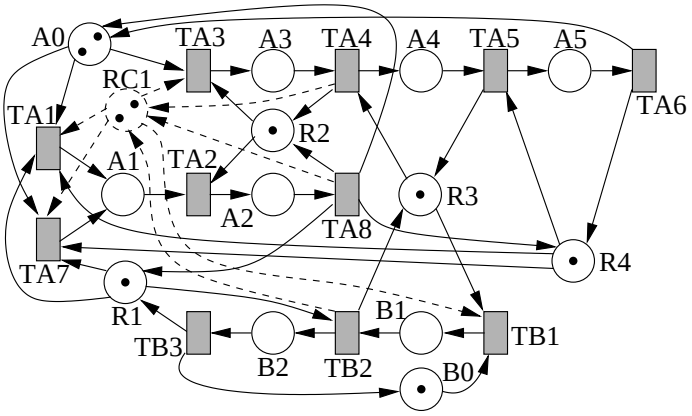



**Figure 4.9**: The CPR net obtained after extending arcs in the net of Fig. 4.8

whether it outputs into a transition belonging to an expanded conflict, i.e., if the control place projects some arc into a transition which appears in the third component of a tuple in $\Theta_{\mathcal{N}}$. For each one of these arcs, Rule 4.3 must be applied. The latter moves those arc to transitions which were preceding them in the original process paths. This rule is applied as many times as necessary until the control place does not output into any of these transitions.

After moving the corresponding arcs, the monitor place cuts a superset of markings which includes those markings which were previously cut. An endpoint is guaranteed for this extension: this is the first transition of the process path (that transition that follows the idle place) because at least that one does not belong to any conflict induced by a process place. Obviously, in cases where it is necessary to extend the arc, which symbolises the acquisition of the virtual resource, into that particular transition, concurrency is reduced to the maximum. In that sense, studying the permissiveness of the method remains as an open line of future work.

The net of Fig. 4.9 illustrates the application of Rule 4.3 on the net of Fig. 4.8. Note that in this last figure, the control place RC1 has output arcs to transitions TA3 and TA7, both belonging to the same expanded conflict. Consequently, whenever Rule 4.2 is applied on the net of Fig. 4.8, these transitions will again belong to the same conflict. The arcs outgoing from RC1 to these transitions would then infringe point 4 of the definition of controlled Gadara net (e-Gadara net), and the resulting net would be out of the class. That is why it is necessary to apply Rule 4.3. Note that in this case those arcs still remain in the resulting net after applying Rule 4.3. However, the new outgoing arc from TA8 to RC1 guarantees the desired property as these arcs will cancel each other when Rule 4.2 is applied. More on this fact later.

After applying the arc extension rule as many times as necessary, new insufficiently

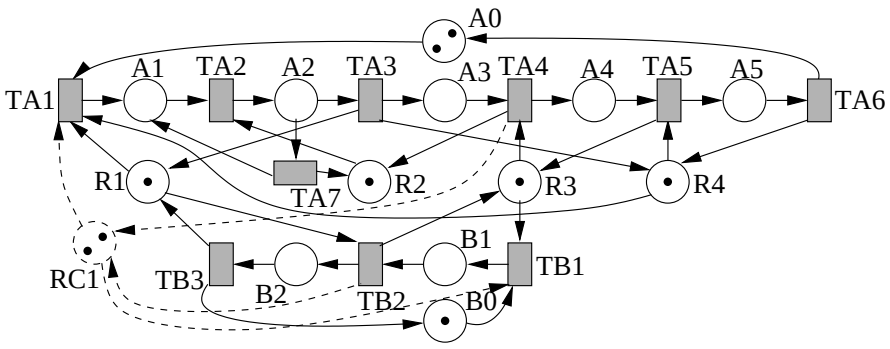



**Figure 4.10**: After applying the last transformation: The corrected Gadara net

marked siphons are seeked and, if found, corrected by adding a new control place. Again, Rule 4.3 is applied on the new controlled net as many times as needed. This process is repeated iteratively until no other siphon is suspected of being insufficiently marked for any reachable marking. In the net of the example, there is no other eligible siphon under the restrictions of the ILPP proposed in the technique previously exposed in Subsection 3.2.2, i.e., the net system on Fig. 4.9 is live.

The third and last stage of the method consists in reducing previously expanded conflicts, so that the processes again have identical structure to those of the original net. This is achieved by applying Rule 4.2 as many times as necessary until the AER of the net returns to its original configuration, i.e., $\langle T, \emptyset \rangle$. This is guaranteed to eventually happen since Rule 4.2 is the inverse of Rule 4.1 and the aggregated monitor places do not appear in $\Theta_{\mathcal{N}}$. Moreover, Theorem 4.19 ensures that if the controlled CPR net is live, the e-Gadara net obtained is also live.

In the end, the resulting net is identical to the original one except for some control places which have been added: these control places are the same the controlled CPR had. Thanks to the application of Rule 4.3, these places have no output arcs into transitions belonging to conflicts induced by process places which contain several transitions. The net on Fig. 4.10 shows the resulting net after applying Rule 4.2 on the net of Fig. 4.9. Indeed, we can observe that transitions TA3 and TA7 have no input arcs from CP1, but the corresponding arc inputs at TA1 instead: a transition that precedes them. The net obtained is a live controlled Gadara net.

## 4.5 Conclusions

In this chapter, an overview of Gadara nets has been presented, along with its limitations for modelling multithreaded control software. From the structural analysis and synthesis perspective, it has been proved that the syntactic restrictions introduced in

Gadara nets provoke significant constraints from the point of view of the behaviours allowed in the allocation of resources. This means Gadara nets can be connected with a subclass of $S^4PR$ in which the allocation of resources internal to a process is deterministic, i.e., resources do not participate in the internal choices. Consequently, liveness enforcing methods based solely on structural information can be used, leaving this class close to the well-studied $S^4PR$ class in that context. Unfortunately, state-space exploration and region theory based methods can be too consuming for real-world concurrent control software systems due to their usually huge dimensions. On the other hand, a more versatile class for modelling multithreaded software systems ($PC^2R$) was introduced in Chap. 2. Nevertheless, new and more complex phenomena arise, in such a way that a structural liveness characterisation still remains elusive.

# Chapter 5

# Some complexity results on the resource allocation problem


## Summary

This chapter takes a different approach to provide an insight into the inherent computational complexity of the RAP, from the perspective of optimality in either prevention, avoidance or detection of deadlocks. In particular, it will be proved that most of the problems involved fall within the category of NP or co-NP-complete problems.

## 5.1 Introduction

As earlier chapters have hopefully made clear, Petri nets are nowadays consolidated as a powerful formalism for the analysis and treatment of deadlocks in RASs. In particular, the methodological framework yielded by the $S^4PR$ class has raised considerable interest on the grounds of a well-balanced compromise between modelling flexibility and the provision of sound and effective correction techniques. Through the process of abstraction, these models and techniques can be effectively applied to diverse application domains, as discussed in Chap. 1, including some simple multithreaded software systems. Most works on that class focus on providing tools and algorithms for dealing with the so-called resource allocation problem, i.e., the emergence of deadlocks. This was precisely the scientific rationale of the previous chapters but approached from the perspective of multithreaded software systems modelled with $PC^2R$ nets.

The strategies for handling deadlocks are traditionally categorised in three large groups: (deadlock) prevention, avoidance and detection. Deadlock prevention techniques consist in constructing a system such that, by definition, no deadlock is reachable. Deadlock avoidance techniques rely on evaluating and deciding on-line whether a resource allocation request is 'safe'. The request is allowed or not depending on the current system state information (e.g., the banker's algorithm [Tri03]). Proceeding that way, deadlocks are avoided. Finally, deadlock detection techniques act 'a posteriori', allowing the deadlock situation to occur and subsequently resolve the situation.

This chapter investigates the computational complexity on providing optimal solutions for the problems of deadlock prevention, avoidance and detection for Sequential RASs supported by the $S^4PR$ class. $S^4PR$ nets were already introduced in Chap. 1. In rough words, they are a subclass of $PC^2R$ nets in which the processes have no internal loops and no resource is allocated in the initial state (i.e., there is no resource lending). Apart for their suitability to many application domains (see Chap. 1), $S^4PR$ nets are enough to model those multithreaded software systems in which: (i) for each process, the acquire/wait and release/signal operations are located at the same iteration level (usually, the main block), and (ii) the resources are used in a first-acquire-then-release basis (i.e., no `signal` call precedes a `wait` call over the same mutex/semaphore).

Some previous works have successfully studied computational issues on Sequential RASs, although they differ from this chapter both on the type of systems and the problems subject to analysis. The problem of deciding whether a resource allocation request is safe in an intermediate step of execution of the system (assuming that it is safe iff there exists a feasible sequence that terminates all processes) is studied by E.M. Gold [Gol78], proving that the problem is NP-complete for Sequential RASs with multi-resource requests and processes without routing decisions. In this model, resources that are freed in intermediary states are immediately required back. Additionally, some restrictions on this problem are presented, which are proved poly-

nomial. A similar problem is proven NP-complete for a different class of Sequential RASs in which alternative paths are allowed [SL01], but only one resource type is used in each stage for this kind of models, resulting again in a subclass of $S^4PR$ nets.

Toshimi Minoura [Min82] proves that the problem of deciding whether an RAS is non-live from the initial marking is NP-complete for a subclass of systems that can be modelled with $S^4PR$ nets as well. Again, processes are here acyclic and, though they can have routing decisions, the allocation of resources is forbidden in the corresponding conflicting transitions. Besides, the resource places represent mutexes (thus, they are binary) and although several resources can be simultaneously allocated to a single process, these must be acquired one at a once (i.e., no multi-resource requests).

Besides, Minoura also proves that the problem of deciding wheter a resource allocation request is safe is, for this kind of models, PSPACE-complete [Min82]. However, the author assumes that a resource request is safe iff there exists a feasible sequence for every possible routing decision that the processes can take until their successful termination. This interpretation fully makes sense if routing decisions are uncontrollable in the real-world system. The author also proves that the problem is NP-complete if it is assumed that the processes have no routing decisions. Likewise, M.A. Lawley and S.A. Reveliotis [LR01] also prove that optimal deadlock avoidance is NP-complete for a subclass of Sequential RASs in which no alternative paths per process are allowed.

Although not explicitly categorising a decision problem within a computational complexity class, many other interesting computational issues related to the RAP are discussed in different works. For instance, M.P. Fanti et al. [FMMT97] discuss the computational complexity of different scheduling policies for deadlock avoidance in RASs modelled with digraph models. X.D. Koutsoukos and P.J. Antsaklis lead some informal discussion on the computational complexity of applying optimal supervisory control techniques on Petri nets by means of the addition of monitor places [KA99]. This discussion is later complemented in the book of .V. Iordache and P.J. Antsaklis [IA06]. Other works discuss the complexity of applying deadlock prevention on RASs modelled through augmented marked graphs [CX97]: a Non-Sequential RAS model in which processes are marked graphs.

Last, but not least, F. García-Vallés [GV99] proves the NP-completeness of deciding, in general, whether the initial marking of an SIP is the minimum initial marking that makes it implicit for a subclass of systems in which the removal of the SIP makes it a live and safe free-choice system. This is also an interesting question in the context of RAS models since every resource place is a SIP and if it is made implicit then it cannot be the source of any deadlock situation. In other words, there exist a lot of problems that must be approached to outline a more or less complete picture of the complexity of dealing with the RAP in RASs. In the end, this chapter does not pretend to characterise the computational complexity of every issue concerned with

deadlock prevention/avoidance/detection but present a glimpse of the complexity of tackling these three approaches from an analytical point of view.

Section 5.2 provides a motivating example to illustrate the scope and limits of the $S^4PR$ class. Section 5.3 deals with the computational complexity of characterising non-liveness for a marked $S^4PR$ with an acceptable initial marking. This is strongly related to optimal deadlock prevention. Section 5.4 states the computational complexity in determining the markings that are doomed to deadlock, which is the key to optimal deadlock avoidance and detection in this context. Finally, in Sect. 5.5 the computational complexity in determining spurious markings is revealed, which severely affects the efficiency of structural techniques for this type of models.

## 5.2 Motivation of the complexity analysis and methodology

In order to motivate the four problems whose complexity is studied in this chapter, we introduce an example coming from a different application domain to those presented in previous chapters.

**Example 5.1** ***The video streaming case study***.

Suppose an entrepreneur is considering the installation of an on-line, on-demand video streaming service business on the Internet. In order to provide a reasonably good service, certain Quality of Service (QoS) requirements must be formally established and satisfied, for every requested transmission. These QoS specs obviously depend on a wide range of parameters such as the client type, her/his maximum supported bandwidth, the format and resolution of the requested video, etc.

To provide the service, (s)he owns a pool of video servers. These video servers are connected to a mesh network of router nodes. Some of these nodes act as gateways to the Internet. It will be assumed that multicast video streams will disseminate from the gateways onwards, so as to not increase the internal traffic. Figure 5.1 depicts the system structure (on the left, the video servers; on the right, the gateways; in the middle, the intermediate routers).

A video stream is composed of a set of fixed-size packets that must be transmitted from the sender (video server) to the receiver (client). When a receiver requests a video stream to one of the servers, a virtual circuit is constructed. All the packets of the video stream will travel through the same virtual circuit. Besides, each node of the circuit assumes its own minimum resource requirements (CPU, storage, bandwidth) for processing and transmitting each packet of the stream. These requirements will be based on the QoS specs for the transmission.

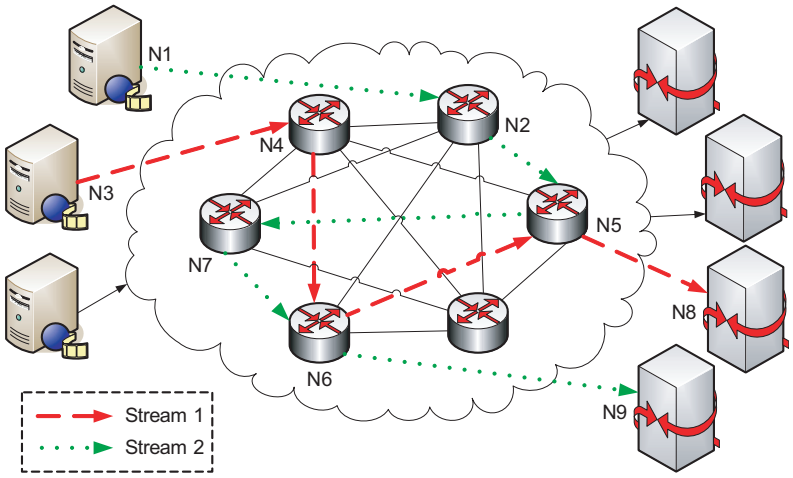



**Figure 5.1**: The video streaming system, simultaneously transmitting two video streams.

Both (circuit and resource requirements) can be determined and established through a signalling protocol in a similar vein to the Resource Reservation Protocol (RSVP) [BZB+97, VB03]. In order to maximise the system productivity and reduce costs, however, the resource reservation strategy must be 'relaxed'. Hence once a packet is effectively transmitted from a node to the next one, the required resources are freed, and must be reacquired for the next packet. Doing so, nodes can accept and manage a higher amount of concurrent streams minimising resource idling. As a drawback, when the traffic is high and resources are overused, some jittering could appear since some packets could be idle in intermediate nodes, waiting for the release of some required resources. In the worst case, a circular wait for resources could appear, and the system would reach a deadlock.

Figure 5.2 models in Petri net terms the flow of the two streams being transmitted in the system of Fig. 5.1. The different constructive elements in this $S^4PR$ model correspond to a RAS abstraction of the original system. Each video stream is modelled as a concurrent sequential process. Resources associated to each node Ni are modelled using the places labelled R-Ni. Note that there could be several resource places per router (one per each resource type, be it physical, e.g. available storage space or CPU slots, or logical, e.g. maximum number of simultaneous packets). Equivalently, there is a resource place per each node interconnection, modelling the available bandwidth and labelled BW-Ni-Nj.

All these resources can be shared among both concurrent processes. In this case, the local resources of the nodes N5 and N6 (held by resource places R-N5 and R-N6) are shared among both video streams. The resources are requested, used and freed

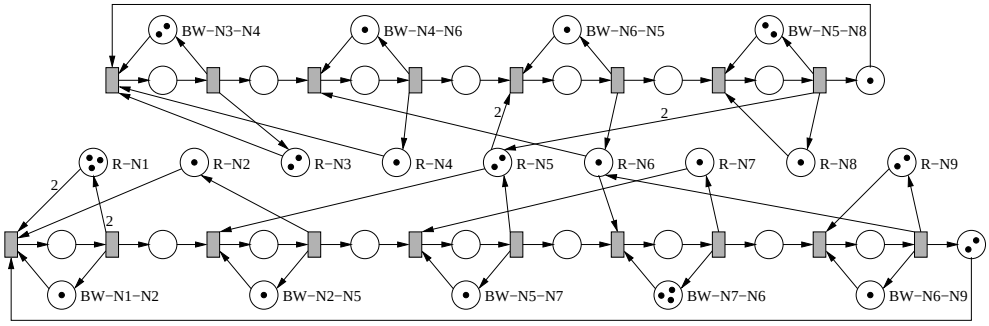



**Figure 5.2**: $S^4PR$ net system which models the system in Fig. 5.1 before the transmission of the first packets

when a packet (a token in the process subnet) is visiting the corresponding node. Finally, the idle places limit the number of potentially concurrent packets per video stream (it is assumed that this number is finite). Speaking in general terms, it is worth noting here that idle places can also be seen as special resource places, and then interpreted as the maximum number of process instances in concurrent execution for each process type.

Summing up, Fig. 5.2 depicts an $S^4PR$ net system with an acceptable initial marking: notice that all the tokens are placed in the idle and resource places, and that there are enough tokens to execute every minimal t-semiflow in isolation. This corresponds to a state of the system in which both of the streams are ready to start (both virtual circuits are established), but no packet is in transit yet (and hence every resource is available).

The marking shown in Fig. 5.3 is *not* an acceptable initial marking but, however, it *is reachable from* the acceptable initial marking depicted in Fig. 5.2. In fact, it corresponds exactly to the situation illustrated in Fig. 5.1. Here, the system has reached a deadlock, so the system depicted in Fig. 5.2 turns out to be non-live. The reader can easily check that, according to the liveness characterisation of Theorem 1.4, the set of $\mathbf{m}$-process-enabled transitions is $\{t1, t2, t3\}$ and each one of those is $\mathbf{m}$-resource-disabled: the resource places R-N5, R-N6 and R-N7 disallow their firing.

The example shows that the liveness property is a necessary requirement in the design of this class of systems. Any other requirement is subordinated to the fulfilment of the liveness property. Therefore, in the search of good algorithms to analyse the property, it is appropriate to determine the complexity of the problem itself. This is the goal of Sect. 5.3, and unfortunately the result is negative with respect to the hope of finding an efficient method. This result gives special value to the heuristics and semidecision algorithms used in the analysis of this hard property.

Departing from the liveness characterisations presented in Chapter 1 and the cor-

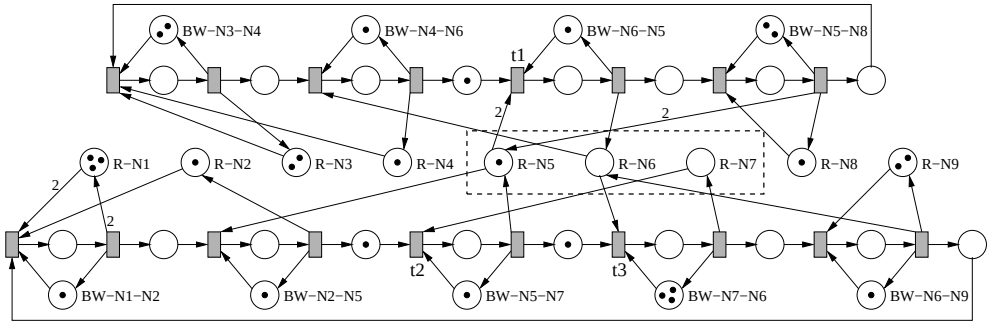



**Figure 5.3**: $S^4PR$ net system which models the actual system state in Fig. 5.1. The system is deadlocked.

responding synthesis techniques discussed in Chapt. 3 it would be possible to handle deadlocks from different standpoints. In particular, deadlock prevention (e.g. disallow a pre-established circuit if there might be a potential deadlock situation), avoidance (e.g. retain temporarily packets if they lead to deadlock situations) or detection and correction techniques (e.g. abort a video stream to free resources and unlock the system) can be applied.

These techniques are essentially based on the characterisation of the liveness property in terms of siphons or certain specific markings characterising the set of markings where deadlocks arise. Therefore, the complexity of the synthesis problem focused in markings and structured objects characterising non-live transitions, which has an iterative nature, is equivalent or subordinated to that of deciding non-liveness, which is proved NP-complete in Sect. 5.3.

Nevertheless, there exists still hope if the synthesis problem can be stated in terms of the marking frontier in the reachability graph partitioning the set of reachable markings into two subsets: (i) the set of markings that inevitably lead to a deadlock; (ii) the rest of reachable markings. If we are able to efficiently elucidate this partition, the problem can be solved easily. Unfortunately, the second result presented in this chapter (Sect. 5.4) conveys that the synthesis problem so defined is also very hard. Once again this endorses the value of those conservative heuristics designed to produce live models.

Finally, another important problem is associated with the computation of the maximal initial marking cutting the minimal number of legal states by means of structural methods. This problem can be reduced to the problem of the detection of spurious markings of the net state equation, and the complexity of this problem is proven very hard, probably because it is an interesting problem. This last result is developed in Sect. 5.5.

In this chapter, the reader is assumed to be instructed on the basics of complexity

theory [GJ79] and particularly NP-completeness. Onwards, several problems will be proved either NP or co-NP-complete. All the problem reductions will be based on the well-known (general) satisfiability problem of boolean formulae in conjunctive normal form, commonly named SATISFIABILITY (SAT), which is NP-complete. A brief reminder follows.

Let $X = \{x_1, ..., x_n\}$ be a set of boolean variables. By the process of *truth assignment*, every variable in X is assigned one value: either true or false. Let $x_i \in X$, we call a *literal* to either $x_i$ or its negation, $\overline{x_i}$. Intuitively, if the variable $x_i$ is assigned the value true, the literals $x_i$ and $\overline{x_i}$ are true and false, respectively (and vice versa if false is assigned). We define a *clause* $\mathcal{C}_j$ as a non-empty set of literals. The value of a clause is the disjunction of its literals, i.e., it is true iff at least one literal is true; and false otherwise. Finally, a *formula* $\mathcal{F}$ is a non-empty set of clauses, and its value is the conjunction of them, i.e., it is true iff all its clauses are true; false otherwise.

Without loss of generality, it will be assumed that, given a formula $\mathcal{F} = \mathcal{C}_1 \cdot ... \cdot \mathcal{C}_k$ and the set of its variables $X$, every variable $x_i \in X$ appears in at least one clause, and also that $x_i$ appears at most once in each clause, be it negated or not.

---

**Problem 5.2. SATISFIABILITY (SAT)**
*Given:* A formula $\mathcal{F}$ and the set of its variables $X$.
*To decide:* Is there a truth assignment for $X$ such that $\mathcal{F}$ is true?

---

## 5.3 On deciding liveness

The problem of optimal deadlock prevention requires determining whether a given system is non-live, in order to apply correction techniques to make the system live, such as those presented in Sect. 3.2. Here the complexity of the problem of non-liveness for a given acceptable initial marking will be studied. In particular, it will demonstrated that this problem is NP-complete. A couple of basic demonstrations are previously required, and hence will be introduced in the following. The studied problem is formally defined in this way:

---

**Problem 5.3. S$^4$PR-Non-Liveness (S$^4$PR-NL)**
*Given:* An S$^4$PR net system $\langle \mathcal{N}, \mathbf{m_0} \rangle$, being $\mathbf{m_0}$ an acceptable initial marking.
*To decide:* Is $\langle \mathcal{N}, \mathbf{m_0} \rangle$ non-live?

---

**Proposition 5.4.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an S$^4$PR net system with an acceptable initial marking. Let $\mathbf{m}$ be a reachable marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$. Then exists a firing sequence $\sigma$, $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$, such that there is no t-semiflow $\mathbf{x}$ with $\boldsymbol{\sigma} - \mathbf{x} \geq \mathbf{0}$.*

*Proof.* Without loss of generality, let $\mathbf{x}$ be a minimal t-semiflow such that $\boldsymbol{\sigma} - \mathbf{x} \geq 0$. Then it will proved that there exists a firing sequence $\sigma'$, $\mathbf{m_0} \xrightarrow{\sigma'} \mathbf{m}$, where $\boldsymbol{\sigma}' - \mathbf{x} \not\geq 0$, and $\boldsymbol{\sigma}' = \boldsymbol{\sigma} - \mathrm{k} \cdot \mathbf{x}$, with $\mathrm{k} \in \mathbb{N} \setminus \{0\}$.

$\mathbf{m}$ is potentially reachable from $\mathbf{m_0}$ with $\sigma'$ because of the net state equation: $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} = \mathbf{m_0} + \mathbf{C} \cdot (\boldsymbol{\sigma}' + \mathrm{k} \cdot \mathbf{x}) = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}'$.

The sequence $\sigma'$ is also firable because a t-semiflow $\mathbf{x}$ is a circuit of a state machine and the completion of $\mathbf{x}$ corresponds to the movement of a token in this state machine from the idle place throughout the circuit returning to the idle place. Taking into account that this token in the idle place does not use resources, while in the rest of the places of the circuit uses some resource, freezing this token in the idle place leaves a greater number of resources to fire the rest of transitions of $\sigma$. Therefore $\sigma'$ is also firable, reaching $\mathbf{m}$. □

**Lemma 5.5.** *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an $S^4PR$ net system with an acceptable initial marking, and let $\mathbf{m}$ be a reachable marking from $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$. Then exists a firing sequence $\sigma$ from $\mathbf{m_0}$ to $\mathbf{m}$, $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$, such that $|\sigma| \leq \mathrm{K} \cdot |T|$, where $\mathrm{K} = \sum_{p \in P_0} \mathbf{m_0}[p]$*

*Proof.* By Proposition 5.4, a firing sequence $\sigma_1$ exists, $\mathbf{m_0} \xrightarrow{\sigma_1} \mathbf{m}$, such that there is no t-semiflow $\mathbf{x}$ with $\boldsymbol{\sigma}_1 - \mathbf{x} \geq \mathbf{0}$. Let us suppose that $|\sigma_1| > \mathrm{K} \cdot |T|$. It is straightforward that there exists a transition $t \in T$ such that $t$ is fired at least $\mathrm{K} + 1$ times in $\sigma_1$. Since the process subnets are conservative, and the process places are empty in $\mathbf{m_0}$, for every reachable marking $\mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, $\sum_{p \in P_0 \cup P_\mathrm{S}} \mathbf{m}'[p] = \mathrm{K}$.

This means that if each token in the process places is labelled with a unique identifier $i \in [1, \mathrm{K}]$, at least one of them should visit twice the process place $p$, where $\{p\} = {}^\bullet t \cap (P_0 \cup P_\mathrm{S})$, i.e., the active process (the token) should travel through a circuit of the state machine. Since every circuit in an $\mathrm{S}^4$PR net induces a minimal t-semiflow [Tri03] then exists a t-semiflow $\mathbf{x}$, $\boldsymbol{\sigma}_1 - \mathbf{x} \geq \mathbf{0}$, contradicting the hypothesis. □

The size of the firing sequence $\sigma$ in Lemma 5.5 is polynomial in the size and population of the net. This will let us prove that $\mathrm{S}^4$PR-NL is in NP.

**Theorem 5.6.** *$S^4PR$-NL is NP-easy.*

*Proof.* The following problem will be used in the proof:

---

**Problem 5.7. $\mathrm{S}^4$PR-Bad-Marking ($\mathrm{S}^4$PR-BM)**

*Given:* An $\mathrm{S}^4$PR net system $\langle \mathcal{N}, \mathbf{m_0} \rangle$, being $\mathbf{m_0}$ an acceptable initial marking, and a firing sequence $\sigma$ such that $(|\sigma| \leq \mathrm{K} \cdot |T|), (\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m})$ and $(\mathbf{m} \neq \mathbf{m_0})$, where $\mathrm{K} = \sum_{p \in P_0} \mathbf{m_0}[p]$.

*To decide:* Does $\langle \mathcal{N}, \mathbf{m} \rangle$ hold that every $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled?

---

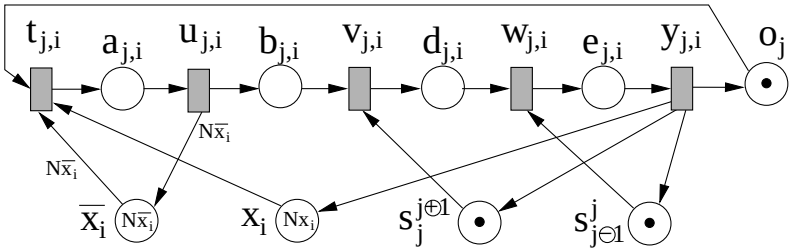



**Figure 5.4**: SAT $\rightarrow$ S$^4$PR-NL. Net $\mathcal{N}_i^j$ for each literal $x_i$ in $\mathcal{C}_j$.

1. S$^4$PR-BM is in P. Given $\sigma$, $\mathbf{m}$ can be easily computed using the net state equation. For every transition, $\mathbf{m}$-process-enabling and $\mathbf{m}$-resource-disabling can be checked in deterministic linear time in the size of $\mathcal{N}$.

2. Let $(\mathcal{N}, \mathbf{m_0}, \sigma)$ be a valid input for S$^4$PR-BM, being $(\mathcal{N}, \mathbf{m_0})$ an input for S$^4$PR-NL. Since the length of $\sigma$ is polynomial in the size of the input, it is trivial to find two encodings $e_1(\mathcal{N}, \mathbf{m_0}, \sigma)$ and $e_2(\mathcal{N}, \mathbf{m_0})$ such that $|e_1(\mathcal{N}, \mathbf{m_0}, \sigma)| \leq \mathrm{c}' \cdot |e_2(\mathcal{N}, \mathbf{m_0})|^{\mathrm{c}}$, given c, c$'$.[1]

3. S$^4$PR-NL can be *verified* in deterministic polynomial time. By Theorem 1.4, S$^4$PR-NL returns YES with input $(\mathcal{N}, \mathbf{m_0})$ iff exists a firing sequence $\sigma$, $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$ and $\mathbf{m} \neq \mathbf{m_0}$, such that every $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled. In that case, by Lemma 5.5, a firing sequence $\sigma$ can be found such that with $|\sigma| \leq \mathrm{K} \cdot |T|$. Thus S$^4$PR-NL($\mathcal{N}$, $\mathbf{m_0}$) returns YES iff exists $\sigma$ such that S$^4$PR-BM($\mathcal{N}$, $\mathbf{m_0}$, $\sigma$) returns YES.

□

Now NP-hardness will be proven, reducing SAT to S$^4$PR-NL. Let $\mathcal{F} = \mathcal{C}_1 \cdot \mathcal{C}_2 \cdot ... \cdot \mathcal{C}_{N_c}$ be a formula in conjunctive normal form, and let $X = \{x_1, ...x_k\}$ be the set of its variables. For every $x_i \in X$ let $\mathrm{N}_{x_i}$ ($\mathrm{N}_{\overline{x_i}}$) be the number of clauses of $\mathcal{F}$ in which the literal $x_i$ ($\overline{x_i}$) appears.

Also please note that, for every $j \in [1, \mathrm{N_c}]$, the index $j \oplus 1$ is defined as either $j+1$ (iff $j < \mathrm{N_c}$) or 1 (iff $j = \mathrm{N_c}$). Similarly, the index $j \ominus 1$ is defined as either $j-1$ (iff $j > 1$) or $\mathrm{N_c}$ (iff $j = 1$).

The net $\mathcal{N}_{\mathcal{F}}$ is constructed in the following compositional manner:

1. For every $x_i \in X$, $i \in [1, \mathrm{k}]$, the place $\mathrm{x}_i$ is added (in case $\mathrm{N}_{x_i} > 0$) and the place $\overline{\mathrm{x}_i}$ is added (in case $\mathrm{N}_{\overline{x_i}} > 0$).

2. For every clause $\mathcal{C}_j$, $j \in [1, \mathrm{N_c}]$, two places are added to $\mathcal{N}_{\mathcal{F}}$, called $\mathrm{o}_j$ and $\mathrm{s}_j^{j \oplus 1}$.

[1] The length of the encoding $e$ is denoted by $|e|$.

3. For every literal $x_i$ in $\mathcal{C}_j$, $i \in [1, \mathrm{k}]$, $j \in [1, \mathrm{N_c}]$, four places are added ($\mathrm{a}_{j,i}$, $\mathrm{b}_{j,i}$, $\mathrm{d}_{j,i}$, $\mathrm{e}_{j,i}$), as well as five transitions ($\mathrm{t}_{j,i}$, $\mathrm{u}_{j,i}$, $\mathrm{v}_{j,i}$, $\mathrm{w}_{j,i}$, $\mathrm{y}_{j,i}$), and they are connected to the rest of the net as depicted in Fig. 5.4.

4. For every literal $\overline{x_i}$ in $\mathcal{C}_j$, $i \in [1, \mathrm{k}]$, $j \in [1, \mathrm{N_c}]$, the same places and transitions described in the last point are added. However, they are not *exactly* connected as depicted in Fig. 5.4. Instead, the same pattern of the figure must be followed but interchanging $\mathrm{x}_i$ with $\overline{\mathrm{x}_i}$, and $\mathrm{N}_{x_i}$ with $\mathrm{N}_{\overline{x_i}}$.

In order to avoid unnecessary confusions, it should be stressed that the place $\mathrm{s}^{j}_{j\ominus 1}$ in Fig. 5.4 is the same place as $\mathrm{s}^{j'\oplus 1}_{j'}$, for $j' = j \ominus 1$ ($j = j' \oplus 1$).

The initial marking $\mathbf{m_0}$ of every place will be as shown in Fig. 5.4. The reader can check that the resulting net system $\langle \mathcal{N}_\mathcal{F}, \mathbf{m_0} \rangle$ is an $\mathrm{S^4PR}$ net system with an acceptable initial marking, where $I_{\mathcal{N}_\mathcal{F}} = [1, \mathrm{N_c}]$, every clause $\mathcal{C}_j$ results in a process subnet where $\mathrm{o}_j$ is the idle place, and the resource places are every $\mathrm{x}_i$, $\overline{\mathrm{x}_i}$, and $\mathrm{s}^{j\oplus 1}_{j}$.

In Fig. 5.5 it is depicted the resulting net $\mathcal{N}_\mathcal{F}$ for the formula $\mathcal{F} = x_1(x_1 + \overline{x_2})(x_2 + \overline{x_3})$. In this example, $\mathrm{SAT}(\mathcal{F})$ returns YES since the formula is satisfiable, e.g. assigning $x_1$="true", $x_2$="false" and $x_3$="false".

**Theorem 5.8.** *SAT → S⁴PR-NL*

*Proof.* It will be proved that $\mathrm{SAT}(\mathcal{F})$ returns YES iff $\mathrm{S^4PR\text{-}NL}(\mathcal{N}_\mathcal{F}, \mathbf{m_0})$ returns YES. By Theorem 1.4, $\langle \mathcal{N}_\mathcal{F}, \mathbf{m_0} \rangle$ is non-live iff exists a reachable $\mathbf{m}$, $\mathbf{m} \neq \mathbf{m_0}$, such that every $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled. The four necessary conditions defined by E.G. Coffman [CES71] establish that in this state a circular wait exists. This is only possible with a circular wait on the resource places $\mathrm{s}^{j\oplus 1}_{j}$, since (by construction) the only transitions that can be $\mathbf{m}$-process-enabled and $\mathbf{m}$-resource-disabled are $\mathrm{v}_{j,i}$ or $\mathrm{w}_{j,i}$. Since it is also necessary that every locked process is in a "hold and wait" state on the blocking set of resources (as expressed by Coffman [CES71]), it can easily be inferred: $\langle \mathcal{N}_\mathcal{F}, \mathbf{m_0} \rangle$ is non-live iff exists $m \in \mathrm{RS}(\mathcal{N}_\mathcal{F}, \mathbf{m_0})$ such that $\forall j \in [1, \mathrm{N_c}] \exists | i \in [1, \mathrm{k}]$ such that $\mathbf{m}[\mathrm{d}_{j,i}] = 1$ (thus, $\mathbf{m}[\mathrm{s}^{j\oplus 1}_{j}] = \mathbf{m}[\mathrm{a}_{j,i}] = \mathbf{m}[\mathrm{b}_{j,i}] = \mathbf{m}[\mathrm{e}_{j,i}] = \mathbf{m}[\mathrm{o}_j] = 0$).

Now, $\forall j \in [1, \mathrm{N_c}], i \in [1, \mathrm{k}]$ such that $\mathbf{m}[\mathrm{d}_{j,i}] = 1$, there are two mutually exclusive alternatives: either (i) $\mathbf{y}_{\mathbf{x_i}}[\mathrm{d}_{j,i}] = 1$, $\mathbf{y}_{\overline{\mathbf{x_i}}}[\mathrm{d}_{j,i}] = 0$, or (ii) $\mathbf{y}_{\overline{\mathbf{x_i}}}[\mathrm{d}_{j,i}] = 1$, $\mathbf{y}_{\mathbf{x_i}}[\mathrm{d}_{j,i}] = 0$. Note that $\mathbf{y}_{\mathbf{x_i}}$ and $\mathbf{y}_{\overline{\mathbf{x_i}}}$ are the minimal p-semiflows induced by the resource places $\mathrm{x}_i$, and $\overline{\mathrm{x}_i}$, respectively.

By construction, (i) is applied to $\mathcal{N}_\mathcal{F}$ when literal $x_i$ appears in the clause $\mathcal{C}_j$ of the formula $\mathcal{F}$. Equivalently, (ii) is applied to $\mathcal{N}_\mathcal{F}$ when literal $\overline{x_i}$ appears in the clause $\mathcal{C}_j$ of the formula $\mathcal{F}$.

If (i) holds, then $\nexists j' \in [1, \mathrm{N_c}]$, $j \neq j'$, such that $\mathbf{y}_{\overline{\mathbf{x_i}}}[\mathrm{d}_{j',i}] = 1$ and $\mathbf{m}[\mathrm{d}_{j',i}] = 1$. Otherwise, $\mathrm{t}_{j,i}$ and $\mathrm{t}_{j',i}$ should have been fired to reach $\mathbf{m}$. But the firing of $\mathrm{t}_{j,i}$ requires that no token from $\overline{\mathrm{x}_i}$ is taken, and the firing of $\mathrm{t}_{j',i}$ requires that no

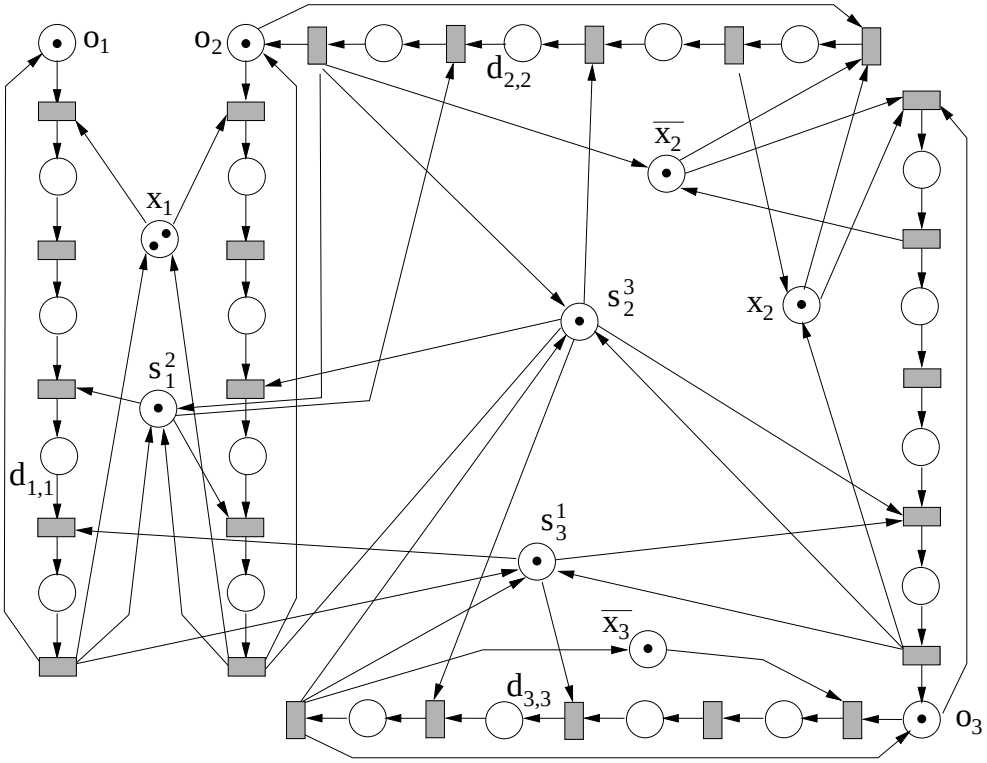



**Figure 5.5**: SAT $\rightarrow$ S$^4$PR-NL. Example: $F = x_1(x_1 + \overline{x_2})(x_2 + \overline{x_3})$.

token from $\mathrm{x}_i$ is taken, so $\mathrm{t}_{j,i}$ cannot be fired after $\mathrm{t}_{j',i}$ and vice versa, leading to a contradiction. By an analogous reasoning, if (ii) holds, then $\nexists j' \in [1, \mathrm{N_c}]$, $j \neq j'$, such that $\mathbf{y}_{\mathbf{x_i}}[\mathrm{d}_{j',i}] = 1$ and $\mathbf{m}[\mathrm{d}_{j',i}] = 1$.

Let $f$ be a truth assignment for the set of boolean variables $X$, $f : X \rightarrow \{$true, false, don't care$\}$. For every $x_i \in X$, $f(x_i)$ is defined as:

- $f(x_i)$="true" iff $\exists j \in [1, \mathrm{N_c}]$ such that $(\mathbf{m}[\mathrm{d}_{j,i}] = 1) \wedge (\mathbf{y}_{\mathbf{x_i}}[\mathrm{d}_{j,i}] = 1)$. This corresponds to case (1).
- $f(x_i)$="false" iff $\exists j \in [1, \mathrm{N_c}]$ such that $(\mathbf{m}[\mathrm{d}_{j,i}] = 1) \wedge (\mathbf{y}_{\overline{\mathbf{x_i}}}[\mathrm{d}_{j,i}] = 1)$. This corresponds to case (2).
- $f(x_i)$="don't care", iff $\nexists j \in [1, \mathrm{N_c}]$ such that $\mathbf{m}[\mathrm{d}_{j,i}] = 1$.

As we have seen, these assignments are mutually exclusive. Without loss of generality, the non-liveness condition can be finally rewritten in the following way, which proves the hypothesis: $\langle \mathcal{N}_{\mathcal{F}}, \mathbf{m_0} \rangle$ is non-live iff exists a truth assignment $f$ such that

$\forall j \in [1, \mathrm{N_c}] \exists i \in [1, \mathrm{k}]$ such that either $f(x_i)$ = "true" and $x_i$ appears in $\mathcal{C}_j$, or $f(x_i)$ = "false" and $\overline{x_i}$ appears in $\mathcal{C}_j$. □

Note that, as expected, the net system in Fig. 5.5 is non-live: the total deadlock $\langle \mathcal{N}_{\mathcal{F}}, \mathbf{m} \rangle$ is reachable from $\langle \mathcal{N}_{\mathcal{F}}, \mathbf{m_0} \rangle$, where $\mathbf{m}[\mathrm{d}_{1,1}] = \mathbf{m}[\mathrm{d}_{2,2}] = \mathbf{m}[\mathrm{d}_{3,3}] = \mathbf{m}[\mathrm{x}_1] = \mathbf{m}[\mathrm{x}_2] = 1$, being the rest of the places empty. Finally:

**Theorem 5.9.** *$S^4$PR-NL is NP-complete.*

*Proof.* $\mathrm{S^4PR}$-NL is NP-hard since, by Theorem 5.8, SAT is reducible to $\mathrm{S^4PR}$-NL, and it is also NP-easy by Theorem 5.6. □

The reader may have been left wondering why the $\mathrm{S^4PR}$ problem was defined specifically beginning from an acceptable initial marking. Instead, we could have studied the more general problem of determining if, given $\langle \mathcal{N}, \mathbf{m} \rangle$, $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, the system is non-live. Indeed, the same complexity result applies: we can easily reduce this problem to $\mathrm{S^4PR}$-NL. This is rather obvious from the fact that we can fire an arbitrary sequence from $\mathbf{m}$ trying to lead every active process to the idle places. If we are able to reach $\mathbf{m_0}$, then the reduction applies. If we are not able to reach $\mathbf{m_0}$, we will have found a marking such that every $\mathbf{m}$-process-enabled transition is $\mathbf{m}$-resource-disabled, and the system is thus non-live.

Note that this is not true in general for every solution of the net state equation, $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \mathbf{x}$, $\mathbf{x} \gneq 0$. The problem resides in the fact that $\mathrm{S^4PR}$ net systems may have killing spurious solutions, i.e., solutions of the net state equation that are not reachable and which are non-live while the system $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is live. Note that the problem of determining if a given marking is a spurious solution is studied in Subsection 5.5, and it is proven to be co-NP-complete.

## 5.4 On detecting bad markings

In previous works [Gol78, Min82, LR01, SL01], the complexity of the optimal deadlock avoidance problem has been approached for different classes of RASs, in some sense more restrictive than the $\mathrm{S^4PR}$ category, as explained earlier. These seminal results are based on the study of safeness (usually as defined in the deadlock prediction problem [Gol78]: "the existence of a feasible sequence in which to allocate the remaining resource requirements of the processes"). However, the process structure in these earlier models was finite and acyclic: once a process had satisfied all the resource requirements, it was terminated and hence removed from the system. On the other hand, an $\mathrm{S^4PR}$ net system does not have a target state; instead, the processes are structurally repetitive. Hence, it is desirable to ensure that the *feasible sequence* is arbitrarily long. This leads us to the following definition:

**Definition 5.10.** *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ *be an S⁴PR net system with an acceptable initial marking, and let* $\mathbf{m}$ *be a reachable marking,* $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$. *Then* $\langle \mathcal{N}, \mathbf{m} \rangle$ *(or simply,* $\mathbf{m}$*) is* doomed to deadlock *iff* $\exists\, \mathrm{k} \in \mathbb{N}$ *such that for every firable sequence* $\sigma$, $\mathbf{m} \xrightarrow{\sigma}$, *exists* $t \in T$ *such that* $t$ *is fired at most* k *times,* $\boldsymbol{\sigma}[t] \leq \mathrm{k}$.

The negation of this property (i.e. $\mathbf{m}$ is not doomed to deadlock) is somehow an extension of that concept of safeness and leads us to the optimal deadlock avoidance strategy. Hence, in the following a resource allocation will be considered "safe" iff $\mathbf{m}$ is not doomed to deadlock. Soon it will be seen that markings which are doomed to deadlock are well characterised in the $\mathrm{S}^4\mathrm{PR}$ class.

In contrast, an optimal deadlock detection strategy should detect iff a marking $\mathbf{m}$ is doomed to deadlock, and apply recovery techniques if so. It must be remarked that optimality is here interpreted in the strictest sense: the ability to detect the problem as soon as possible, i.e., as soon as a transition in the net is bound to die. Note that other works define optimal detection as simply deciding iff there exists a transition which is effectively dead, i.e. no longer firable, in the current marking. The latter is less general and also computationally easier. The earlier will be proved co-NP-complete:

---

**Problem 5.11. S⁴PR-Deadlock-Detection (S⁴PR-DD)**
*Given:* An $\mathrm{S}^4\mathrm{PR}$ net system $\langle \mathcal{N}, \mathbf{m_0} \rangle$, being $\mathbf{m_0}$ an acceptable initial marking, and a reachable marking $\mathbf{m}$, $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$.
*To decide:* Is $\langle \mathcal{N}, \mathbf{m} \rangle$ doomed to deadlock?

---

**Lemma 5.12.** *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be an S⁴PR net system with an acceptable initial marking, and let* $\mathbf{m}$ *be a reachable marking,* $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$. *Then* $\langle \mathcal{N}, \mathbf{m} \rangle$ *(or simply,* $\mathbf{m}$*) is doomed to deadlock iff* $\mathbf{m_0} \notin \mathrm{RS}(\mathcal{N}, \mathbf{m})$.

*Proof.* The necessary part ("only if") is rather obvious: every minimal t-semiflow is firable in isolation from $\mathbf{m_0}$. This means that a repetitive sequence can be built so that every minimal t-semiflow is successively fired, hence firing every transition an arbitrarily large number of times. Regarding the sufficient part ("if"), let us proceed by reduction to absurd. Suppose that $\mathbf{m_0} \notin \mathrm{RS}(\mathcal{N}, \mathbf{m})$, and that exists an infinite finite sequence $\sigma$, $\mathbf{m} \xrightarrow{\sigma}$ such that every transition is fired infinite times. In that case, every time a transition $t \in {}^\bullet P_0$ is fired in $\sigma$ (so the marking of an idle place is increased), the token can be frozen in the correspondent idle place (i.e. leave the token there). Since the idle places are the unique places in which no resource is used, this augments the number of resources available in the system, so the rest of active processes (i.e. tokens in the process places) can be moved in the same way as in the original sequence $\sigma$. Proceeding this way, a sequence $\sigma'$ could be constructed such

that it moves all the tokens to the idle places, reaching $\mathbf{m_0}$, unless there exists a place $p \in P_S$ with frozen tokens in it ($\mathbf{m} \xrightarrow{\sigma'} \mathbf{m'}$, $\mathbf{m'}[p] > 0$). But this is impossible, since that would imply that $p^\bullet$ is $\mathbf{m'}$-resource-disabled. Since the number of available resources has not been decreased, that would imply that $p^\bullet$ was not infinitely firable in $\sigma$, reaching a contradiction. □

Thus the problem of deadlock avoidance can be reduced to the problem of determining the reachability of the initial marking: a problem that is NP-complete, as will be seen.

---

**Problem 5.13. S$^4$PR-Reachable-Initial-Marking (S$^4$PR-RIM)**
*Given:* An S$^4$PR net system $\langle \mathcal{N}, \mathbf{m_0} \rangle$, being $\mathbf{m_0}$ an acceptable initial marking, and a reachable marking $\mathbf{m}$, $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$.
*To decide:* Is $\mathbf{m_0}$ reachable from $\langle \mathcal{N}, \mathbf{m} \rangle$?

---

**Theorem 5.14.** *S$^4$PR-RIM is NP-easy.*

*Proof.* In order to prove NP-easiness, let us introduce the following problem:

---

**Problem 5.15. S$^4$PR-Path-to-Initial-Marking (S$^4$PR-PIM)**
*Given:* An S$^4$PR net system $\langle \mathcal{N}, \mathbf{m_0} \rangle$, being $\mathbf{m_0}$ an acceptable initial marking, a reachable marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, and a firing sequence $\sigma$, $|\sigma| \leq \mathrm{K} \cdot |T|$, where $\mathrm{K} = \sum_{p \in P_0} \mathbf{m_0}[p]$.
*To decide:* Is $\mathbf{m_0}$ reached firing $m \xrightarrow{\sigma}$?

---

1. S$^4$PR-PIM is in P (this is rather trivial: checking the firability of every transition in the sequence can be done in deterministic linear time).

2. Let $(\mathcal{N}, \mathbf{m_0}, \mathbf{m}, \sigma)$ be a valid input for S$^4$PR-PIM, being $(\mathcal{N}, \mathbf{m_0}, \mathbf{m})$ an input for S$^4$PR-NL. As the size of $\sigma$ is polynomial in the number of transitions and population of the net, it is trivial to find two encodings $e_1(\mathcal{N}, \mathbf{m_0}, \mathbf{m}, \sigma)$ and $e_2(\mathcal{N}, \mathbf{m_0}, \mathbf{m})$ such that $|e_1(\mathcal{N}, \mathbf{m_0}, \mathbf{m}, \sigma)| \leq \mathrm{c}' \cdot |e_2(\mathcal{N}, \mathbf{m_0}, \mathbf{m})|^{\mathrm{c}}$, given c, c′.

3. S$^4$PR-RIM can be *verified* in deterministic polynomial time. By Lemma 5.5, but reasoning over the reverse net, if $\mathbf{m_0}$ is reachable there is a firing sequence $\sigma$, $\mathbf{m} \xrightarrow{\sigma} \mathbf{m_0}$, with ($|\sigma| \leq \mathrm{K} \cdot |T|$). Hence, S$^4$PR-NL returns YES with input $(\mathcal{N}, \mathbf{m_0})$ iff exists a firing sequence $\sigma$ such that S$^4$PR-PIM returns YES.

□

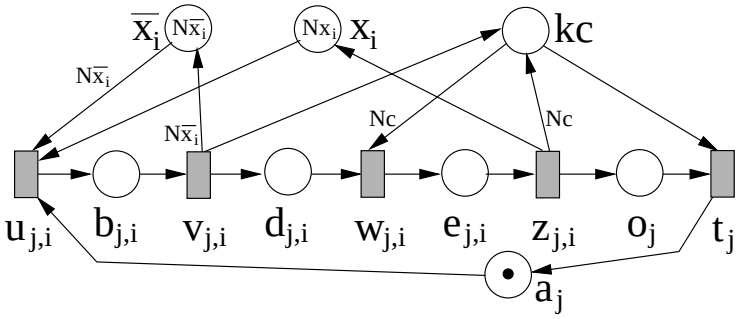



**Figure 5.6**: SAT $\rightarrow$ S$^4$PR-RIM. Net $\mathcal{N}_i^j$ for each literal $x_i$ in $\mathcal{C}_j$.

Now that NP-easiness is proven, it is required to prove NP-hardness. But this part is rather straightforward, due to the fact that (as commented before) the problem of safeness in previous works [Gol78, LR01, SL01] can be easily proven a subcase of S$^4$PR-RIM. Since the problem was already NP-hard for this models, we conclude that the problem is NP-hard through restriction [GJ79].

Nevertheless, an alternative proof will be provided on the following. Hopefully, this will more clearly illustrate the fact that S$^4$PR-RIM is NP-hard.

Let $\mathcal{F} = \mathcal{C}_1 \cdot \mathcal{C}_2 \cdot ... \cdot \mathcal{C}_{\mathrm{N_c}}$ be a formula in conjunctive normal form as introduced in Problem 5.2, and let $X = \{x_1, ...x_k\}$ be the set of its variables. Let $\mathrm{N_c}$ be the number of clauses in $\mathcal{F}$, and for every $x_i \in X$ let $\mathrm{N}_{x_i}$ ($\mathrm{N}_{\overline{x_i}}$) be the number of clauses of $\mathcal{F}$ in which the literal $x_i$ ($\overline{x_i}$) appears.

A net called $\mathcal{N}_\mathcal{F}$ will be constructed in the following compositional manner:

1. A unique place kc is added to $\mathcal{N}_\mathcal{F}$.

2. For every $x_i \in X$, $i \in [1, \mathrm{k}]$, the places $\mathrm{x}_i$ (in case $\mathrm{N}_{x_i} > 0$) and $\overline{\mathrm{x}_i}$ (in case $\mathrm{N}_{\overline{x_i}} > 0$) are added.

3. For every clause $\mathcal{C}_j$, $j \in [1, \mathrm{N_c}]$, two places ($\mathrm{o}_j$, $\mathrm{a}_j$) and one transition ($\mathrm{t}_j$) are added and connected as depicted in Fig. 5.6.

4. For every literal $x_i$ in $\mathcal{C}_j$, $i \in [1, \mathrm{k}]$, $j \in [1, \mathrm{N_c}]$, three places ($\mathrm{b}_{j,i}$, $\mathrm{d}_{j,i}$, $\mathrm{e}_{j,i}$) and four transitions ($\mathrm{u}_{j,i}$, $\mathrm{v}_{j,i}$, $\mathrm{w}_{j,i}$, $\mathrm{z}_{j,i}$) are added and connected to the rest of the net as depicted in Fig. 5.6.

5. For every literal $\overline{x_i}$ in $\mathcal{C}_j$, $i \in [1, \mathrm{k}]$, $j \in [1, \mathrm{N_c}]$, the same places and transitions described in the last point are added but, to connect them, the pattern depicted in Fig. 5.6 must be followed interchanging $\mathrm{x}_i$ per $\overline{\mathrm{x}_i}$, and $\mathrm{N}_{x_i}$ per $\mathrm{N}_{\overline{x_i}}$.

The marking $\mathbf{m}$ of every place will be as shown in Fig. 5.6. The reader can check that the resulting net system $\langle \mathcal{N}_\mathcal{F}, \mathbf{m} \rangle$ is an S$^4$PR net system, where $I_{\mathcal{N}_\mathcal{F}} = [1, \mathrm{N_c}]$,

and every clause $\mathcal{C}_j$, with $j \in I_{\mathcal{N}_\mathcal{F}}$, results in a process subnet where $\mathrm{o}_j$ is the idle place. The resource places of the net are kc and every $\mathrm{x}_i$, $\overline{\mathrm{x}_i}$.

It is also true that the marking $\mathbf{m_0}$ is an acceptable initial marking for $\mathcal{N}_\mathcal{F}$, where:

- $\mathbf{m_0}[\mathrm{kc}] = \mathrm{N_c}$,
- $\forall j \in [1, \mathrm{N_c}] : \mathbf{m_0}[\mathrm{o}_j] = 1$,
- $\forall\, i \in [1, k] : \mathbf{m_0}[\mathrm{x}_i] = \mathrm{N}_{x_i}$, $\mathbf{m}[\overline{\mathrm{x}_i}] = \mathrm{N}_{\overline{x_i}}$,
- The rest of the places are empty in $\mathbf{m_0}$.

Moreover, the marking $\mathbf{m_0}$ is reachable from $\mathbf{m}$, by firing the sequence $\sigma = t_1\ t_2\ ...\ t_{\mathrm{N_c}}$.

In Fig. 5.7 it is depicted the resulting net system $\langle \mathcal{N}_\mathcal{F}, \mathbf{m} \rangle$ for the formula $\mathcal{F} = \overline{x_2}(x_1 + \overline{x_2})x_2$. Obviously, SAT($\mathcal{F}$) returns NO since the formula is not satisfiable for any possible truth assignment.

**Theorem 5.16.** *SAT → S⁴PR-RIM*

*Proof.* Let us prove that SAT($\mathcal{F}$) returns YES iff S$^4$PR-RIM($\mathcal{N}_\mathcal{F}, \mathbf{m_0}, \mathbf{m}$) returns YES.

It is obvious that $\mathbf{m_0} \in \mathrm{RS}(\mathcal{N}, \mathbf{m})$ iff exists a firing sequence $\sigma$, $\mathbf{m} \xrightarrow{\sigma} \mathbf{m_0}$ such that $\forall j \in [1, \mathrm{N_c}] \exists i \in [1, \mathrm{k}]$ such that $\mathrm{v}_{j,i}$, $\mathrm{w}_{j,i}$ and $\mathrm{z}_{j,i}$ appear in $\sigma$.

In order to fire some $\mathrm{w}_{j,i}$ from a marking $\mathbf{m'}$, $\mathbf{m'} \in \mathrm{RS}(\mathcal{N}, \mathbf{m})$, however, it is required that $\mathbf{m'}[\mathrm{kc}] = \mathrm{N_c}$. This is, all the tokens in $P_0 \cup P_\mathrm{S}$ should be in the places labelled $\mathrm{d}_{j',i'}$ or $\mathrm{o}_{j'}$, for any $j'$, $i'$. But for marking any $\mathrm{o}_{j'}$ it is again required that some $\mathrm{w}_{j',i'}$ is fired, so the problem can be reduced to:

$\mathbf{m_0} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ iff exists $\mathbf{m} \in \mathrm{RS}(\mathcal{N}_\mathcal{F}, \mathbf{m_0})$ such that $\forall j \in [1, \mathrm{N_c}] \exists | i \in [1, \mathrm{k}]$ such that $\mathbf{m}[\mathrm{d}_{j,i}] = 1$ (thus, $\mathbf{m}[\mathrm{kc}] = \mathrm{N_c}$, $\mathbf{m}[\mathrm{a}_j] = \mathbf{m}[\mathrm{b}_{j,i}] = \mathbf{m}[\mathrm{e}_{j,i}] = \mathbf{m}[\mathrm{o}_j] = 0$).

Now the same reasoning than in the demonstration of Theorem 5.8 can be followed, taking into account that the transitions $\mathrm{t}_{j,i}$ are now called $\mathrm{u}_{j,i}$, and we conclude. □

Returning to the example in Fig. 5.7, it can be verified that $\mathbf{m_0}$ is not reachable by any means, since there is no reachable marking $\mathbf{m'}$ such that $\mathbf{m'}[\mathrm{kc}] = 3$, and this implies that the transitions $\mathrm{w}_{i,j}$ (according to the notation in Fig. 5.6) are dead, and the idle places $\mathrm{o}_j$ cannot ever be marked.

Finally:

**Theorem 5.17.** *S⁴PR-RIM is NP-complete.*

*Proof.* S$^4$PR-RIM is NP-hard since, by Theorem 5.16, SAT is reducible to S$^4$PR-RIM, and it is also NP-easy by Theorem 5.14. □

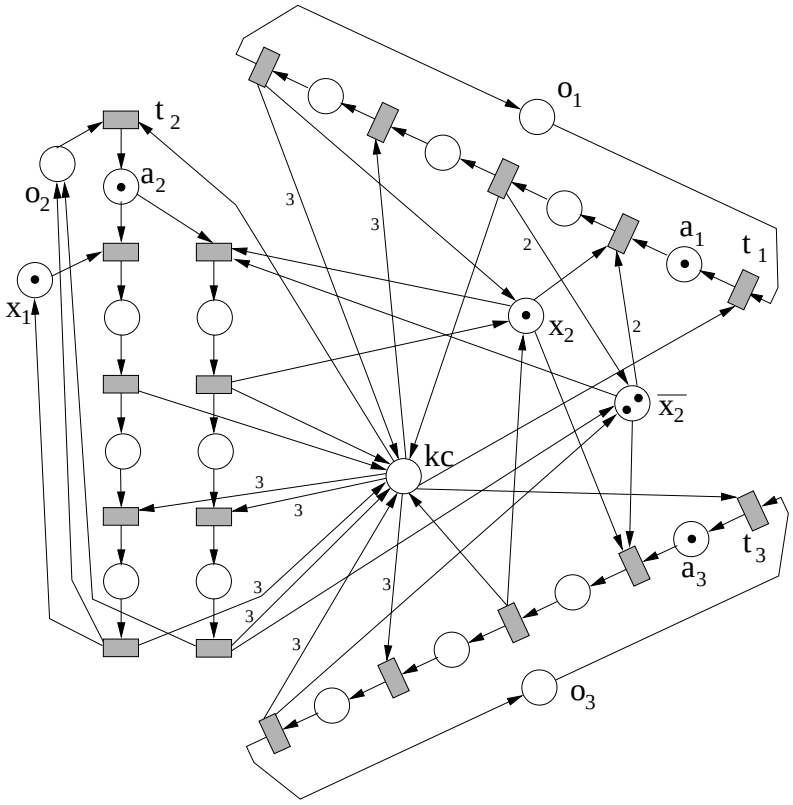



**Figure 5.7**: SAT $\rightarrow$ S$^4$PR-RIM. Example: $\mathcal{F} = \overline{x_2}(x_1 + \overline{x_2})x_2$.

Summing up, S$^4$PR-DD is co-NP-complete (i.e., optimal deadlock detection in the S$^4$PR is co-NP-complete)[2]. The problem of deciding if the firing of a transition is safe for applying optimal deadlock avoidance techniques remains NP-complete for the S$^4$PR class.

## 5.5 On detecting spurious markings

A spurious marking is a solution of the net state equation, $\mathbf{m} = \mathbf{m_0} + \mathbf{C}\cdot\mathbf{x}$, $\mathbf{x} \gneq \mathbf{0}$, that is not reachable from $\mathbf{m_0}$. A killing spurious solution is a spurious marking such that $\langle \mathcal{N}, \mathbf{m} \rangle$ is non-live. There exist Petri net subclasses, such as EQ systems [TS93], for which killing spurious solutions are not possible. In those cases, the linear description

[2]However, we remind the reader that there exists a reachable marking $\mathbf{m}'$ such that it can be structurally characterised as a bad marking by Theorem 1.4, but this does not affect the inherent computational complexity of the problem.

provided by the net state equation can be used to determine the liveness of the system.

Unfortunately, the $S^4PR$ class is not one of those classes, and this limits the potential of the net state equation for this purpose. Unless that, noticeably, spurious solutions were efficiently detectable for a given $S^4PR$ system. As will be seen, however, this is a co-NP-complete problem:

---

**Problem 5.18. $S^4PR$-Spurious-Detection ($S^4PR$-SD)**
*Given:* An $S^4PR$ net system $\langle \mathcal{N}, \mathbf{m_0} \rangle$, being $\mathbf{m_0}$ an acceptable initial marking, and $\mathbf{m} \in \mathbb{N}^{|P|}$, $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \mathbf{x}$, $\mathbf{x} \geq \mathbf{0}$.
*To decide:* Is $\mathbf{m}$ an spurious marking?

---

Intuitively, $\mathbf{m}$ is an spurious marking iff $\mathbf{m_0}$ is not reachable from $\mathbf{m}$ in its reverse (note that there may be isolated spurious solutions, i.e. not connected to the reachability space). Meanwhile, the reverse net of an $S^4PR$ net is another $S^4PR$ net. This is quite trivial, since the polarity inversion of the incidence matrix does not affect its (left or right) annullers, so the p and t-semiflows are preserved with respect to $\mathcal{N}$.

It is easy to see now that $S^4PR$-SD is co-NP-complete. This is bad news since, unless NP=P, this implies that it cannot be *verified* that a marking is spurious in deterministic polynomial time using *solely* the structure of the net.

## 5.6 Conclusions

RASs are abstractions of real systems allowing to concentrate on the study of problems such as deadlocks due to the sharing of resources used in mutual exclusion. Modelling RASs with Petri nets is particularly easy through the identification of processes with state machines and resources with monitor places representing the allocation of copies of resources. As a consequence, the $S^4PR$ subclass has already been proven specially useful and suitable for the RAS abstraction of FMSs [TGVCE05], but can be applied in many other different contexts. To illustrate this fact, a motivating example was introduced in order to depict the utility of the conceptual framework in the study and correction of deadlock problems in distributed systems and protocols, beyond the FMS context. More importantly in the context of this thesis, $S^4PR$ nets can be rather handy as a restricted, powerful model for tackling certain simple multithreaded software systems. Nevertheless, this is a most general class of models for which structural results exist characterising non-liveness. For this reason, it has been addressed an insight on the complexity of some problems related to handling with deadlocks using this kind of models.

As expected, many of the important problems are proven computationally intractable, and for this reason, the heuristics presented by F. Tricas et al. [Tri03,

TGVCE05] have special interest. Obviously these results also work as lower bounds of computational complexity for the (more general) class of $PC^2R$ nets. Regarding optimal deadlock prevention, it has been established that the problem of determining if an $S^4PR$ net system is non-live is NP-complete. Besides, evidence has been provided for NP-completeness of optimal deadlock avoidance for this class, generalising earlier results for other types of RASs which were already proven NP-hard. This was accomplished thanks to proving the equivalence of this problem with that of deciding the reachability of the initial marking. The inverse problem (optimal deadlock detection, in the strictest sense) is co-NP-complete.

Moreover, because the mathematical methods presented by Tricas et al. [TGVCE05] are based on the net state equation, an insight on the complexity of the detection of spurious markings is also relevant. The intractability of the problem, along with the existence of killing spurious solutions, constrains the practicality of the net state equation for determining non-liveness.

# Conclusions

During recent times, Petri nets have emerged as a powerful modeling paradigm for dealing with the problem of allocation of shared resources in concurrent systems. So attests the gradual flourish of Petri net-based analysis and synthesis techniques based on the study of structural properties in RASs composed of sequential processes and shared resources. Such approaches are framed within a methodological context which advocates for a first study and correction of the problems caused by resource sharing, prior to addressing other systemic problems. The application of structural correction techniques allows that the outcome of this process can survive subsequent system refinements. In essence, this approach is based on the principle that the so-named RAS view of a system is but one of many facets of the system, which is observed through a prior process of abstraction that allows applying this type of techniques on concurrent systems belonging to enormously varied application domains. This PhD thesis attempts to bring such techniques to the context of multithreaded programming, which due to its inherent complexity presents serious difficulties when it comes to successfully adapt the classical results in the field, and particularly to obtain live multithreaded software systems.

First, Chap. 1 has presented a comprehensive overview on the RASs from an essentially systemic point of view. In that sense, the chapter attempts to shed light on some aspects rarely present in the literature. First, it identifies the basic principles of the proposed methodology and puts in value the abstraction process leading to obtaining a view of the system based on processes and resources that is easily translatable to Petri net models. In that sense, the process of abstraction has been structured, identifying some of the usual features that allow categorising processes and resources at the time of their identification. Discursively, that has been integrated with the presentation of the RAS abstraction of different application domains from the literature, projecting the features identified on these domains. As a result of this process, it has been observed that there are often abundant commonalities because of which the resulting Petri net models often belong to a restricted family of subclasses. These subclasses have been introduced, compared and classified with respect to their modeling capabilities. As a paradigm of these subclasses, $S^4PR$ has been presented as

the one representing the final frontier, since it is the most general subclass for which there exist structural results characterising liveness, which facilitates the application of synthesis techniques for liveness enforcing.

Next, an RAS view of multithreaded software systems is discussed in Chapter 2. As a result, we have presented a list of requirements that an RAS model must meet to be able to model with sufficient versatility the richness of such systems. In parallel, it has also been detected that those earlier models exploited in the literature are not sufficient to meet that objective. Consequently, we have presented a new class of Petri nets, called $PC^2R$, which generalises the previous RAS subclasses and fully meets the above requirements. Unfortunately, after *crossing the Rubicon* it has been observed that finding some sort of structural liveness characterisation is a complicated task in general for such complex systems. Some of the behavioural and structural properties of these nets have been explored and compared to those found in more restricted subclasses, therefore reaching a twofold objective. First, a general overview of the family of classes to model RASs has been sketched; not only from a syntactical point of view (the classical approach in this type of work) but also from a behavioural one. And second, we have remarked some major obstacles encountered when approaching into more complex systems such as multithreaded software systems to address the problem of enforcing liveness. In addition, certain instruments are defined, such as the shrinking graph, which are useful for addressing subsequent analysis and synthesis techniques along Chap. 3 that allow dealing with the anomalous situations.

In Chap. 3, a review and categorisation of those Petri net-based techniques for enforcing liveness in the literature has been addressed. In particular, it has been proved that the classical control theory based on the restriction of firing sequences to inhibit abnormal behaviour by adding virtual resources which act as monitors is in general difficult to apply in multithreaded software systems. Rather, in this context such techniques are relegated to subclasses of systems (or subsystems of them) in which resources are used under certain significant restrictions. However, new techniques based on the addition of behaviour through the privatisation of resources, originally raised in the context of the construction of minimum adaptive deadlock-free routing algorithms for interconnection networks [Rov11], are revealed as a promising direction for obtaining live systems from a different approach. The above results have been approached from the application domain under study in this thesis. The result is a toolbox of heuristics to transform and correct, in a modular manner, multithreaded software systems modelled through the $PC^2R$ class. This required to investigate the peculiar characteristics and complex properties that siphons present in nets of such kind, as well as to define or refocus the constructive elements that allow deploying this type of techniques. In addition, a bridge has been established with classical structural theory, introducing sufficient or necessary conditions that collapse in the characterisation for simpler subclasses, thus outlining the disruption frontier of the

classical results.

Gadara [WLR+09] is a subclass of Petri nets introduced to deal with the deadlock problem due to the emergence of circular waits on binary mutexes in multithreaded software. Although the structure of each process of a Gadara net is modelled by means of a (general) state machine, this type of model presents a number of restrictions with respect to the $PC^2R$ class (e.g., binary mutexes, resources that do not participate in the internal choices, etc.) that constrain their application to a limited subclass of multithreaded software systems. In Chap. 4 bridges have been established with the family of RAS models in the literature. In particular it has been shown that Gadara nets are strongly related to a subclass of $S^4PR$ in which the internal allocation of resources to a process is deterministic. In addition, tools are provided to move from one to the other net subclass, and a formal, exhaustive proof has been presented for a siphon-based liveness characterisation for a superclass of Gadara nets: a result that, despite already being stated for Gadara nets [WLR+09], remained publicly unproven at the time of publishing our work [LGC11].

Finally, given the existing relations between the different subclasses of RASs and the special place occupied by $S^4PR$ among all these subclasses, Chap. 5 introduces a formal study of the computational complexity inherent to various problems related to the application of liveness enforcing techniques in systems modelled through the $S^4PR$ subclass. This subclass of nets is capable of modelling the RAS abstraction of some simple multithreaded software systems. In particular, it has been proved that most of the interesting problems of such systems regarding liveness fall within the family of NP-complete or co-NP-complete problems. This intractability formally justifies the search for efficient heuristics that imbues the literature and the pragmatical approach pursued in some of the results presented in previous chapters. Furthermore, these results mark a lower bound of computational complexity to more complex systems such as those addressed in the context of the first chapters: the $PC^2R$ class, for instance.

# Appendix A

# Basic Petri nets notation

A *Place/Transition (P/T)* is a 4-tuple $\mathcal{N} = \langle P, T, F, W \rangle$, where $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs, and $W$ is a total function $W : F \to \mathbb{N}^+$, being $P$, $T$ non empty, finite and disjoint sets. Elements belonging to the sets $P$ and $T$ are called respectively *places* and *transitions*, or generally nodes. P/T nets can be represented as a directed bipartite graph, where places (transitions) are graphically denoted by circles (rectangles): let $p \in P$, $t \in T$, $u = W(p,t)$, $v = W(t,p)$, there is a directed arc, labelled $u$ ($v$), beginning in $p$ ($t$) and ending in $t$ ($p$) iff $(p,t) \in F$ ($(t,p) \in F$).

The *preset* (*poset*) or set of input (output) nodes of a node $x \in P \cup T$ is denoted by ${}^\bullet x$ ($x^\bullet$), where ${}^\bullet x = \{y \in P \cup T \mid (y,x) \in F\}$ ($x^\bullet = \{y \in P \cup T \mid (x,y) \in F\}$). The preset (poset) of a set of nodes $X \subseteq P \cup T$ is denoted by ${}^\bullet X$ ($X^\bullet$), where ${}^\bullet X = \{y \mid y \in {}^\bullet x, x \in X\}$ ($X^\bullet = \{y \mid y \in x^\bullet, x \in X\}$

An *ordinary P/T net* is a net with unitary arc weights (i.e., $\forall (x,y) \in F : W(x,y) = 1$, and hence $\mathcal{N}$ can be defined with a 3-tuple $\langle P, T, F \rangle$). If the arc weights can be non-unitary, the P/T net is also called *generalised*. A *state machine* is an ordinary net such that for every $t \in T$, $|{}^\bullet t| = |t^\bullet| = 1$. An *acyclic state machine* is an ordinary net such that for every $t \in T$, $|{}^\bullet t| \leq 1$, $|t^\bullet| \leq 1$, and there is no circuit in it.

A self-loop place $p \in P$ is a place such that $p \in p^{\bullet\bullet}$. A *pure P/T net* (also self-loop free P/T net) is a net with no self-loop places. In pure P/T nets, the net can be also defined by the 3-tuple $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$, where $\mathbf{C}$ is called the *incidence matrix*, and $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$, with $\mathbf{Pre}[p,t] = W(p,t)$ iff $(p,t) \in F$ (otherwise, $\mathbf{Pre}[p,t] = 0$) and $\mathbf{Post}[p,t] = W(t,p)$ iff $(t,p) \in F$ (otherwise, $\mathbf{Post}[t,p] = 0$). Nets with self-loop places can be easily transformed into pure P/T nets without altering most significant behavioural properties, such as liveness, as shown in Fig.A.1.

A *p-flow* (*t-flow*) is a vector $\mathbf{y} \in \mathbb{Z}^{|P|}$, $\mathbf{y} \neq \mathbf{0}$ ($\mathbf{x} \in \mathbb{Z}^{|T|}$, $\mathbf{x} \neq \mathbf{0}$), which is a left (right) annuler of the incidence matrix, $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$ ($\mathbf{C} \cdot \mathbf{x} = \mathbf{0}$). The *support* of a p-flow (t-flow), denoted $\|\mathbf{y}\|$ ($\|\mathbf{x}\|$), is the set of places (transitions) for which

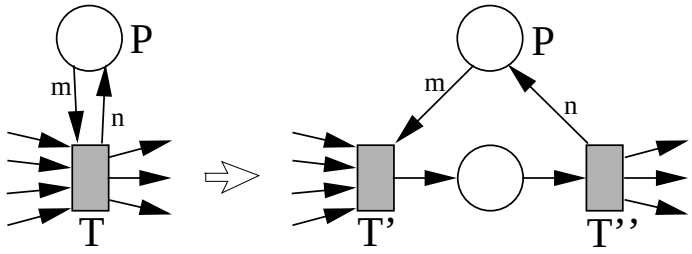



**Figure A.1**: Transformation rule: Removing self-loop places

their corresponding components in the p-flow (t-flow) are not zero, i.e., $\|\mathbf{y}\| = \{p \in P \mid \mathbf{y}[p] \neq 0\}$ ($\|\mathbf{x}\| = \{t \in T \mid \mathbf{x}[t] \neq 0\}$). The aforementioned places (transitions) are said to be *covered* by $\mathbf{y}$ ($\mathbf{x}$). A *minimal p-flow* (*minimal t-flow*) is a p-flow (t-flow) such that the g.c.d of its non-null components is one and its support $\|\mathbf{y}\|$ ($\|\mathbf{x}\|$) is not an strict superset of the support of another p-flow (t-flow). A *p-semiflow* (*t-semiflow*) is a non-negative p-flow. The P/T net $\mathcal{N}$ is *conservative* (*consistent*) iff every place (transition) is covered by a p-semiflow (t-semiflow).

A set of places $D \subseteq P$ ($\Theta \subseteq P$) is a *siphon* (*trap*) iff every place $p \in {}^\bullet D$ ($p \in \Theta^\bullet$) satisfies $p \in D^\bullet$ ($p \in {}^\bullet\Theta$). The support of a p-semiflow is a siphon (trap) but the opposite does not hold in general.

Let $\mathcal{N} = \langle P, T, F, W \rangle$ be a P/T net, and let $P' \subseteq P$ and $T' \subseteq T$, where $P', T' \neq \emptyset$. The P/T net $\mathcal{N}' = \langle P', T', F', W' \rangle$ is the subnet generated by $P'$, $T'$ iff $(x, y) \in F' \iff (x, y) \in F$ and $W'(x, y) = W(x, y)$, for every pair of nodes $x, y \in P' \cup T'$. Let $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ be a pure P/T net. Its *reverse net* $\mathcal{N}^{\mathrm{r}} = \langle P, T, \mathbf{C^r} \rangle$ is the same net with its arcs inverted, i.e. $\forall p \in P,\ t \in T : \mathbf{C^r}(p, t) = -\mathbf{C}(p, t)$. Its *dual net* $\mathcal{N}^{\mathrm{d}} = \langle P^{\mathrm{d}}, T^{\mathrm{d}}, \mathbf{C^d} \rangle$ is the result of replacing every place of $\mathcal{N}$ with a transition, and viceversa, i.e. $P^{\mathrm{d}} = T$, $T^{\mathrm{d}} = P$ and $\forall p \in P^{\mathrm{d}}, t \in T^{\mathrm{d}} : \mathbf{C^d}(p, t) = \mathbf{C}(t, p)$.

A *marking* $\mathbf{m}$ of a P/T net $\mathcal{N}$ is a vector of $\mathbb{N}^{|P|}$, assigning a finite number of *tokens* $\mathbf{m}[p]$ to every place $p \in P$. Tokens are usually represented by black dots within the places. The *support* of a marking, $\|\mathbf{m}\|$, is the set of places which are marked in $\mathbf{m}$, i.e. $\|\mathbf{m}\| = \{p \in P \mid \mathbf{m}[p] \neq 0\}$. We define a *marked P/T net* (also P/T net system) as the pair $\langle \mathcal{N}, \mathbf{m_0} \rangle$, where $\mathcal{N}$ is a P/T net, and $\mathbf{m_0}$ is a marking for $\mathcal{N}$, also called *initial marking*. $\mathcal{N}$ is said to be the structure of the system, while $\mathbf{m_0}$ represents the initial state of the system. Throughout this thesis, net markings are usually denoted in the form: $[\mathrm{P}_1^{K_1}, \mathrm{P}_2^{K_2}, ..., \mathrm{P}_n^{K_n}]$, with $K_1$, ..., $K_n \in \mathbb{N}$. For a marking $\mathbf{m}$ denoted in this way, the naturals $K_1$, ..., $K_n$ represent the values of the vector $\mathbf{m}$ corresponding to the places $\mathrm{P}_1$, ..., $\mathrm{P}_\mathrm{n}$ of the net, i.e., $\forall i \in [1, n] : \mathbf{m}[\mathrm{P}_\mathrm{i}] = K_i$. The rest of components of the vector $\mathbf{m}$ are assumed to be zero-valued.

Also in the context of this PhD thesis, p-semiflows are often represented as marking invariants in the form: $K_1 \cdot \mathbf{m}[\mathrm{P}_1] + K_2 \cdot \mathbf{m}[\mathrm{P}_2] + (...) + K_n \cdot \mathbf{m}[\mathrm{P}_\mathrm{n}] = K'$, with $K_1$,

..., $K_n$, $K' \in \mathbb{N}$. In the first part of the equality, the naturals $K_1$, ..., $K_n$ represent the values of the vector $\mathbf{y}$ corresponding to each place $\mathrm{P}_1$, ..., $\mathrm{P}_\mathrm{n}$ of the net, i.e., $\forall i \in [1,n] : \mathbf{y}[\mathrm{P_i}] = K_i$. The rest of components of the vector are assumed to be zero-valued. Meanwhile, $K'$ is the result of the weighted sum of tokens in the net for a given initial marking, where $\mathbf{y}$ determines the weighting (in other words: $K' = \mathbf{y}^T \cdot \mathbf{m_0}$).

Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a marked P/T net. A transition $t \in T$ is *enabled* (also *firable*) at $\mathbf{m_0}$ iff $\forall p \in {}^\bullet t : \mathbf{m_0}[p] \geq W(p,t)$, which is denoted by $\mathbf{m_0} \xrightarrow{t}$. The *firing* of an enabled transition $t \in T$ changes the system state or marking to $\mathbf{m_1}$, where $\forall p \in P : \mathbf{m_1}[p] = \mathbf{m_0}[p] + \mathbf{C}[p,t]$, and is denoted by $\mathbf{m_0} \xrightarrow{t} \mathbf{m_1}$. A *firing sequence* $\sigma$ from $\mathbf{m_0}$ in $\mathcal{N}$ is a non-empty sequence of transitions $\sigma = t_1\, t_2 \ldots t_k$ such that $\mathbf{m_0} \xrightarrow{t_1} \mathbf{m_1} \xrightarrow{t_2} \ldots \mathbf{m_{k-1}} \xrightarrow{t_k} \mathbf{m_k}$. The firing of $\sigma$ is denoted by $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m_k}$. The *language* of $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{L}(\mathcal{N}, \mathbf{m_0})$, is the set of firing sequences from $\langle \mathcal{N}, \mathbf{m_0} \rangle$. A marking $\mathbf{m}$ is *reachable* from $\langle \mathcal{N}, \mathbf{m_0} \rangle$ iff there exists a firing sequence $\sigma$ such that $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$. The *reachability set* $\mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ is the set of reachable markings, i.e. $\mathrm{RS}(\mathcal{N}, \mathbf{m_0}) = \{\mathbf{m} \mid \exists\, \sigma : \mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}\}$. The *reachability graph* of $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is a labelled directed graph $(V, E, L)$ such that $V = \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, $E = \{(\mathbf{m_1}, \mathbf{m_2}) \in V \times V \mid \exists t \in T : \mathbf{m}1 \xrightarrow{t} \mathbf{m}2\}$ and $L : E \to T$ is a labelling function such that $L(\mathbf{m}1, \mathbf{m}2) = \{t \in T \mid \mathbf{m}1 \xrightarrow{t} \mathbf{m}2\}$. A place $p \in P$ is a *sequential implicit place* in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ (or, simply, *implicit*) iff the removal of the place $p$ preserves all firing sequences of the original net, i.e., $\mathcal{L}(\mathcal{N}, \mathbf{m_0}) = \mathcal{L}(\mathcal{N}', \mathbf{m_0}')$, where $\langle \mathcal{N}', \mathbf{m_0}' \rangle$ is the net system resulting from removing the place $p$ from the original net.

A transition $t \in T$ is *live* iff for every reachable marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, $\exists \mathbf{m'} \in \mathrm{RS}(\mathcal{N}, \mathbf{m})$ such that $\mathbf{m'} \xrightarrow{t}$. The system $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is *live* iff every transition is live. Otherwise, $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is *non-live*. A transition $t \in T$ is *dead* at a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ iff there is no reachable marking $\mathbf{m'} \in \mathrm{RS}(\mathcal{N}, \mathbf{m})$ such that $\mathbf{m'} \xrightarrow{t}$. The marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ is a *total deadlock* of $\langle \mathcal{N}, \mathbf{m_0} \rangle$ iff every transition is dead at $\mathbf{m}$. A t-semiflow $\mathbf{x}$ is realisable iff $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ and a firing sequence $\sigma$ such that $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}$ and $\boldsymbol{\sigma} = \mathbf{x}$. A *home state* $\mathbf{m_k}$ is a marking such that it is reachable from every reachable marking, i.e. $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) : \mathbf{m_k} \in \mathrm{RS}(\mathcal{N}, \mathbf{m})$. The net system $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is *reversible* iff $\mathbf{m_0}$ is a home state. A *livelock* of $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is a terminal strongly connected component of the reachability graph with one or more arcs.

The *net state equation* of a marked pure P/T net $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is an algebraic equation defined as $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$, where $\boldsymbol{\sigma} \geq \mathbf{0}$. Every reachable marking holds the net state equation, but there may exist solutions to the equation which are not reachable markings. Thus we will call $\mathbf{m} \in \mathbb{N}^{|P|}$ a *potentially reachable marking*. The *potential reachability set* $\mathrm{PRS}(\mathcal{N}, \mathbf{m_0})$ is defined as $\mathrm{PRS}(\mathcal{N}, \mathbf{m_0}) = \{\mathbf{m} \mid \exists \boldsymbol{\sigma} \in \mathbb{N}^{|T|} : \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}, \boldsymbol{\sigma} \geq \mathbf{0}\}$.

A *directed path* (or path) $\pi$ of a P/T net $\mathcal{N}$ is a sequence of nodes $\pi = x_1\, x_2 \ldots x_n$ such that the odd components are places and the even components transitions, or viceversa, and for every pair $(x_i, x_{i+1})$, $(x_i, x_{i+1}) \in F$. An *elementary path* is a path

such that $\forall i, j \in [1, n]$ . $x_i \neq x_j$, except for $x_1 = x_n$ (which is allowed). A *general circuit* is a path such that $x_1 = x_n$. An *elementary circuit* (or simply *circuit*) is both an elementary path and a general circuit.

# Appendix B

# Some additional examples and figures

Throughout this thesis, various Petri nets are provided in order to exemplify and illustrate the various results and methods presented. Although all chapters have been written to be self-contained, many of these examples can be completed with additional graphical information providing insights into the structure and behavior of some of these nets. To this end, we include below some figures which hopefully enhance the documentary completeness of this thesis.

It should be noted that among these figures are some s-reg and resource pruning graphs of $PC^2R$ nets which do not belong to the $SOAR^2$ subclass. Although these instruments have not been formally defined for this more general class of nets, it is not difficult to extend the results presented for $SOAR^2$ to these examples. Furthermore, we believe they provide additional insight valuable enough to justify their inclusion on an appendix.

Finally, note that all figures below are grouped by chapter. The chapter number that labels each of these groups marks the chapter in which the examples that these new figures complement originally appeared.

# Chapter 2

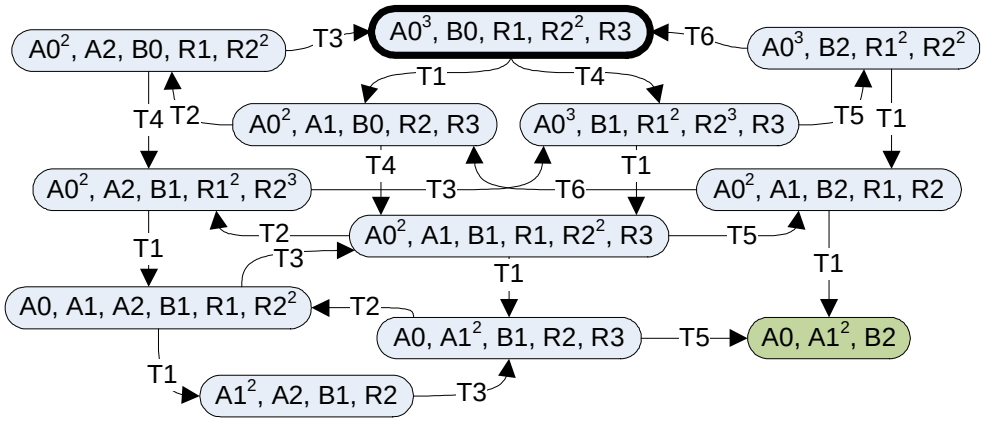



**Figure B.1**: Reachability graph of the $PC^2R$ net in Fig 2.6

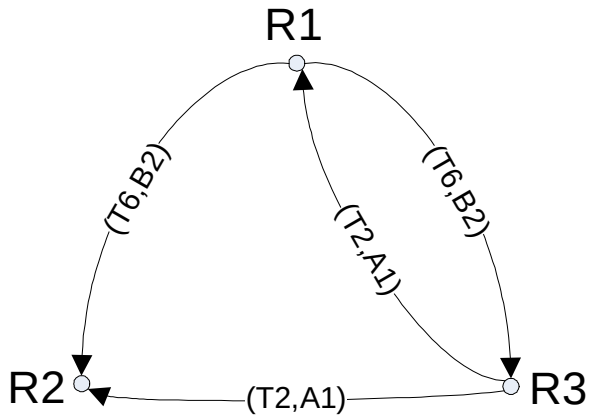



**Figure B.2**: Resource pruning graph of the $PC^2R$ net in Fig 2.6

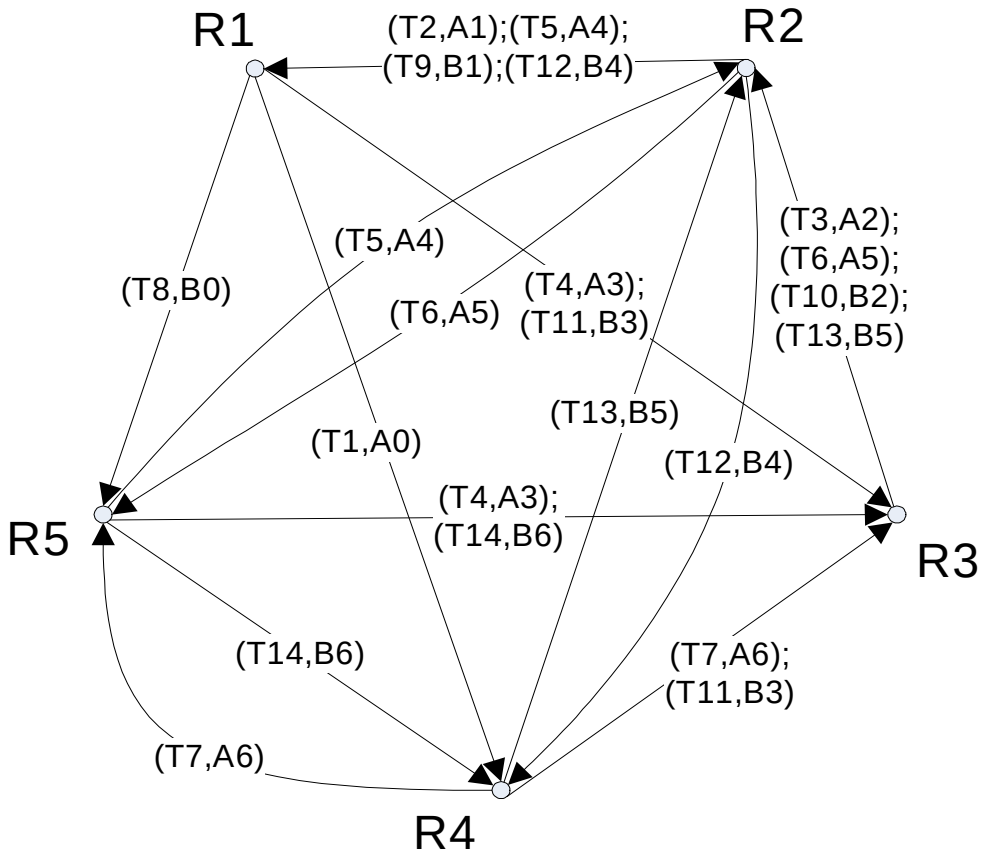



**Figure B.3**: Resource pruning graph of the $PC^2R$ net in Fig. 2.21

## Chapter 3

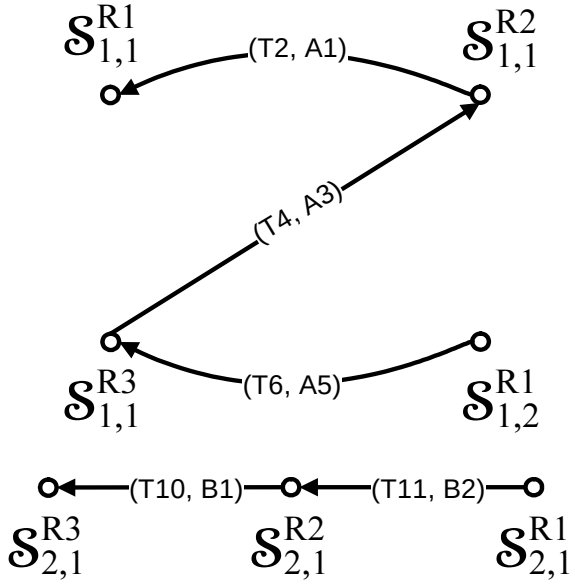



**Figure B.4**: S-reg pruning graph of Fig. 3.11, including labels

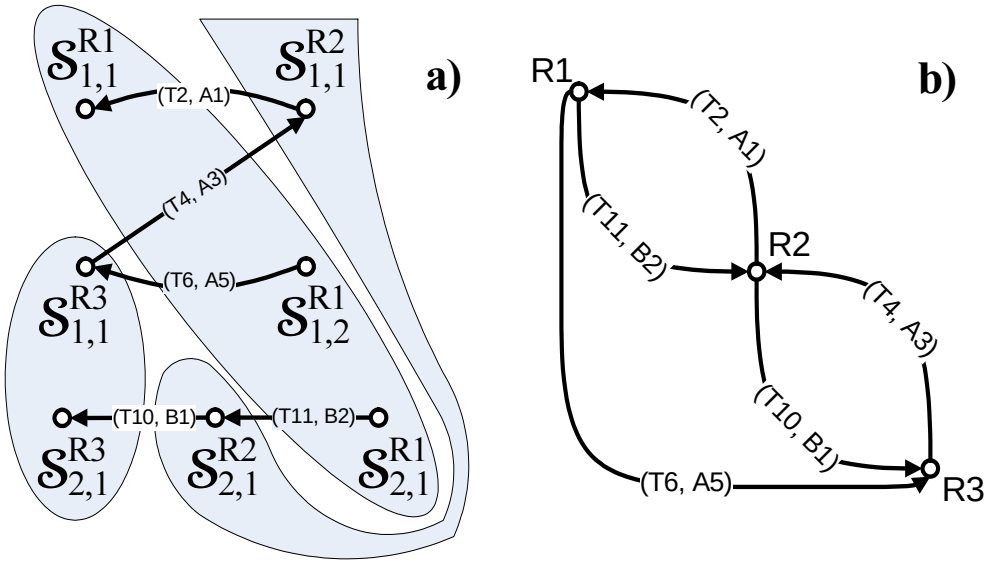



**Figure B.5**: Agglomerated resource pruning graph of Fig. 3.12, including labels

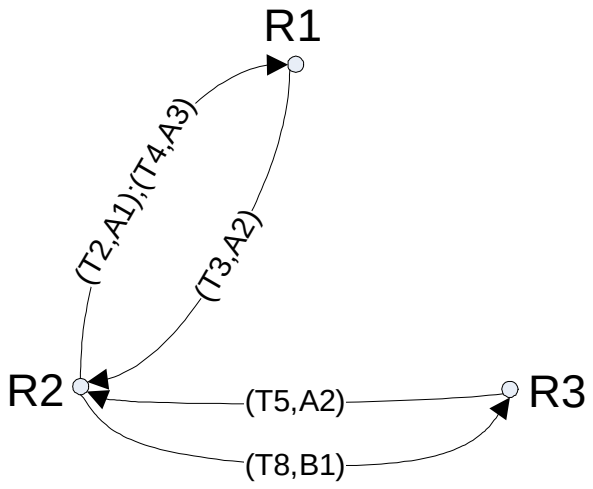



**Figure B.6**: Resource pruning graph of the net in Fig. 3.22

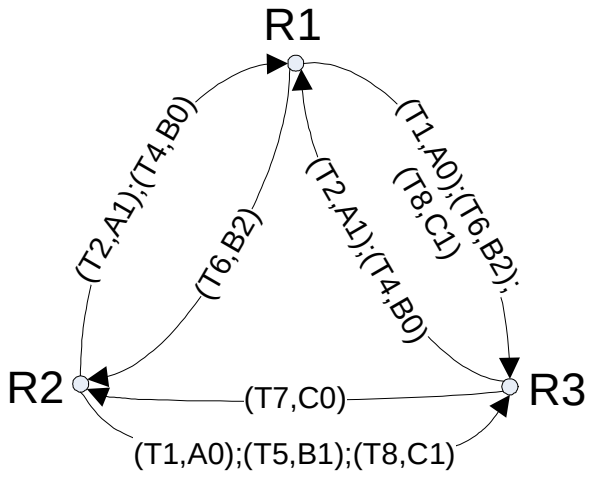



**Figure B.7**: Resource pruning graph of the net in Fig. 3.23

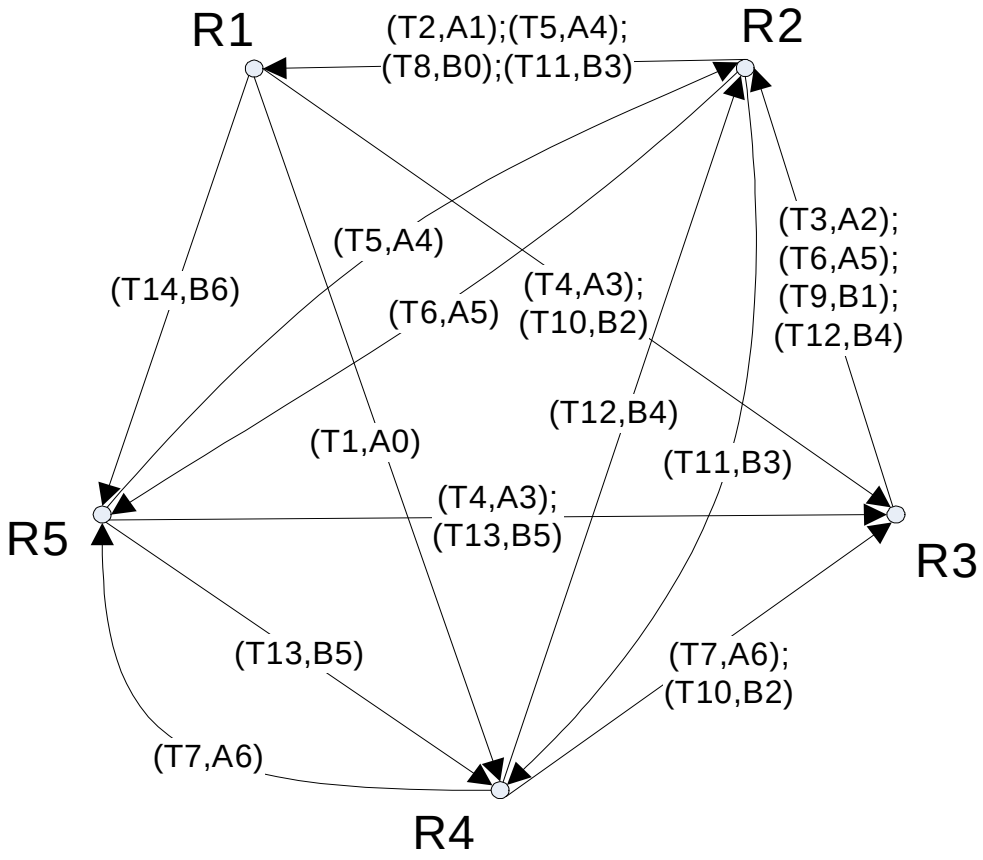



**Figure B.8**: Resource pruning graph of the net in Fig. 3.24

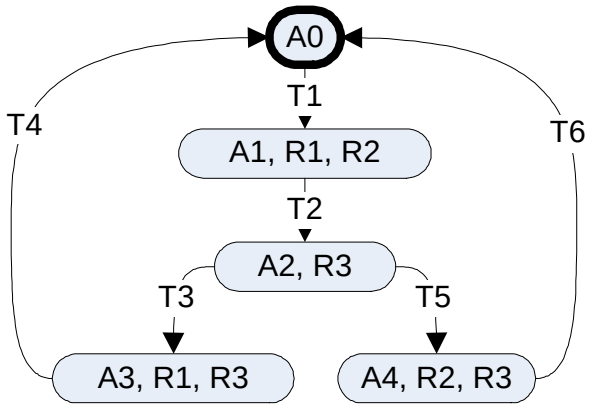



**Figure B.9**: Reachability graph of the net in Fig. 3.26

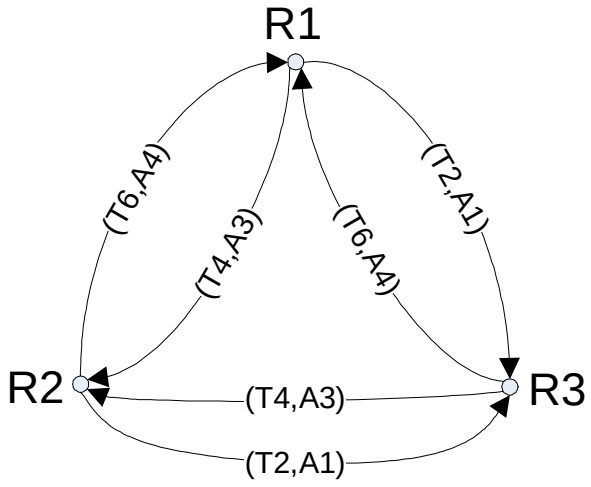



**Figure B.10**: Resource pruning graph of the net in Fig. 3.26

# Bibliography


[BV84] E. Best and K. Voss. Free choice systems have home states. *Acta Informatica 21*, pages 89–100, 1984.

[BZB+97] R. Brade, L. Zhang, S. Berson, S. Herzog, and S. Jamin. RFC 2205: Resource ReSerVation Protocol – Version 1 Functional Specification, September 1997.

[CES71] E.G. Coffman, M. Elphick, and A. Shoshani. System deadlocks. *ACM Computing Surveys*, 3(2):67–78, 1971.

[Col03] J-M. Colom. The resource allocation problem in flexible manufacturing systems. In W.M.P. van der Aalst and E. Best, editors, *Proc. of the 24th Int. Conf. on Applications and Theory of Petri Nets*, volume 2679 of *LNCS*, pages 23–35, Eindhoven, Netherlands, June 2003. Springer-Verlag.

[Col11] *Collins English Dictionary*. HarperCollins Publishers, 2011.

[CRC12] E.E. Cano, C.A. Rovetto, and J-M. Colom. An algorithm to compute the minimal siphons in $S^4PR$ nets. *Discrete Event Dynamic Systems*, 22(4):403–428, 2012.

[CS91] J-M. Colom and M. Silva. Improving the linearly based characterization of P/T nets. In Rozenberg, G., editor, *Advances in Petri Nets 1990*, volume 483 of *LNCS*, pages 113–145. Springer-Verlag, Berlin, Germany, 1991.

[CX97] F. Chu and X.L. Xie. Deadlock analysis of Petri nets using siphons and mathematical programming. *IEEE Transactions on Robotics and Automation*, 13(6):793–804, December 1997.

[Dij67] E.W. Dijkstra. The structure of the "THE"-multiprogramming system. In *Proc. of the 1st ACM Symposium on Operating System Principles*, SOSP '67, pages 10.1–10.6, New York, NY, USA, 1967. ACM.

[Dij82] E.W. Dijkstra. The mathematics behind the Banker's Algorithm. *Selecting Writings on Computing: A Personal Perspective*, pages 308–312, 1982.

[DS86] W.J. Dally and L. Seitz. The torus routing chip. *Distributed Computing*, 1(4):187–196, 1986.

[Dua95] J. Duato. Necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10):1055–1067, 1995.

[ECM95] J. Ezpeleta, J-M. Colom, and J. Martínez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 11(2):173–184, April 1995.

[EGVC98] J. Ezpeleta, F. García-Vallés, and J-M. Colom. A class of well structured Petri nets for flexible manufacturing systems. In J. Desel and M. Silva, editors, *Proc. of the 19th Int. Conf. on Application and Theory of Petri Nets*, volume 1420 of *LNCS*, pages 64–83, Lisbon, Portugal, June 1998. Springer-Verlag.

[ER04] J. Ezpeleta and L. Recalde. A deadlock avoidance approach for non-sequential resource allocation systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 34(1):93–101, January 2004.

[ETGVC02] J. Ezpeleta, F. Tricas, F. García-Vallés, and J-M. Colom. A banker's solution for deadlock avoidance in FMS with flexible routing and multiresource states. *IEEE Transactions on Robotics and Automation*, 18(4):621–625, August 2002.

[Fan02] M.P. Fanti. A deadlock avoidance strategy for AGV systems modelled by coloured Petri nets. In *Proc. of the 6th Int. Workshop on Discrete Event Systems (WODES'02)*, pages 61–66, Washington, DC, USA, 2002. IEEE Computer Society.

[FGS06] M.P. Fanti, A. Giua, and C. Seatzu. Monitor design for colored Petri nets: An application to deadlock prevention in railway networks. *Control Engineering Practice*, 14(10):1231–12472, October 2006.

[FMMT97] M.P. Fanti, B. Maione, S. Mascolo, and B. Turchiano. Event-based feedback control for deadlock avoidance in flexible production systems. *IEEE Transactions on Robotics and Automation*, 13(3):347–363, 1997.

[FZ04] M.P. Fanti and M.C. Zhou. Deadlock control methods in automated manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 34(1):5–22, January 2004.

[GDS92] A. Giua, F. DiCesare, and M. Silva. Generalized mutual exclusion contraints on nets with uncontrollable transitions. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 974–979, Chicago, USA, October 1992.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[Gol78] E. M. Gold. Deadlock prediction: Easy and difficult cases. *SIAM Journal on Computing*, 7(3):320–336, 1978.

[GRX03] A. Ghaffari, N. Rezg, and X.L. Xie. Design of a live and maximally permissive Petri net controller using the theory of regions. *IEEE Transactions on Robotics*, 19(1):137–142, 2003.

[GV99] F. García-Vallés. *Contributions to the structural and symbolic analysis of place/transition nets with applications to flexible manufacturing systems and asynchronous circuits*. PhD thesis, University of Zaragoza, Zaragoza, April 1999.

[GVC99] F. García-Vallés and J-M. Colom. Implicit places in net systems. In P. Bucholz and M. Silva, editors, *IEEE 8th International Workshop on Petri Nets and Performance Models (PNPM'99)*, pages 104–113, Zaragoza, Spain, September 1999. IEEE Computer Society Press.

[GVTCE98] F. García-Vallés, F. Tricas, J-M. Colom, and J. Ezpeleta. Optimal control of discrete event systems. In *Proc. of the 4th Int. Workshop on Discrete Event Systems (WODES98)*, pages 88–93, Cagliari, Italy, August 1998. Institution of Electrical Engineers, London.

[GW92] F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57:323–389, October 1992.

[Har80] D. Harel. On folk theorems. *Communications of the ACM*, 23(7):379–389, 1980.

[Hel13] A. Hellemans. Ring around the nanowire. *IEEE Spectrum*, 50(5):14–15, May 2013.

[Hil85] D. Hillen. Relationship between deadlock-freeness and liveness in free-choice nets. *Newsletter*, (19):28–32, February 1985.

[HJXC06] Y.S. Huang, M.D. Jeng, X.L. Xie, and D.H. Chung. Siphon-based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 36(6):1248–1256, 2006.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[HZL09] H.S. Hu, M.C. Zhou, and Z.W. Li. Liveness enforcing supervision of video streaming systems using non-sequential Petri nets. *IEEE Transactions on Multimedia*, 11(8):1446–1456, December 2009.

[HZL11] H.S. Hu, M.C. Zhou, and Z.W. Li. Supervisor optimization for deadlock resolution in automated manufacturing systems with Petri nets. *IEEE Transactions on Automation Science and Engineering*, 8(4):794–804, October 2011.

[IA06] M.V. Iordache and P.J. Antsaklis. *Supervisory Control of Concurrent Systems: A Petri Net Structural Approach*. Systems & Control: Foundations & Applications. Birkhäuser, 2006.

[IA07] M.V. Iordache and P.J. Antsaklis. Petri net supervisors for disjunctive constraints. In *Proc. of the 2007 American Control Conference*, pages 4951–4956, New York, USA, July 2007.

[IA09] M.V. Iordache and P.J. Antsaklis. Petri nets and programming: A survey. In *Proc. of the 2009 American Control Conference*, pages 4994–4999, St. Louis, Missouri, USA, June 2009.

[Joh75] D.B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[JX01] M.D. Jeng and X.L. Xie. Modeling and analysis of semiconductor manufacturing systems with degraded behavior using Petri nets and siphons. *IEEE Transactions on Robotics and Automation*, 17(5):576–588, 2001.

[JXC04] M.D. Jeng, X.L. Xie, and S.L. Chung. ERCN* merged nets for modeling degraded behavior and parallel processes in semiconductor manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 34(1):102–112, 2004.

[JXP02] M.D. Jeng, X.L. Xie, and M.Y. Peng. Process nets with resources for manufacturing modeling and their analysis. *IEEE Transactions on Robotics*, 18(6):875–889, 2002.

[KA99] X.D. Koutsoukos and P.J. Antsaklis. Computational issues in intelligent control: Discrete-event and hybrid systems. In N.K. Sinha and M.M. Gupta, editors, *Soft Computing and Intelligent Systems: Theory and Practice*, chapter 3, pages 39–69. Academic Press, October 1999.

[Kid98] P.A. Kidwell. Stalking the elusive computer bug. *IEEE Annals of the History of Computing*, 20(4):5–9, October 1998.

[KS89] J.F. Kurose and R. Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Transactions on Computers*, 38(5):705–717, 1989.

[Lau02] K. Lautenbach. Reproducibility of the empty marking. In J. Esparza and C. Lakos, editors, *Proc. of the 23rd Int. Conf. on Applications and Theory of Petri Nets*, volume 2360 of *LNCS*, pages 237–253, London, UK, 2002. Springer-Verlag.

[LDZ06] Z.W. Li, W. Ding, and R.M. Zhu. On deadlock prevention in case of failures in flexible manufacturing systems. *International Journal of Manufacturing Technology and Management*, 8(1-2):58–74, 2006.

[LGC06] J-P. López-Grao and J-M. Colom. Lender processes competing for shared resources: Beyond the $S^4PR$ paradigm. In *Proc. of the 2006 Int. Conf. on Systems, Man and Cybernetics*, pages 3052–3059, Taipei, Taiwan, October 2006. IEEE.

[LGC11] J-P. López-Grao and J-M. Colom. On the deadlock analysis of multithreaded control software. In *Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'2011)*, Toulouse, France, September 2011.

[LGC12] J-P. López-Grao and J-M. Colom. A Petri net perspective on the Resource Allocation Problem in software engineering. In K. Jensen, S. Donatelli, and J. Kleijn, editors, *Transactions On Petri Nets and Other models of Concurrency V (ToPNoC V)*, volume 6900 of *LNCS*, pages 181–200. Springer, Heidelberg, Germany, 2012.

[LGCT14] J-P. López-Grao, J-M. Colom, and F. Tricas. Structural deadlock prevention policies for Flexible Manufacturing Systems: A Petri net outlook. In J. Campos, C. Seatzu, and X.L. Xie, editors, *Formal Methods in*

*Manufacturing*, Series on Industrial Information Technology, chapter 7. CRC Press/Taylor and Francis, Toulouse, France, To appear. 2014.

[LR01] M.A. Lawley and S.A. Reveliotis. Deadlock avoidance for sequential Resource Allocation Systems: Hard and easy cases. *Int. Journal of Flexible Manufacturing Systems*, 13:385–404, 2001.

[LSW+11] H. Liao, J. Stanley, Y. Wang, S. Lafortune, S.A. Reveliotis, and S. Mahlke. Deadlock-avoidance control of multithreaded software: An efficient siphon-based algorithm for Gadara Petri nets. In *Proc. of the 50th IEEE Conf. on Decision and Control and European Control Conference (CDC-ECC 2011), 2011 50th IEEE Conference on*, pages 1142–1148. IEEE, December 2011.

[LT79] K. Lautenbach and P.S. Thiagarajan. Analysis of a resource allocation problem using Petri nets. In Syre, J.C., editor, *Proc. of the 1st European Conf. on Parallel and Distributed Processing*, pages 260–266, Toulouse, 1979. Cepadues Editions.

[LT93] N.G. Leveson and C.S. Turner. An investigation of the Therac – 25 accidents. *IEEE Computer*, 26(7):18–41, July 1993.

[LWC+13] H. Liao, Y. Wang, H.K. Cho, J. Stanley, T. Kelly, S. Lafortune, S. Mahlke, and S.A. Reveliotis. Concurrency bugs in multithreaded software: Modeling and analysis using Petri nets. *Discrete Event Dynamic Systems*, pages 1–39, 2013.

[LZ08] Z.W. Li and M.C. Zhou. Control of elementary and dependent siphons in Petri nets and their application. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 38(1):133–148, January 2008.

[LZ09] Z.W. Li and M.C. Zhou. *Deadlock Resolution in Automated Manufacturing Systems: A Novel Petri Net Approach*. Springer, New York, USA, 2009.

[Min82] T. Minoura. Deadlock avoidance revisited. *Journal of the ACM*, 29(4):1023–1048, Oct 1982.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[NR12] A. Nazeem and S.A. Reveliotis. Designing compact and maximally permissive deadlock avoidance policies for complex resource allocation systems through classification theory: The nonlinear case. *IEEE Transactions on Automatic Control*, 57(7):1670–1684, July 2012.

[PCF09] L. Piroddi, R. Cordone, and I. Fumagalli. Combined siphon and marking generation for deadlock prevention in Petri nets. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 39(3):650–661, May 2009.

[PR01] J. Park and S.A. Reveliotis. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. *IEEE Transactions on Automatic Control*, 46(10):1572–1583, 2001.

[Rev99] S.A. Reveliotis. Accomodating FMS operational contingencies through routing flexibility. *IEEE Transactions on Robotics and Automation*, 15(1):3–19, 1999.

[Rev00] S.A. Reveliotis. Conflict resolution in AGV systems. *IIE Transactions*, 32(7):647–659, 2000.

[Rev03] S.A. Reveliotis. On the siphon-based characterization of liveness in sequential resource allocation systems. In van der Aalst, W.M.P. and Best, E., editor, *Proc. of the 24th Int. Conf. on Applications and Theory of Petri Nets*, volume 2679 of *LNCS*, pages 241–255, Eindhoven, Netherlands, June 2003. Springer-Verlag.

[Rev07] S.A. Reveliotis. Implicit siphon control and its role in the liveness-enforcing supervision of sequential resource allocation systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 37(3):319–328, May 2007.

[RLF97] S.A. Reveliotis, M.A. Lawley, and P.M. Ferreira. Polynomial complexity deadlock avoidance policies for sequential resource allocation systems. *IEEE Transactions on Automatic Control*, 42(10):1344–1357, 1997.

[Rov11] C.A. Rovetto. *Métodos basados en Redes de Petri para el diseño de algoritmos de encaminamiento adaptativos mínimos libres de bloqueos.* PhD thesis, University of Zaragoza, Zaragoza, September 2011.

[SC88] M. Silva and J-M. Colom. On the computation of structural synchronic invariants in P/T nets. In G. Rozenberg, editor, *Advances in Petri Nets 1988*, volume 340 of *LNCS*, pages 386–417. Springer-Verlag, Berlin, 1988.

[Sil93] M. Silva. Introducing Petri nets. In F. DiCesare, G. Harhalakis, J-M. Proth, M. Silva, and F. Vernadat, editors, *Practice of Petri nets in manufacturing*, pages 1–62. Chapman and Hall, 1993.

[SL01] W. Sulistyono and M.A. Lawley. Deadlock avoidance for manufacturing systems with partially ordered process plans. *IEEE Transactions on Robotics and Automation*, 17(6):819–832, 2001.

[Ter04] E. Teruel. *Structure theory of weighted place/transition net systems: The equal conflict hiatus*. PhD thesis, University of Zaragoza, Zaragoza, June 2004.

[TGVCE05] F. Tricas, F. García-Vallés, J-M. Colom, and J. Ezpeleta. A Petri net structure-based deadlock prevention solution for sequential resource allocation systems. In *Proc. of the 2005 Int. Conf. on Robotics and Automation (ICRA)*, pages 272–278, Barcelona, Spain, April 2005. IEEE.

[Tip95] F. Tip. A survey of program slicing techniques. *Journal of programming languages*, 3(3):121–189, 1995.

[Tri03] F. Tricas. *Deadlock analysis, prevention and avoidance in sequential resource allocation systems*. PhD thesis, University of Zaragoza, Zaragoza, May 2003.

[TS93] E. Teruel and M. Silva. Liveness and home states in equal conflict systems. In M. Ajmone Marsan, editor, *Proc. of the 14th Int. Conf. on Application and Theory of Petri Nets*, volume 691 of *LNCS*, pages 415–432. Springer-Verlag, 1993.

[UZ07] M. Uzam and M.C. Zhou. An iterative synthesis approach to Petri net-based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 37(3):362–371, May 2007.

[VB03] M. E. Villapol and J. Billington. Analysing properties of the resource reservation protocol. In W.M.P. van der Aalst and E. Best, editors, *Proc. of the 24th Int. Conf. on Applications and Theory of Petri Nets*, volume 2679 of *LNCS*, pages 377–396. Springer-Verlag, June 2003.

[Wan09] Y. Wang. *Software Failure Avoidance Using Discrete Control Theory*. PhD thesis, University of Michigan, Michigan, 2009.

[Wei84] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, (4):352–357, 1984.

[Wei91] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, September 1991.

[WLR+09] Y. Wang, H. Liao, S.A. Reveliotis, T. Kelly, S. Mahlke, and S. Lafortune. Gadara nets: Modeling and analyzing lock allocation for deadlock avoidance in multithreaded software. In *Proc. of the 49th IEEE Conf. on Decision and Control*, pages 4971–4976, Atlanta, Georgia, USA, December 2009.

[WZ05] N. Wu and M.C. Zhou. Modeling and deadlock avoidance of automated manufacturing systems with multiple automated guided vehicles. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(6):11931202, December 2005.

[XJ99] X.L. Xie and M.D. Jeng. ERCN-merged nets and their analysis using siphons. *IEEE Transactions on Robotics and Automation*, 29(4):692–703, 1999.

[Zei84] B.P. Zeigler. Multifaceted modeling methodology: Grappling with the irreducible complexity of systems. *Behavioral Science*, 29:169–178, 1984.