Análisis, prevención y evitación de bloqueos en sistemas secuenciales de asignación de recursos

Fernando Tricas García

TESIS DOCTORAL

Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Febrero 2003

A Alicia, gracias.
Para Carmen y Juan.

Agradecimientos

Mi agradecimiento al director de esta tesis, Joaquín Ezpeleta, que me dió la ayuda necesaria para progresar en este trabajo. Este agradecimiento ha de ser forzosamente extendido a José Manuel Colom y Fernando García Vallés, cuya ayuda ha sido decisiva. También le corresponde una parte del agradecimiento a Javier Martínez, que me introdujo en mis primeros pasos en este trabajo.

También quiero agradecer a todos los miembros del Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza, en particular a la gente del grupo de métodos formales, por su pronta disposición a echar una mano cuando ha hecho falta. Especial mención merece el personal de administración y servicios encargado de los equipos informáticos del departamento, en particular José Antonio Gutierrez Elipe, siempre dispuesto a buscarle las vueltas a cualquier problema con los computadores y a solucionar los problemas que han ido apareciendo.

El trabajo desarrollado en esta tesis ha sido apoyado financieramente por una beca de la D.G.A. y la participación en los proyectos TIC91-0354, TIC95-0614-C03-01, TAP98-0679, TIC2001-1819, y la acción integrada Hispano-Germana, HA2000-0047, financiados por la C.I.C.Y.T, PLAN NACIONAL DE I+D. También se recibió financiación a través del proyecto CHRX - CT94 - 0452 de la CEE

Tengo que expresar agradecimiento infinito a mi familia: por la paciencia y por su apoyo incondicional. No creo que me sea posible devolverles la deuda acumulada durante estos años, pero prometo intentar compensarles a partir de ahora.

Finalmente tengo que expresar mi profundo agradecimiento y admiración a los desarrolladores del sistema GNU/Linux y los múltiples programas de este entorno que han ayudado a que el trabajo con las máquinas haya sido menos duro. A Enrique Teruel, Salvador Sans Rica, Alberto Tappe Martínez, y Germán Lozano Fernández por la parte que les corresponde a cada uno de ellos en el desarrollo de HARP y herramientas relacionadas. A Giovanni Chiola y otros investigadores de la Universidad de Turín, autores del programa GreatSPN, utilizado para dibujar muchas de las redes que aparecen en este trabajo. A los desarrolladores de la Optimization Subroutine Library de IBM, que ha sido utilizada para la resolución de los problemas de programación entera del capítulo 3. Finalmente, a los desarrolladores de daVinci, cuya herramienta ha sido empleada para dibujar algunos de los grafos del capítulo 4.

Resumen

El propósito de este trabajo es generalizar y extender los resultados existentes en el análisis, prevención y evitación de bloqueos en sistemas de asignación de recursos, con una atención especial hacia los sistemas de fabricación flexible.

En este sentido, se proponen nuevas clases de sistemas con restricciones similares a las que podemos encontrar dentro del ámbito de los sistemas de fabricación. En un primer paso se estudiarán las propiedades estructurales de estas clases para comprobar que son adecuadas para el modelado y análisis del tipo de problemas considerado.

Las soluciones al problema de los bloqueos se presentarán desde dos puntos de vista: prevención y evitación de los problemas de bloqueo, junto con algunos datos comparativos con otras soluciones al problema. El objetivo es obtener políticas de control muy permisivas, que puedan implantarse según diferentes consideraciones, proporcionando flexibilidad al diseñador del sistema.

Finalmente se propone una mejora de un método de cálculo de cerrojos. Estas componentes estructurales están ligadas a la existencia de problemas de bloqueo en algunas clases de sistemas, y en ese sentido es muy conveniente disponer de métodos eficientes para su cálculo. El método propuesto mejora a los existentes mediante la utilización de paralelismo, y la adaptación a las características de los sistemas considerados.

Abstract

This work concentrates on deadlock problems in concurrent systems due to the common use of system resources organized in what is commonly known as Sequential Resource Allocation Systems and paying a special attention to subclasses of manufacturing systems.

To do that, special classes of Petri net models are defined that allow to capture resource allocation events used to synchronize processes that have to share a set of reusable system resources. The classes of Petri nets introduced are studied from the structure point of view, showing the clear mapping among system and model structures. It is also shown how deadlock related situations can be explained in terms of markings and model structures.

To solve deadlock problems, two different approaches are adopted. The first one is known as a deadlock prevention perspective, and makes an intensive use of different liveness characterizations developed in this work. The final result is a deadlock prevention algorithm that iteratively constrains the language of the input model so that the final controlled model is live in terms of Petri net definitions, which implies that the controlled system is free of deadlocks and ensures that the execution of any active process can terminate. The second approach falls into the deadlock avoidance family of solutions. In this work it is shown how the specific characteristics of the class of systems in consideration can be used to extend and improve the well-known Banker's solution for deadlock avoidance, allowing us to give a solution to deadlock problems in the most general class of sequential resource allocation systems.

In both cases, and taking into account that obtaining the most permissive solution is NP-complete, the proposed solutions are experimentally compared with other solutions in order to get insight of how permissive the proposed algorithms are, showing they provide a good trade-off between computation cost and permissiveness.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

When several activities need to be done, and they can be in progress at the same time, we can configure them as a concurrent system: users trying to run programs in an operating system, processes trying to access to a database system, clients trying to do some bank transactions, parts being processed in a production plant, etc.

A concurrent system can be seen as the composition of a set of independent, interacting components. The management of such a concurrent system is the management of these different simultaneous activities and the interactions among them. In many cases, a concurrent organization can improve the system performance, abilities, and usage.

As a classical example of concurrent system, let us recall the dining philosophers problem, proposed in [Dij65]:

> Five philosophers spend their lives thinking and eating. The philosophers share a common circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table there is a bowl of spaghetti, and the table is laid with five forks, as shown in Figure 1.1. When a philosopher does think he does not interact with other philosophers. From time to time, a philosopher gets hungry. In order to eat he must try to pick up the two forks that are closest (and are shared with his left and right neighbors), but may only pick up one fork at a time. He cannot pick up a fork already held by a neighbor. When a hungry philosopher has both his forks at the same time he eats

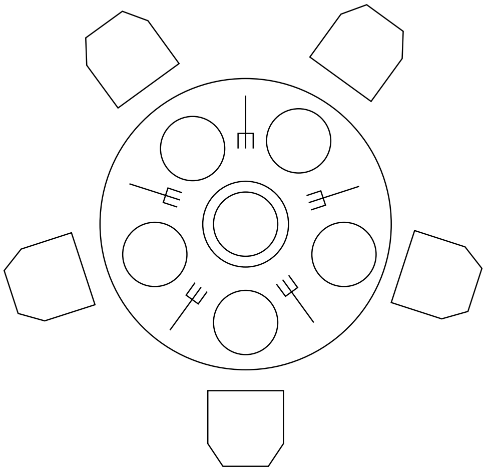


Figure 1.1: The dining philosophers problem

> without releasing them and when he has finished eating, he puts down both forks and starts thinking again.

These systems, being able to carry out several activities simultaneously, have a new property: the sequence of operations involved in the different concurrent activities show only a *partial ordering*, as opposed to sequential systems, where a *total ordering* is imposed on the execution of the system activities.

In the previous example, each philosopher can be seen as a sequential system: when he is hungry, he gets the forks, starts eating, finishes eating, and releases the forks. When all the philosophers are considered this same ordering is acceptable (for each one individually), but no total ordering relation can be established if the whole set of operations is considered. This uncertainty over the precise order of some events is a property that is referred to as *nondeterminism*.

Usually, a concurrent system is generated by a set of agents, executing each one of them their own sequence of activities, but that need to interact with others in order to terminate their tasks. This introduces a new concept in concurrent systems, which does not exist in sequential systems: interaction. In general, this interaction can occur in three different circumstances [SPG91]:

- **Competition** for shared resources: several processes may need the use of the same set of resources, having in some occasions to compete for them. The term *resources* is used here in a broad sense: it will represent the physical or logical entities needed to carry out the system activities (in the philosopher example they are the forks).

For example, each philosopher needs to share a fork with one of his neighbors. To eat, they need to compete: if one of the philosophers gets the fork, the other will not be able to eat until the former release the fork.

- **Cooperation**. This can be done in two ways:
  - **Exchange** of data between processes: the results of some processes can be useful/necessary for other processes, as they may need them in order to advance.

    For example, a philosopher could inform his neighbors about when he expects to finish eating, in order to let them know when to get the forks. Of course, and if it has sense, they could share the result of their thinking periods.
  - **Temporal considerations**: that is, when the activities need to occur and how their relative timings are: at the same time (in parallel), one after another (in a sequential way), etc.

    For example, a philosopher only can begin to eat before or after his two neighbors use the corresponding forks.

In these three cases the processes need to *synchronize* their activities; either to avoid conflicts, or to cooperate to achieve some task.

Concurrent systems may/must exhibit some specific properties which can be non–defined (or trivially obtained) in non–concurrent ones. These are properties related to the interactions among the processes. They can be classified as safety properties and liveness properties.

- **Safety properties**: these properties represent that nothing bad will ever happen. That is, they are related to the avoidance/non–existence of bad states. Let us remark the following desirable properties:
  - **Mutual exclusion** is needed when some sections of the concurrent system have to be atomic respect to other sections. That is, when one of these sections is being executed, the others cannot be being executed.

    In the philosophers example, two neighbor philosophers cannot be eating at the same time. The sections ‘eating’ of those two philosophers are in mutual exclusion.

  - **Absence of deadlock**; a deadlock occurs when some processes are waiting for the evolution of other processes, that are also waiting for the former ones to evolve.

    For example, if there are activities using resources and waiting for the resources that others are holding, and if these activities are holding some resources requested by the first ones all of them will be waiting for each other's resources and no evolution will be possible.

    If the five philosophers get hungry at the same time, all of them get the fork that is on their right, and they are actively observing the others until the other fork is available, the reached situation is a deadlock: none of the philosophers will releases his fork, and none of them will be able to eat because the other fork is being held by another philosopher.

  - **Absence of race conditions**; race conditions occur when several entities are about to perform some action. Depending on the exact timing, they will perform the action following some ordering. There is a problem when the correct result depends on this ordering.

    For example, if a philosopher checks for the availability of his right fork and then requests it, his colleague on the right can be fast enough to get the fork between these two steps.

- **Liveness properties** represent that some good things will eventually happen. That is, they are related to the existence of good behavioral properties: it not only works, but it works well. Let us remark the following ones:

  - **Absence of Livelock**; a livelock occurs when an entity is busy waiting for some event to occur, and it cannot be ensured this to happen.

    Let us suppose that the philosophers have adopted the following policy: a philosopher who becomes hungry will get first the fork on the left, and then the fork on the right. If the fork on the right is not available, he will desist and leave the left fork on the table. Now, the philosopher number one becomes hungry: he gets his left fork, then the right one and starts eating. While he is eating, the philosopher number three becomes hungry, gets his left fork and later the right one. The philosopher number one finishes eating and releases his forks. However, before the number three finishes eating, philosopher number one becomes hungry again. Both philosophers continue eating and thinking following the same pattern. The philosopher number two (who is located between number one and three) will never eat: when philosopher number one is eating, he will not be able to get his left fork; when philosopher number three is eating, he will take the left fork but he will not be able to get the right one and will return to thinking state.

- **Fairness** properties are related to every entity (user, process, etc.) being able to carry out its activity in similar conditions as the rest of entities in the system.

  If some philosopher gets hungry very often and is very fast acquiring the forks, or he gets the forks and does not release them, the philosophers next to him will not be able to eat, and the system is not fair.

- **Absence of starvation**; starvation occurs when an entity needs some system event to occur and it is repeatedly overtaken in such a way that it is not guaranteed that the activating event will occur in finite time.

  Both deadlock and livelock situations presented above will make that no philosopher can eat, so they are clear cases of (literal) starvation. Another example can be when some philosophers never get to eat because their neighbors are faster.

1.2 The deadlock problem

Remember that a deadlock occurs when some processes are waiting for the evolution of other processes, that are also waiting for the former ones to evolve.

When a deadlock affects all the system activities, a **total deadlock** occurs. When it only affects some activities, it is called a **partial deadlock**. Both of them are undesirable situations since they make that some (or all) of the activities cannot terminate. Furthermore, in some cases no new activities can start, or if they can start, they will never terminate.

Let us remark that two kinds of deadlocks can appear in concurrent systems (see for example [Sin89a]): resource deadlocks and communication deadlocks:

- In **resource deadlocks**, processes make access to resources (data objects in database systems, machines, tools or buffer space in manufacturing systems, etc.). When the state of a concurrent system is such that each process is waiting for some resource that is being held by other process in the set, in such a way that any further change of state depends on the allocation of one of the involved resources, we say that it is a resource deadlock.

In the previous example, the resources are the forks; the philosophers need them in order to eat. If all of them decide to start eating at the same time and get their left fork, a deadlock is reached.

- In **communication deadlocks**, processes communicate via message passing. A communication deadlock corresponds to a system state such that a set of processes exist so that each one of them is blocked, waiting to receive some message from another process in the set, but none of them can deliver his messages.

  Let us now suppose that the philosophers used as example decide to modify the synchronization protocol: before taking a fork, a philosopher will ask the philosopher who is near to it whether the fork is free or not. If it is free, he will get it; if not, he will request the other philosopher to inform him about when he will release the fork so that he can get it. If all the philosophers decide to eat at the same time, they'll ask the left philosopher about the fork, they will be able to get this fork and then, they will ask to the right philosopher for the other one. None of them will be able to release the corresponding fork, so none of them will be able to notify the other philosopher that he has finished. We will have a situation where all the philosophers are waiting for the message about the availability of its right fork, but none of them will be able to send such message.

As stated before, this work concentrates on resource deadlock problems. In resource related deadlocks, four necessary conditions must occur (see, for example [CES71]):

- **Mutual exclusion:** At least one resource must be held exclusively; that is, it can be used by only one process at a time. Other processes requesting this resource will be delayed until the resource has been released by the process that is using it.

- **Hold and wait:** There must be at least one process that is holding a resource and waiting for other resources currently held by another processes.

- **No preemption:** Processes cannot be forced by any external entity to release resources.

- **Circular wait:** There must exist a set of waiting processes that can be ordered in such a way that each one of them is waiting for a resource that is held by the next one, and the last one is waiting for a resource held by the first one.

In the philosophers example, at the described deadlock situation, we can see that the four conditions hold:

- A fork can be used by just a philosopher at a time (mutual exclusion).
- Each philosopher is waiting for the philosopher at its left, in order to get the fork (hold and wait).
- There is no way for a philosopher to release its fork without obtaining first the other fork and eating (no preemption).
- Each philosopher is waiting for the philosopher at his left, who is waiting for the philosopher at his left, and so on, in such a way that they are in a circular wait.

1.2.1 Strategies to deal with the deadlock problems

Typically there are four strategies to deal with deadlock problems:

- **To ignore the problem** (The Ostrich algorithm [Tan87]). The idea is to leave the system evolve, without worrying about deadlock problems. This strategy is adequate when deadlock problems are not as frequent as other events that force the system halting (breakdown, reconfiguration, etc.) and a deadlock is not a risky situation.
- **Deadlock Detection and Recovery**. The system freely evolves. A monitoring subsystem is running; when a deadlock situation is detected, a rollback process should move the system to an adequate state. A recovery strategy requires to "kill" some of the active deadlocked processes. In some cases (in an operating system, for instance) this can be easily done. However, in other systems this can be almost impossible or very expensive (imagine to have to move a plane or a car in order to free some resources).

  For example, in a deadlock situation, the philosophers can discuss. Then, they can decide which ones have to release their forks in order the others can eat.

- **Deadlock Prevention**. The system is designed to be deadlock–free by ensuring that no deadlock situation can occur. In some cases, this can mean that some restrictions have to be imposed to the situations under which a process can be activated (or it can evolve). Usually, some off–line computations are needed before a prevention approach can be applied.

  Let us consider, for instance, a system for which it is known that it can deadlock if there are three or more active processes, but no deadlock can occur with two or less active processes. Then, a prevention policy could consist in ensuring that no more than two processes are active at the same time. As a second example, let us consider another system where a set of processes share a set of non–consumable resources. It is well known that if an ordering can be established in the set of resources in such a way that each process uses the resources according to this ordering, no deadlock can occur (since no circular wait is possible). Then, a prevention policy would consist in designing the system in such a way that only processes that request the resources following this ordering are accepted. For example, if this approach is adopted for the philosophers problem, it is necessary to convince the last one to get the forks in reverse order than the other philosophers (first the right fork, then the left one).

- **Deadlock Avoidance**. These strategies constrain the system evolution so that only *safe* states are reached. A reachable state is said to be safe if, once it has been reached, it can be ensured that all the active processes can terminate.

  An avoidance policy must be able to know whether a state is safe or not (or, at least, to be able to select a subset of safe states). Usually a deadlock avoidance strategy runs as follows: when a state change is possible, the controller checks for the safeness of that state. If it is safe, the system transition is allowed; if not, it is forbidden. Therefore, avoidance control policies require the on–line checking for the safeness of a given state, which implies that very efficient algorithms are needed. The most classical example for this approach is the well–known Banker's algorithm [Dij65].

In this work we are going to concentrate on the prevention and avoidance approaches. Their main differences can be abstracted as follows. Initially, we have a set of processes that share a set of system resources. In order to execute a process action, two conditions must hold:

- First, the necessary resources must be available.

- Second, the state reached if the action is executed must be neither a deadlock nor a state leading in an inevitable way to a future deadlock.

In the case of a prevention approach the control necessary to ensure the good behavior has been, in some way, embedded in the system structure, so that instead of executing the original processes a set of transformed ones are executed. The way the processes have been transformed ensures that as soon as the needed resources are available, a process action can be executed, because no system deadlock can be reached in the modified system.

In the class of systems we are going to concentrate on in this work each process state will be explicitly modeled. A deadlock state corresponds to a given tuple of process states. A way to prevent such state would consist in establishing a generalized mutual exclusion on the involved processes so that the tuple is not reachable. From the model perspective, these mutual exclusions can be implemented as a kind of **logical resources**, whose integration in the model will be straightforward (if we are able to model physical resources, we are also able to model logical ones) obtaining the model of the controlled system. If the control is carried out in this way, the **prevention** approach has been adopted.

In the case of an **avoidance** approach, an external decision procedure has to be launched each time a resource related action has to be executed. This procedure will allow the action only if it is sure that no future deadlock problems can arise.

1.3 Flexible manufacturing systems

Flexible manufacturing systems (FMS) are part of an interesting class of concurrent systems. They are used to organize production systems in such a way that they can be quickly adapted to new customer demands. In this work we are mainly concerned with the control of such systems (to avoid deadlock problems). Let us introduce in this section their main features.

A flexible manufacturing system is an automatically controlled set of machines, material handling, and storage facilities that can process simultaneously a set of different types of products.

Usually, a FMS has two main subsystems [VN92], as depicted in Figure 1.2:

- The **physical subsystem**, composed of the physical elements (hardware components) such as transport facilities (conveyors, robots, pallets, automated guided vehicles –AGVs–, etc.); processing machines (work stations, tools,

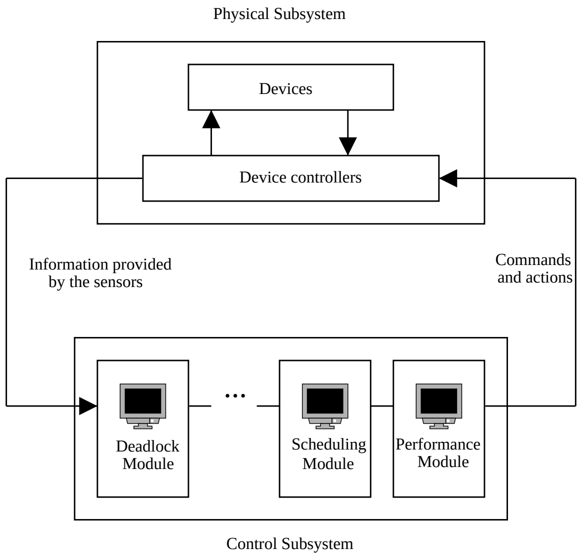


Figure 1.2: The architecture of an FMS

automated inspection systems,etc.) storage places (intermediate stores, machine buffers, etc.); it also includes other control related hardware components such as sensors, and a local area network interconnecting all of the physical elements.

- The **control subsystem**, which manages and controls how the elements in the physical subsystem must be used and interact in order to organize and optimize the production process. It makes possible the automated operation of the entire production system. The control system is typically composed of an interconnected network of programmable controllers, cell controllers, and the supervisory computers. Each one of these individual controllers need to communicate with other controllers in the system. The software resident in the controllers has the capability to enable automated operation as well as system monitoring and diagnostics.

The main characteristics of a manufacturing system are:

- **Flexibility**. This parameter has been conventionally associated to the system ability to process a variety of part types, to carry out a variety of different tasks, and to be easily adapted to produce new types of goods.

- **Automation**. The system must be computer controlled: the main objective for the controller is to accomplish a variety of jobs using the system available devices in an efficient way.

Flexibility is related to many parameters: machine flexibility (possibility of a machine to execute different operations), routing flexibility (possibility of parts to follow different processing routes), ... (see, for example, [BDR$^+$84, SR95] for a more complete list of flexibility parameters).

As previously stated, in a manufacturing system a set of parts are processed at a given time. These in–process parts must cooperate (in the case of assembly systems, for instance) but must also compete for the system resources. These properties give to FMSs a concurrent nature.

A second characteristic of such systems is complexity [Can98]. Many hardware and software components must be monitored and controlled.

These two features, together with the diversity, specificity, and difficulty to manage make necessary the use of formal models [ST97]. Among the wide set of problems related to manufacturing systems the present work concentrates on deadlock problems related to the use of shared resources.

1.3.1 A classification of systems

Flexible manufacturing systems are usually configured as a set of processes (activities, parts, jobs, ...) requesting different quantities of resources (machines, tools, buffer space, ...). When the attention focuses on the use of system resources, an FMS can be seen as a special class of concurrent systems called **resource allocation systems** (RAS) [PS85]. A RAS is composed of a finite set of processes that share in a competitive way a finite set of resources. In a system there can be resources of several types, and for each type there can be several available copies.

The processing of more than one part in a FMS has, as previously stated, a concurrent nature, where the set of processes is composed of the set of parts being processed in the system at a given moment. If we pay attention to the processing of a unique product, it can have either a sequential structure (which corresponds to the case of a raw material which suffers successive transformations until its final state) or a concurrent nature (this is usually due some assembly/disassembly operations that introduce the possibility of independent processing steps of different part components).

RAS where all the products have a sequential structure are named as *Sequential RAS* (S–RAS), while RAS where at least one product has a non–sequential nature are named as *Non Sequential RAS* (NS–RAS). Obviously, S–RAS is a strict subset of NS–RAS.

In the case of S–RAS, the set of states in which an in–process part can stay can be modeled by means of a state machine. In the case of NS–RAS, more complicated models are needed to represent the processing of the involved components.

As a matter of fact, while a lot of work related to S–RAS can be found in the literature, it is much more difficult to find solutions for NS–RAS systems.

In this work we will concentrate on a wide range of **sequential resource allocation systems**. Then, as stated above, the set of reachable states for a part can be modeled by means of a state machine: that is, a sequence of steps representing different operations to be applied to the raw part in order to get a finished product. In each one of these processing steps different system elements can be needed in order to carry out the operation: storage capacity, tools, machines, robots, drills, etc.

From the point of view of deadlock prevention/avoidance strategies, the most important information related to a given resource is its capacity/availability, and it can be defined as the number of process components that can be simultaneously using it (or the physical space size they need in the processing facility, if it is relevant). In the case of a buffer, the capacity term seems the more adequate one, and refers to the maximal number of parts it can contain. In the case of a tool, the term availability seems to be more correct: the tool can be busy (non available) or free (available). However, in the case of several identical tools (or tools with multi–processing capabilities), they can be grouped and seen as a tool with a higher degree of availability: more than one process can be using this (grouped) tool at a given moment. Since from the deadlock point of view both types of resources behave in the same way, the term ‘resource capacity’ will be used for both.

Solving the deadlock problem, even for S–RAS is a very complicated task. Of course, it is always possible to solve it by means of strategies based on the whole set of reachable states. However, this perspective cannot be applied in many (most) cases since the number of states of an even small system can be very big, being impossible to manage them (the *state explosion problem* [CGL94, RW89]). Therefore, we are going to adopt a structural point of view: the models we are going to consider are composed of the models of each one of the involved processes (each type of process is modeled independently of the rest of processes) and the models of the system resources. Adopting this point of view, the solution of the deadlock problem becomes more complicated, qualitatively speaking. Due to these facts, researches have not attacked the problem for the general class of S–RAS, but for strict subclasses. Each subclass is defined by some constraints imposed to the

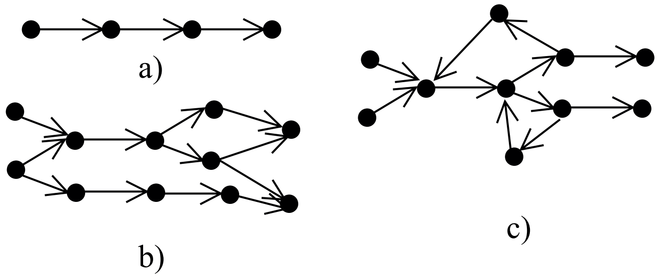


Figure 1.3: Skeletons sketching different on–line routings

more general class of S–RAS, focusing on two main aspects: constrains related to the **routing of parts**, that is, how are the paths a part can follow across the system; and constraints related to the **number and types of resources** allowed to be used in each processing step.

About the routing of parts

The main constraint about the process structure (when considering sequential processes) is related to the availability of different routings: that is, if a part can follow different routings once loaded into the system or, on the contrary, the controller must establish the complete processing route before entering the system (See, for example [BK90, VNJ90, FMMT97, HC94, XHC96]). However, not much work has been done allowing real–time decisions for the part routing (for example, in [BCZ97, ECM95, TCE00, TGVCE00, PR01]). Predetermined processing path approaches usually have the advantage of improving response time; however, flexibility is reduced. In consequence, according to this point of view, resource allocation systems can be classified into the following categories considering which constraints are imposed over the set of states in which a part can stay during its processing. It is important to point out that S–RAS have "precedence" relations (we use the same term as in [GK90]) among the operations to be carried out on the parts. Then, the set of states a part can visit can be modeled by means of a directed graph. The constraints imposed to the graph structure allow us to establish the following classification for S–RAS:

- **Totally Ordered RAS** (TO–RAS): the graph modeling the set of states (processing steps) for a part is totally ordered, as sketched in Figure 1.3(a). These systems do not allow flexible routing at all. The exact route a part can follow during its processing is fully defined by the controller before the part is loaded into the system.

- **Partially Ordered RAS** (PO–RAS): the graph modeling the set of states for a part is partially ordered, as sketched in Figure 1.3(b). The controller will have to take some on–line decisions, according to the system state and the scheduling policies.
- **Non–Ordered RAS** (NO–RAS): the graph modeling the set of states for a part is not ordered (that is, it can have inner cycles), as sketched in Figure 1.3(c).

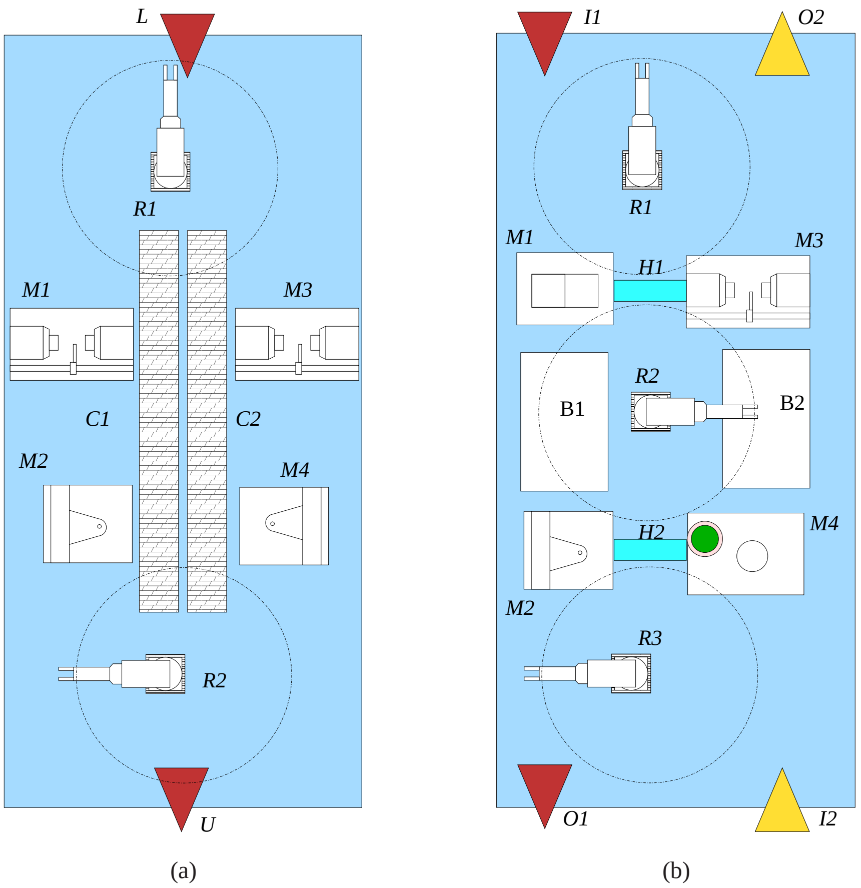
  Since each graph node represents a different state, this means that it is possible for a part to return back to a state previously reached. Part recirculation is allowed. This class of inner loops in the system correspond to systems with some kind of circular transport system, such as a carousel or intermediate buffering devices where the parts can temporarily stay between two processing steps.

It is clear that TO–RAS⊂PO–RAS⊂NO–RAS.

Example 1 *Let us consider a FMS whose layout is shown in Figure 1.4(a). In it, parts arrive at the loading station L and once processed are unloaded via the unloading station U. Machines $M1$ and $M3$ are identical; $M2$ and $M4$ are also identical. Conveyor $C1$ is used to load and unload $M1$ and $M2$, while conveyor $C2$ is used to load and unload $M3$ and $M4$. There are two types of parts. The first type of parts must be produced in such a way that either $M3$, and then $M4$ or $M1$ and then, $M2$ have to be visited. The system, once a part is loaded in $R1$, decides to execute any of the two possible production sequences, $M3$;$M4$ or $M1$;$M2$, depending on the load of the machines. There is a second type of part, which must be processed in $M1$ and then unloaded from the system.* □

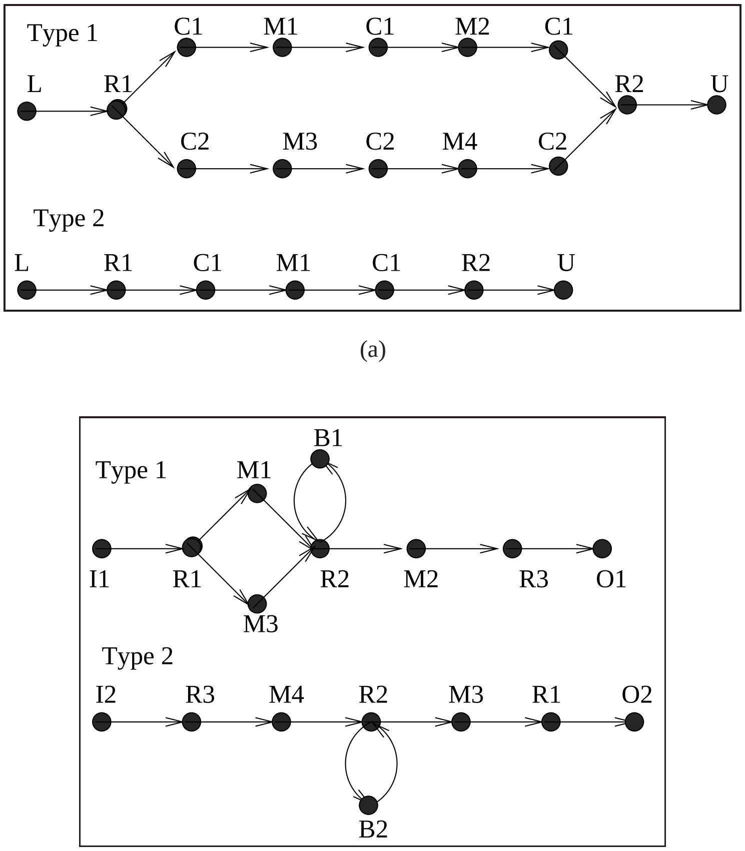
The system of this example is clearly a PO-RAS, and the routes available for these types of parts can be seen in Figure 1.5(a).

Example 2 *Figure 1.4(b) shows the layout of another production cell, where two types of parts have to be processed. The cell is composed of four machines, $M1$, $M2$, $M3$, $M4$; each machine has a processing capacity of two parts. The cell also contains two tool stores, $H1$ and $H2$; the first one contains two classes of tools, $h1$ and $h3$, which have to be shared by $M1$ and $M3$. There are two copies of each*



(a) (b)

Figure 1.4: Layout of two production systems



(a)

(b)

Figure 1.5: Skeleton of the different routings for the types of parts to be processed in the cells depicted in Figure 1.4

one of such tools. Machine $M3$ uses one copy of each tool for the processing of each part, while machine $M1$ uses only one copy of $h1$. $H2$ contains two copies of $h2$ tool and two copies of $h4$ tool. Machines $M2$ and $M4$ use one $h2$ tool and one $h4$ tool for the processing of each part. In order to move parts between the cell components there are three robots, $R1$,$R2$,$R3$. Robot $R1$ loads machines $M1$ and $M3$ from $I1$, and unloads machine $M3$ towards point $O2$. Robot $R2$ loads and unloads the four machines, and can also interact with the intermediate buffers $B1$ and $B2$. These buffers are used for the intermediate storage of parts of type 1 and type 2, respectively, whose processing has not finished yet. Each one of these buffers has capacity for simultaneously storing a maximum of four parts. Finally, the cell also contains a robot $R3$, which can load parts into machine $M4$ from $I2$ and unload parts from $M2$ to $O1$. Parts of type 1 are taken from a conveyor at point $I1$, processed in machine $M1$ or $M3$, then in machine $M2$ and finally unloaded on a conveyor at point $O1$. Parts of type 2 are first loaded into the system from a conveyor at point $I2$, then processed in machine $M4$ and machine $M3$ and finally unloaded to another conveyor at point $O2$. □

The system of this example is clearly a NO-RAS. The possible routes for these types of parts can be seen in Figure 1.5(b).

About the use of resources

The main constraint related to resources refers to the number and type of resources a process can use at a given state. According to this point of view, resource allocation systems can be classified into the following categories [LRF98a]:

- **Single Unit RAS** (SU–RAS): at each processing step, a part requires a single unit from a single resource type (just one unit of buffering capacity of the resource holding the part).

- **Single Type RAS** (ST–RAS): at each processing step, a part can use several units of a single resource type. This allows to model the use of different buffering capacities that different parts can need, and also the different buffering capacities that jobs organized in batches can need, depending on the size of the batch.

- **Multiple Type RAS** (MT–RAS): at each processing step, a part can use several units of several types of resources (the buffering capacity used by the part plus a set of tools, for instance).

It is clear that SU–RAS⊂ST–RAS⊂MT–RAS. Most previous work concentrate on the SU–RAS, however, this constraint was removed, for example, in [BCZ97, TCE99, TGVCE00].

For the MT–RAS there have been partial approaches, where the resources must be requested one copy at a time [TGVCE98, JXH00, HJW02, GS02], until the needed buffering capacity is reached. Another approach is the MT–single unit–RAS (just one copy of several types of resources allowed) as can be seen in [CX97].

We will comment more on some of them to compare with our approaches, when needed.

- A lot of work can be found for the SU–TO–RAS ([BK90, VNJ90], [HC94, FMMT97, XHC96, EGVC98b]).

- A solution for the MT–TO–RAS can be found in [RR92b].

- For a subclass of the SU–PO–RAS class, where a special case of routing flexibility is allowed (each operation can be carried out in a set of different resources), solutions can be found in [Rev98, Rev99, Law00]. The flexibility is reduced because these methods need that a part can flow between any pair of resources that can be used in two consecutive steps.

- In [GK90, Lan99, TCE00] different solutions for the MT–PO–RAS can be found.

- Solutions for the MT–NO–RAS can be found in Banker's like approaches. For example [KTJK97, TCE00].

In this work, we are going to present different solutions for the deadlock problem for MT–PO–RAS (Chapter 3) and for the MT–NO–RAS (Chapter 4).

The previous review has been constrained to the case of S–RAS, and less attention has been paid to the case of NS–RAS. Nevertheless, let us note here that there are some alternative approaches, exploring the problem different models. Some examples can be found in [RR92a, FTM99, JXH00].

1.3.2 Petri nets and other formalisms to model and control FMS

In previous sections we have seen FMS's as complex systems. To deal with this complexity, and given the special characteristics of the type of systems we are considering, the use of formal methods is highly desirable. Formal methods improve the understanding of the system, giving tools for the analysis and implementation steps. They also help in the dialog between the different people related to the design, construction and system management [ST97].

Although the processing of parts in machines and the transport across different handling devices can be continuous processes, the system can be seen as a Discrete Event System (DES) when we concentrate on the use of resources: we have to consider the events related to the allocation/deallocation of resources (or the events of sending/receiving messages) which occur in a discrete way.

Different formalisms have been used to deal with the modeling and control of flexible manufacturing systems (and concurrent systems in general.) Some examples of models used with similar objectives are:

- Models based on formal languages [RW87];
- Models based on finite state automata [RF96];
- Models based on temporal logic [Ost89];
- Finitely recursive processes [IV88], which are based on Hoare's communicating sequential processes [Hoa85];
- Graph theoretic tools [CKW95, FMT00, Law99];
- Finally, the approach used here, Petri nets, that have been widely used (see, for example, [Giu96, HKG97, ST96], for some survey papers. For some recent work see, for example, [ECM95, BCZ97, Che00, TCE00, TGVCE00, PR01, Ge03].)

The first three approaches are mainly based on finite state automata: formal language models use automata in order to model discrete event processes; and models based on temporal logic use finite state transition systems plus temporal logic formulae in order to specify and verify some behavioral properties. All of these approaches have the main disadvantage of the state space explosion problem. There have been some approaches to overcome this problem. Some examples are the use of some subclasses of Petri nets to model the system in [Sre00], the use of a modular and decentralized control [RW92], or the application of a modular and algebraic manipulation for component interaction [XHD99].

The graph theoretic tools have also the same problems of lack of modularity and state space explosion.

Finitely recursive processes allow to model the system as a set of recursive equations. However, as stated in [ZD93a], it is not clear how to use them to design supervisory controllers for real–time systems.

We are going to use Petri nets in order to model and control the systems considered here. Petri nets are effective for modeling DESs and FMSs for the following reasons [SV89, ZD93b, ST97]:

- Easy representation of concurrency, resource sharing, conflicts, mutual exclusions, and non–determinism.
- Availability of different levels of abstraction, allowing to adopt different classes of Petri nets at different phases of the production process.
- A well–defined semantics, which allows the system validation and property verification by means of the model analysis.
- The possibility of code generation from the Petri net model in order to get a prototype of the control program.
- A nice and intuitive graphical representation, which in some cases can be a great help for the people involved in the modeling of the system.

In consequence Petri nets can be used in all aspects of the design and operation of a FMS: modeling and verification, performance analysis, scheduling, control and monitoring, implementation, etc.

The study of a general concurrent system is a difficult task because of the variety of situations that can appear. Fortunately, the class of systems that we are considering will be modeled with special subclasses of Petri nets, which will be introduced and studied in the following chapters. The special characteristics of such classes of nets will allow us to obtain very useful system behavioral properties which will be used to control the system. These properties will be obtained in a structural way; that is, using the structure of the model instead of the set of reachable states, avoiding the state explosion problem.

1.3.3 Characterizing deadlock problems

When trying to eliminate deadlock problems, it is very important to be able to characterize what causes these problems. Let us concentrate now on the most usual methods used to study deadlock problems in concurrent systems. These methods depend on the model used to represent the system, and most of them take advantage of some structural limitations imposed to the way processes can interact. Anyway, in some cases models take advantage of methods proposed in other frameworks. The different methods used to deal with deadlock problems can be classified as follows:

- **Methods based on the reachability set**: They construct, either in a total or partial way, the set of states the system can reach. This makes possible to exactly know which are the undesirable markings and, in consequence, avoid them. This approach can give, in general, more accurate results. However, it is very expensive and, in some cases, unaffordable due to the size of the reachability set.

  Most of the *supervisory*–based work [RW89], some Petri nets–based work, and other studies using model checking, use the total or partial construction of the reachability graph [VNJ90, CKW95, BLP96, CG96, OH00, Giu96, RJ96, LRF97, BCG98, XHD99, MBSD99, QJ99, LMB97, Sre00, YB00].

- **Methods based on structural characterizations**: Several approaches try to capture the *hold and wait* situations using the structure of the involved processes:

  - The first approach, which will be called **based on cycles**, looks for cycles of resources that can reach a hold and wait situation. Some of the work following this line lack of a complete characterization of the problem; cycles of resources do not completely represent all the *hold and wait* situations than can appear. Some work following this approach can be found in [RYJ91, FNTS94, JD95, HSBM96, FMMT97].

  - The second approach, which will be called **based on siphons**, is related to some structural components of the Petri net model, called siphons. The previous approach can be considered as a partial version of the siphon based methods. Although in some cases ([EGVC98a]) cycles and siphons are equivalent, the approaches based on siphons can deal with more general classes of systems. There exist also some algebraic–based approaches [PR00b, PR00a, Law00] that, in most cases, are closely related to siphons, and will be considered as siphon–based.

    Some work following this approach can be found in [ECM95, BPP96, CX97, Jen96, AE98, LRF98b, LGB$^+$98, XJ99, Che00, MV00, PR00a, TGVCE00].

- **Look–ahead** based methods. These methods are based on some knowledge about the future needs of resources of each active process. These approaches do not focus on the set of 'bad states'; the goal is to keep the system in 'safe' states. That is, given a state that is known to be 'good', the control policy will only allow the system evolution into another 'good' state. The problem for these methods is to establish whether a state is safe or not. For this, several approaches have been considered in the literature, being the main difference the knowledge about the future states used to define a state as ‘safe’:

- an estimation of the future maximal needs of resources used in the original Banker’s proposal [Dij65, Hab69, SPG91],
- information about the resources needed to finish the processing in a *zone* (zones are usually defined as subsequences of the available processing sequences. These zones are selected in such a way that at the beginning and the end of each zone, the use of resources does not interfere with the use of resources of other processes.) In this way, if the system can be partitioned taking advantage of intermediate points, better policies can be obtained [BK90, RR92b, EH93, RF96, Lan99].
- detailed information about where and when resources will be needed, used in [TCE00, Rev98].
- finally, some approaches use a partial look–ahead policy [VNJ90]. A number of steps is fixed, and before allowing any system evolution, they simulate the advance of these steps. If they do not find problems, the system is allowed to evolve. As they only look forward a fixed number of steps, these control policies need to have a deadlock detection algorithm because the absence of problems in a fixed number of steps does not guarantee the absence of problems in further steps.

  Of course, it is always possible to do a complete look-ahead checking, trying to see if there are system evolutions from a given state that allow to finish all the active processes [YB00]. Clearly, this approach is very time consuming since in some cases it can be equivalent to compute the whole reachability graph.

1.4 Work Outline

Chapter 2 is devoted to the introduction of a new class of nets, named S^4PR, that will be used for the modeling of flexible manufacturing systems. The class is introduced in a compositional way, which allows a simple and useful model construction. The class of models considered is able to deal with the multiple type, partially ordered resource allocation systems (MT–PO–RAS.) For this class, a liveness characterization will be established, followed by some results that will be applicable for deadlock prevention. It will be presented in three different forms: the first one is a characterization of the deadlock problem in terms of the marking of some places related to the set of transitions directly involved in a deadlock; the second and third characterizations show how liveness problems are related to the existence of a special kind of siphons, which will be used to select some ‘representative’ markings in order to prevent deadlock related problems.

In Chapter 3 the siphon–related liveness characterization presented in Chapter 2 is used to prevent deadlocks. The process behind the control method is as follows. The liveness characterization relates siphons and deadlocked markings. The system behavior is represented by the state equation and some integer programming problems that allow to obtain a (potential) deadlocked marking. We introduce a way for preventing such markings by means of the addition to the some new place which makes the considered marking unreachable. It is then proved that the behavior of the added place is like a virtual resource, and it is then concluded that the net controlled in this way is a S^4PR. Therefore, the method can be iterated. It is also proven that the algorithm terminates, obtaining a final controlled S^4PR which is live and whose language is a subset of the language of the original system.

Chapter 4 concentrates on deadlock problems, but from an avoidance point of view, and using the ideas behind the well–known Banker’s algorithm proposed by Dijkstra [Dij65]. The avoidance approach we propose is able to deal with the multiple–type, non–ordered resource allocation systems (NO-RAS. In order to get a better understanding of the method, a general framework and an algorithm based on this general framework are presented. The framework is based on the definition and parametrisation of a set of functions, used to establish a bound of the future needs of resources for each active process. This framework allows us to present and study several solutions for the problem as special cases of the general case. Some particular solutions, together with the study of their runtime costs, are proposed, being one of them the classical Banker’s algorithm.

Since some of the proposed methods for the deadlock prevention rely on the computation of sets of siphons ([TCE99, IMA02]), some research has been done to obtain better solutions for this problem. These results are shown in Chapter 5. We will show a review of the available methods, trying to establish a classification for them. Later, one of these methods will be selected, having in mind the kind of problem we need to solve (siphons that contain resource places in S^4PR nets), and the way we expect to reach the improvements (parallel computing). Finally, some numeric computations have been done in order to test the proposed approach and to compare it with some recently proposed efficient solutions.

Chapter 2

The S^4PR class: definition and properties

Abstract
A new class of systems is going to be presented. This class, named S^4PR, is adequate for the modeling of a wide variety of RAS. The special syntactic characteristics of the nets of this new class make possible the study of the modeled systems from a structural perspective. The chapter introduces the class, first by means of an an example, and then in a formal way. The main structural properties of the nets belonging to this class are studied. Finally, a characterization of deadlock situations and several re–formulations in terms of siphons are presented. These characterizations will be used in the next chapter in order to control the system to prevent deadlock problems.

2.1 Introduction

As stated in the introductory chapter, the variety of elements involved in a typical FMS makes necessary the use of some formalism in order to manage this complexity and to improve the understanding of the system. We are going to use Petri nets to model and control the systems considered here. In the previous chapter we made a classification of the systems based on two main characteristics: the routing of parts and the restrictions on the use of resources. The class that will be presented later allows to model on–line routing flexibility (that is, a part can follow different paths in the system, and the election can be done the processing) and free–conservative use of resources. According to the classification presented in Section 1.3.1 they belong to the MT–PO–RAS family.

The S^4PR class will be presented following a constructive, process–oriented approach: first, the building blocks will be shown, and then, the way they can be put together in order to model a system. Some interesting properties of the class will be studied, being the characterization of deadlock problems one of the main results of the chapter.

2.1.1 A Class of Nets for Production Systems

Let us introduce by means of an example a S^4PR net. Figure 2.1 sketches a production cell composed of four machines, $M1$, $M2$, $M3$, $M4$, whose processing capacity is of two parts each at a time. They are able to carry out different operations aided with the tools available in two tool stores, $H1$ and $H2$; the first one contains two classes of tools, $h1$ and $h3$, which have to be shared by $M1$ and $M3$. There are two copies of each one of these tools. Machine $M3$ uses one copy of each tool for the processing of a part, while machine $M1$ uses one copy of $h1$ for the processing of each part. $H2$ contains two copies of $h2$ tool and two copies of $h4$ tool. Machines $M2$ and $M4$ use one $h2$ tool and one $h4$ tool for the processing of each part.

In order to transport the parts along the cell there are three robots, $R1$,$R2$,$R3$. Robot $R1$ loads machines $M1$ and $M3$ from point $I1$, and unloads machine $M3$ towards point $O2$. Robot $R2$ moves parts between the four machines. Finally, the cell contains a third robot, $R3$, which can load parts into machine $M4$ from point $I2$ and unload parts from $M2$ to $O1$.

In this cell two different types of parts have to be processed, according to two different working plans. Parts following the working plan $WP1$ are taken from a conveyor at point $I1$, processed in machine $M1$ or $M3$, then in machine $M2$ and finally unloaded on a conveyor at point $O1$. Parts following the working plan $WP2$ are first loaded into the system from a conveyor at point $I2$, then processed in machine $M4$, then in machine $M3$ and finally unloaded to another conveyor at point $O2$.

A usual and natural way to model this kind of systems is based on the use of finite state automata [EH93, CKW95, Law99]. These automata represent the flow relations in the system: that is, they show the steps that a part can follow across the system in order to be processed.

Let us concentrate on the processing of a part following the working plan

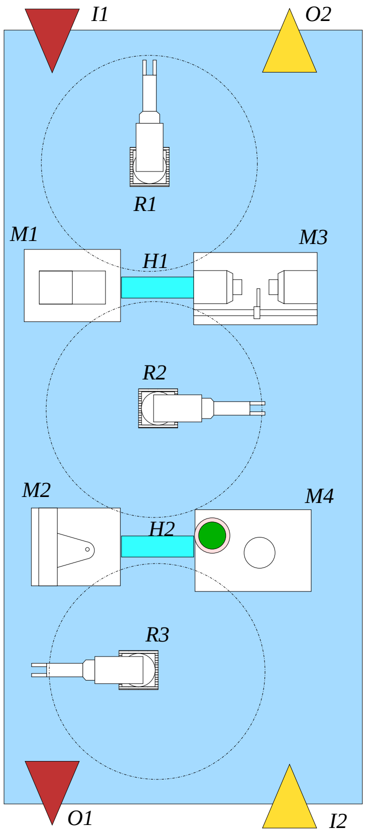


Figure 2.1: Layout of a manufacturing cell

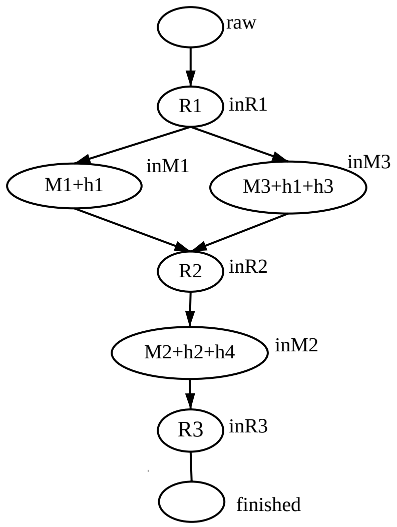


Figure 2.2: An automaton modeling the processing of a part following the processing plan $WP1$ in the cell in Figure 2.1

$WP1$, and let us see the states in which this part can stay. The automaton in Figure 2.2 represents the set of these states, with the following meaning:

- The initial state, raw, corresponds to the part being out of the system, waiting to be processed.
- $inR1$ represents the state in which the raw material is being loaded into the system by means of robot $R1$.
- $inM1$ and $inM3$ represent the states in which the part is being processed at machine $M1$ or $M3$. Notice that the structure of the graph for these two states reflects the availability of two alternatives for this processing step.
- $inR2$ represents the state in which the part is being moved by the robot $R2$ from $M1$ or $M3$ to $M2$.
- $inM2$ represents the state in which the part is being processed by machine $M2$.
- $inR3$ represents the state in which the finished part is being unloaded from the machine $M2$ by robot $R3$.

- Finally, $finished$ corresponds to the state in which the part processing has been terminated and it is out of the system.

In this model, each one of the available paths from the initial state (raw) to the final one ($finished$) corresponds to a possible production sequence for the considered part.

In order to have a more complete model, and taking into account that the considered part uses a different set of resources at each state, each node can be labeled with the multiset of resorces used by parts at the state represented by the node, as shown in Figure 2.2. A state change during the processing of a part (a transition in the automaton) implies that some new resources are engaged and some resources are released. Let us consider, for instance, transition from $inR2$ to $inM2$ in Figure 2.2. When the part is at state $inR2$, robot $R2$ is engaged by the part. The transition to the state $inM2$ needs that the resources $M2$, $h2$, and $h4$ are available. Moreover, when the part moves to machine $M2$ and reaches the state $inM2$, resource $R2$ becomes available. The automaton does not properly represent which is the state of the system resources to know whether transition to $inM2$ is possible when the part is at state $inR2$. In order to complete the model, the state of all the system resources should be represented. In this sense, it is important to identify the relevant aspects of resources for us. Since we are going to concentrate on deadlock problems, and resource related deadlock problems depend on the availability of resources at any given moment (as stated in the introductory chapter), the availability of system resources is the information needed.

Figure 2.3 shows a Petri net model corresponding to the processing of the considered part. In this model there are two kind of places:

- Places corresponding to the original nodes of the automaton and related transitions. They have the same meaning as in the original automaton.
- Places modeling resources. Each one of these places will have an initial marking equal to the capacity of the resource it models. Outgoing arcs represent resource engagement, while incoming arcs represent resource releasing.

In this way, each firing sequence of the Petri net carrying the token from raw to $finished$ corresponds to a processing sequence. The model of Figure 2.3 corresponds to the processing of a unique part following the working plan $WP1$. What about the case in which several WP1–parts must be concurrently processed? This can be easily considered in the model by means of the marking of place raw: putting k tokens in place raw as initial marking the concurrent processing of up to

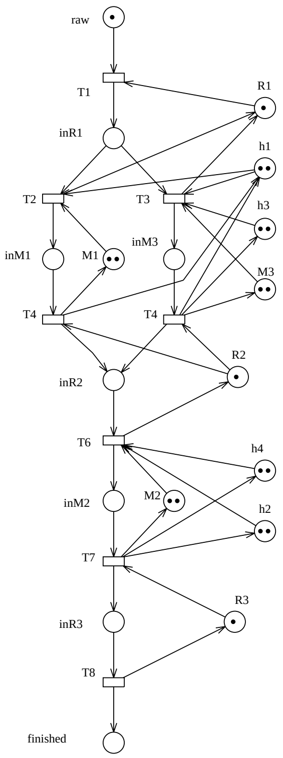


Figure 2.3: A Petri net model for the processing of WP1–parts in the cell of Figure 2.1

k WP1–parts is allowed. However, this is not the usual approach for the modeling of these systems: the concurrent processing of as many parts as needed (limited only by the resource capacities) has to be modeled. This can be modeled substituting places raw and $finished$ by just one place, and modeling each type of part as a kind of cyclic process as in Figure 2.4. This model is what we will call a *process Petri net*. There will be one of these process Petri nets for each type of part to be produced in the system.

Before presenting a formal definition of process Petri nets, let us comment on the main features of the proposed model and some alternative approaches that can be found on the literature. Remember that at each step a process needs a number of resources in order to accomplish the corresponding task. If one or more of these resources are not available, the step cannot be accomplished. Most of the restrictions related to the use of resources of previously defined classes used to deal with the problem of deadlock prevention for similar systems have been suppressed, and for each processing step it is allowed:

- the use of any number of copies of any resource, and
- the use of as many different resources as needed.

The only remaining restriction is related to the modeling of serially reusable resources, as usually referenced in the literature when talking about resources that cannot be created nor destroyed.

According to the classification established in the introductory chapter, and the sketched features of the class, the systems that will be modeled using this new class can be classified as MT–PO–RAS.

The idea of process Petri net is related to concepts introduced in previous work:

- *Free sequential systems*, presented in [LT79]. They were constrained to the family of TO–MT–RAS.
- *Production sequence models* presented in [BK90] allow the representation of a set of sequential processing steps. The authors also provide the way to model resource usage by means of the addition of resource places to these production sequences, composing what they call *process Petri nets*. With these nets they are able to represent TO–SU–RAS with just one type of process.

  We feel that the idea behind this term is adequate for the representation of more general processing structures, and this is the reason for using the same name for our more general class of systems.
- *Job subnet*, presented in [HC92] are used to model TO–SU–RAS. Job subnets do not include the modeling of the resources because the model is divided in two parts: the job subnets to model the process structure, and the resource subnets which are used to model the use of resources.
- *Simple sequential processes with resources* (S^2PR) presented in [ECM95] that can model PO–SU–RAS.
- *Extended simple sequential processes with resources* (ES^2PR) presented in [TGVCE98] are able to model PO–MT–RAS, with the restriction that

the resources need to be taken one copy at a time until the total amount of needed copies is reached.

2.2 The Class of S^4PR Nets

S^4PR nets will be used to model the concurrent processing of a set of parts of different types. All the parts of the same type have the same processing possibilities. The whole model will be obtained by means of the composition of the process Petri net modeling the processing of the different types of parts.

2.2.1 Modeling processes: process Petri nets

Let us remember, before defining it, that the class of S–RAS allows flexible routing (which means that on–line, real–time routing decisions can be modeled) and the use of any number of reusable, non–consumable resources at each state.

Definition 1 *A* process Petri net *is a generalized strongly connected self–loop free Petri net $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ where:*

1. *P is a partition as follows: $P = \{p_0\} \cup P_S \cup P_R$.*
2. *The subnet generated by $\{p_0\} \cup P_S \cup T$, $\mathcal{N}_{|(\{p_0\} \cup P_S, T)}$, is a strongly connected state machine such that every cycle contains p_0.*
3. *$\forall r \in P_R$, there exists a unique minimal P–Semiflow $\mathbf{Y_r} \in \mathbb{N}^{|P|}$ such that $\{r\} = \|\mathbf{Y_r}\| \cap P_R$, $\{p_0\} \cap \|\mathbf{Y_r}\| = \emptyset$, $P_S \cap \|\mathbf{Y_r}\| \neq \emptyset$ and $\mathbf{Y_r}[r] = 1$.*
4. *$P_S = \bigcup_{r \in P_R} (\|\mathbf{Y_r}\| \setminus \{r\})$.*

□

From the application point of view, a process Petri net will be used to model the processing of a type of part.

Place p_0 is the *idle state place* (or idle place); we will use $P_0 = \{p_0\}$. It models a raw part before entering the system. Places in P_S are the *state places* (or process places) and model the states for a part of the considered type. Transitions of T model the state changes. A change in the state can correspond to two different kinds of events: either the part changes its location in the system (moving from one resource to a different one), or a transformation has been done in the part (as the result of a system operation inside a machine.) Arcs joining places of $P_0 \cup P_S$ with transitions correspond to the state changes in the processing of each part.

Places in P_R are the *resource places* and model the state of the system resources. Arcs joining resource places and transitions of the Petri net model how the state of the resources change when the parts evolve in the system. Arcs related to resource places model how resources are used for the processing of parts: outgoing arcs from resource places to transitions model the acquisition of resources; arcs from transitions to resource places model their release.

Let us do some comments about the three last points of Definition 1:

- Point 2 establishes the structure allowed for the set of states that a part can follow during its processing, as commented before. The fact of imposing that each cycle must contain place p_0 corresponds to the idea of "processing evolution" in the system: if transitions are executed, the processing of a part will eventually terminate. This forbids the existence of parts evolving in a limited subset of states inside the system.

- Point 3 establishes that each resource must be serially reusable (it cannot be created nor destroyed in the system.)

- Point 4 imposes that each processing step requires the use of at least one resource. This represents that when a part is being processed, it must be "somewhere" in the system using, at least, some buffer space of it.

  From a theoretical point of view, this constraint is not needed, and could be withdrawn. The results we are going to present will also be valid if constraint 4 is suppressed: it can be easily seen that for each place $p \in P_S$ not belonging to the support of any $\mathbf{Y_r}$ its complementary place can be added, with an initial marking equal to the one of p_0. These added complementary places are implicit and behave as 'virtual' resources. Moreover, the resulting net is a process Petri net with the same firing sequences as the initial one.

The process Petri net that represents the processing of parts following the working plan *WP1* in the cell depicted in Figure 2.1 is shown in Figure 2.4. There, the following elements can be identified:

- $P_0 = \{P1\_0\}$
- $P_S = \{P1M1, P1M3, P1M4, P1R1, P1R2, P1R3, \}$
- $P_R = \{M1, M3, M4, R1, R2, R3, h1, h2, h3, h4\}$

In order to complete the modeling of the dynamics of a process Petri net, an initial marking must be provided. The tokens in a reachable marking can have different meanings:

- A token in a place $p \in P_S$ will model an active process (a part being processed) whose state is modeled by means of place p (the part is at the state represented by this node.) Several tokens in the same process place will represent several active processes (several parts being processed) whose respective states are modeled by means of place p.

- Tokens in a place $r \in P_R$ will model the available buffering capacity of resource r at the system state modeled by the considered marking (remember that, as previously said, *buffering capacity* will be used to represent either capacity or availability.)

Markings need to represent states that have a physical meaning. In this sense, only *acceptable initial markings*, as defined in the following, will be considered. If the system is well defined, and its initial marking is "correct", all the markings that are reachable from it will represent possible states of the system, and will have a physical meaning.

Definition 2 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ be a process Petri net. An initial marking $\mathbf{m_0}$ is* acceptable *for $\mathcal{N}$ if and only if:*

1. $\mathbf{m_0}[p_0] > 0$*;*

2. $\forall p \in P_S \,.\, \mathbf{m_0}[p] = 0$*;*

3. $\forall p \in P_S \,.\, \forall r \in P_R \,.\, \mathbf{m_0}[r] \geq \mathbf{Y_r}[p]$. □

A process Petri net with its marking will be used to represent the processing of a set of parts of the same type. Let us remark the following facts:

- The initial marking of p_0 (condition (1)) represents the maximal number of parts of the type modeled with this net that are allowed to be concurrently processed in the system. This initial marking can be chosen in such a manner that p_0 becomes implicit [Sil85, CS89], making this kind of systems suitable for the modeling of open systems (the maximal number of parts of the type modeled by the process Petri net concurrently processed is limited by the system itself via the capacities of the system resources.)

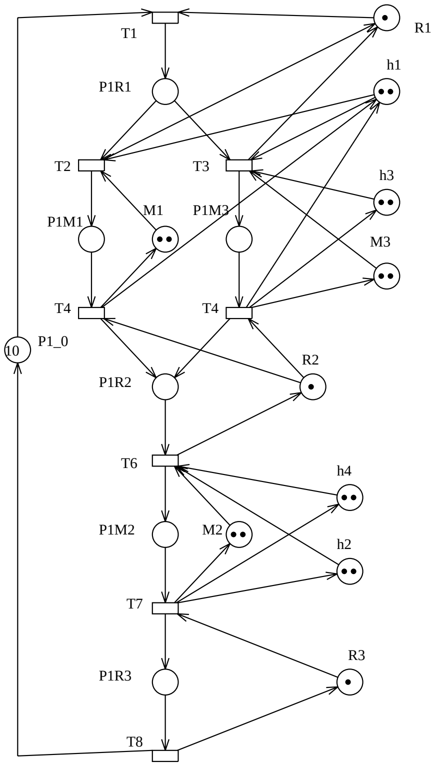


Figure 2.4: The process Petri net model of the system whose layout is shown in Figure 2.1 when the two types of parts to be produced are considered

- No process is active at the initial state (condition (2).)
- The buffering capacity of each resource is such that each processing step can be executed when the isolated execution of one process is considered (condition (3).) This property will be proved later.

Some basic structural properties of process Petri nets

Let us now present some structural properties of process Petri nets relating structure components and its physical meaning.

We are going to show how the minimal P–Semiflows induced by the structure of these nets are, and how can be intrepreted from the application domain point of view. Let us first consider the P–Semiflows related to resources as they appear in Definition 2.

These minimal P–Semiflows induce marking invariant relations of the form $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})\,.\, \mathbf{Y_r} \cdot \mathbf{m} = \mathbf{Y_r} \cdot \mathbf{m_0}$. They can be interpreted in the following way:

1. For every reachable state, the buffering capacity of a resource type, r, is constant and it is equal to the buffering capacity at the initial state: $\mathbf{Y_r} \cdot \mathbf{m_0} = \mathbf{m_0}[r]$. Notice that this property establishes an important feature of the class of systems considered: resources are re–usable. Re–usability implies that the utilization of the resources by the processes does not change them. Then, the buffering capacity of each resource is invariant.

2. At a reachable marking $\mathbf{m}$, the initial capacity of a resource r is distributed in the following way: $\mathbf{m}[r]$ is the capacity of r available at $\mathbf{m}$; for any $p \in P_S$, $\mathbf{Y_r}[p]$ is the buffering capacity of resource r used by a process at state p, and $\mathbf{m}[p] \cdot \mathbf{Y_r}[p]$ is the capacity of resource r used by processes at p.

Considering resource $M3$ in Figure 2.4, the associated P–Semiflow is $\mathbf{Y_{M3}} = M3 + P1M3$, which induces the invariant relation $\mathbf{m}[M3] + \mathbf{m}[P1M3] = \mathbf{m_0}[M3] = 2$ for every reachable marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$. In consequence, one of the following expressions is true:

- $\mathbf{m}[M3] = 0 \ \wedge \ \mathbf{m}[P1M3] = 2$
- $\mathbf{m}[M3] = 1 \ \wedge \ \mathbf{m}[P1M3] = 1$
- $\mathbf{m}[M3] = 2 \ \wedge \ \mathbf{m}[P1M3] = 0$

which means that $M3$ can be processing two, one, or zero parts, respectively. When $\mathbf{m}[M3] = 0$, two parts are being processed at machine $M3$, modeled by the tokens allocated in $P1M3$ ($\mathbf{m}[P1M3] = 2$.)

For a given resource, r, and based on the minimal P–Semiflow $\mathbf{Y_r}$, the *holders of resource* r are going to be introduced as the set of process places using this resource.

Definition 3 *Let* $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ *be a process Petri net. Let* $r \in P_R$*. The set of* holders of r *is the support of the minimal P–Semiflow* $\mathbf{Y_r}$ *without the* *place r: $\mathcal{H}_r = \|\mathbf{Y_r}\| \setminus \{r\}$. This definition can be extended in the natural way to sets of resources $A \subseteq P_R$: $\mathcal{H}_A = \bigcup_{r \in A} \mathcal{H}_r$.* □

Why the name “holder”? Let us consider the net in Figure 2.4 and the resource place $M3$. For it, $\mathcal{H}_{M3} = \{P1M3\}$; considering $\mathbf{Y_{M3}} = M3 + P1M3$, each time a token enters place $P1M3$, a token “disappears” from $M3$ (maintaining the invariant relation), i.e., an active process in $P1M3$ is “holding” one capacity unit of the physical resource represented by place $M3$. Notice that the invariant induced by $\mathbf{Y_r}$ also states that $\mathbf{m}[r] = \mathbf{m_0}[r] - \sum_{p \in \mathcal{H}_r} \mathbf{m}[p] \cdot \mathbf{Y_r}[p]$.

In each process Petri net one more P–Semiflow can be identified; it is related to the process structure and its configuration as a state machine.

Proposition 4 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ be a process Petri net.Then, $\mathbf{Y_S} = \mathbf{1}_{(P_0 \cup P_S | P_0 \cup P_S \cup P_R)}$ is a minimal P–Semiflow.*

Proof
Let us show that $\mathbf{Y_S} = \mathbf{1}_{(P_0 \cup P_S | P_0 \cup P_S \cup P_R)}$ is a P–Semiflow. The net $\mathcal{N}_{|(P_0 \cup P_S, T)}$ is a strongly connected state machine and then, it has a unique minimal P–Semiflow whose support is $P_0 \cup P_S$. Moreover, since $\forall r \in P_R \,.\, \mathbf{Y_S}[r] = 0$, $\mathbf{Y_S}$ is a P–Semiflow of $\mathcal{N}$. Furthermore, being a minimal P–Semiflow of $\mathcal{N}_{|(P_0 \cup P_S, T)}$ and being a P–Semiflow of $\mathcal{N}$, it also needs to be minimal in $\mathcal{N}$: if $\mathbf{Y}'$ was a minimal P–Semiflow whose support is strictly included in the support of $\mathbf{Y_S}$, and as far as $\mathbf{Y_S}$ does not contain places of P_R, it also would be a minimal P–Semiflow of the net $\mathcal{N}_{|(P_0 \cup P_S, T)}$, and then $\mathbf{Y}' = \mathbf{Y_S}$. □

This last minimal P–Semiflow induces a marking invariant relations of the form $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) \,.\, \mathbf{Y_S} \cdot \mathbf{m} = \mathbf{Y_S} \cdot \mathbf{m_0}$. This invariant can be interpreted in the following way:

1. For every reachable marking, the number of tokens representing processes of the type modeled by the net is constant and it is equal to the number of tokens in the idle place at the initial state: $\mathbf{Y_S} \cdot \mathbf{m_0} = \mathbf{m_0}[P_0]$. This constrain is related to the idea that a token that represents a part in the system cannot generate several parts or, reversely, disappear.

2. The number of parts of the considered type that can be concurrently processed is bounded by $\mathbf{m_0}[P_0]$. However, as previously commented, this is not a limitation since the initial marking of P_0 can be big enough to allow the modeling of open systems.

Let us consider once again the net in Figure 2.4. The minimal P–Semiflow $\mathbf{Y_{S_1}} = P1\_0 + P1R1 + P1M1 + P1M3 + P1R2 + P1M2 + P1R3$ generates the following invariant relation: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) \,.\, \mathbf{m}[P1\_0] + \mathbf{m}[P1R1] + \mathbf{m}[P1M1] + \mathbf{m}[P1M3] + \mathbf{m}[P1R2] + \mathbf{m}[P1M2] + \mathbf{m}[P1R3] = \mathbf{m_0}[P1\_0]$, imposing that a part following *WP1* in the cell of Figure 2.1 can be either held by a robot (places $P1R1, P1R2, P1R3$) or processed in one of the machines (places $P1M1, P1M2, P1M3$.)

The next lemma presents a result relating rows representing resources in the flow matrix and rows representing the other places of the net. This lemma will be used later on to relate net circuits and T–Semiflows.

Lemma 5 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$ be a process Petri net.*

$$\forall r \in P_R \,.\, \mathbf{C}[r,T] = \sum_{p \in (P_0 \cup P_S \setminus \mathcal{H}_r)} k_r \cdot \mathbf{C}[p,T] + \sum_{p \in \mathcal{H}_r} (k_r - \mathbf{Y_r}[p]) \cdot \mathbf{C}[p,T]$$

where $k_r = \max_{p \in \mathcal{H}_r} \mathbf{Y_r}[p] (> 0)$.

Proof

1. *The net $\mathcal{N}_{|(P_0 \cup P_S, T)}$ is a strongly connected state machine. Then, there exists only one minimal P–Semiflow which establishes the following invariant relation* $\sum_{p \in P_0 \cup P_S} \mathbf{C}[p,T] = 0 = \sum_{p \in (P_0 \cup P_S \setminus \mathcal{H}_r)} \mathbf{C}[p,T] + \sum_{p \in \mathcal{H}_r} \mathbf{C}[p,T]$
2. *By definition of process Petri net,* $\forall r \in P_R \,.\, 0 = \mathbf{C}[r,T] + \sum_{p \in \mathcal{H}_r} \mathbf{Y_r}[p] \cdot \mathbf{C}[p,T]$.
3. *Then:* $\sum_{p \in (P_0 \cup P_S \setminus \mathcal{H}_r)} k_r \cdot \mathbf{C}[p,T] + \sum_{p \in \mathcal{H}_r} k_r \cdot \mathbf{C}[p,T] = 0$. *Then,*
   $\mathbf{C}[r,T] + \sum_{p \in \mathcal{H}_r} \mathbf{Y_r}[p] \cdot \mathbf{C}[p,T] =$
   $= \sum_{p \in (P_0 \cup P_S \setminus \mathcal{H}_r)} k_r \cdot \mathbf{C}[p,T] + \sum_{p \in \mathcal{H}_r} k_r \cdot \mathbf{C}[p,T] = 0$
   and then:
   $\mathbf{C}[r,T] = \sum_{p \in (P_0 \cup P_S \setminus \mathcal{H}_r)} k_r \cdot \mathbf{C}[p,T] + \sum_{p \in \mathcal{H}_r} (k_r - \mathbf{Y_r}[p]) \cdot \mathbf{C}[p,T]$

□

In fact, since $\forall p \in \mathcal{H}_r \,.\, (k_r - \mathbf{Y_r}[p]) \cdot \mathbf{C}[p,T] \geq 0$, this lemma has proved that each resource place is a structural implicit place (SIP) [Sil85, CS89].

Let us now concentrate on another interesting set of Semiflows related to the sequencing of processing states: T–Semiflows. In order to simplify the notation some conventions are going to be used.

Note 6 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$ be a process Petri net.*

- *Let $\mathbf{X}$ be a T–Semiflow of $\mathcal{N}$. $\mathbf{X}$ induces the following sets: $X_T = \|\mathbf{X}\|$, and $X_P = ({}^\bullet X_T \cap (P_0 \cup P_S)) \cup ({X_T}^\bullet \cap (P_0 \cup P_S))$.*
- *Let F be a simple circuit of $\mathcal{N}$. F induces the following sets: $F_P = F \cap (P_0 \cup P_S)$, and $F_T = F \cap T$.* □

For example, in the net in Figure 2.5 the minimal T–Semiflow

$$\mathbf{X_1} = T1 + T3 + T5 + T6 + T7 + T13$$

induces the sets

$$X_{1T} = \{T1, T3, T5, T6, T7, T13\}$$

and

$$X_{1P} = \{P1\_0, P1R1, M1, P1R2, P1M2, P1R3\}.$$

The following lemma shows the relation between minimal T–Semiflows and simple circuits of the embedded state machine corresponding to the complete processing sequences.

Lemma 7 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$ be a process Petri net.*

- *Let F be a circuit of $\mathcal{N}$ not containing places of P_R. F_T induces the minimal T–Semiflow $\mathbf{X_F} = \mathbf{1}_{(F_T|T)}$.*
- *Conversely, let $\mathbf{X}$ be a minimal T–Semiflow of $\mathcal{N}$. $X_T \cup X_P$ generates a simple circuit*

Proof
First of all, since the net $\mathcal{N}_{|(P_0 \cup P_S, T)}$ is a strongly connected state machine, each minimal T–Semiflow of such net induces a simple circuit and vice versa, (because in a strongly connected state machine the set of transitions in a directed simple circuit is a minimal T–Semiflow. See [Mur89], where the property is presented for marked graphs and P–Semiflows.)

Moreover, according to Lemma 5,

$$\begin{aligned}
\forall r \in P_R \,.\, \mathbf{C}[r,T] \cdot \mathbf{1}_{(F_T|T)} &= \textstyle\sum_{p \in P_0 \cup P_S \setminus \mathcal{H}_r} (k \cdot \mathbf{C}[p,T]) \cdot \mathbf{1}_{(F_T|T)} \\
&\quad + \textstyle\sum_{p \in \mathcal{H}_r} ((k - \mathbf{Y_r}[p]) \cdot \mathbf{C}[p,T]) \cdot \mathbf{1}_{(F_T|T)} \\
&= \textstyle\sum_{p \in P_0 \cup P_S \setminus \mathcal{H}_r} k \cdot (\mathbf{C}[p,T] \cdot \mathbf{1}_{(F_T|T)}) \\
&\quad + \textstyle\sum_{p \in \mathcal{H}_r} (k - \mathbf{Y_r}[p]) \cdot (\mathbf{C}[p,T]) \cdot \mathbf{1}_{(F_T|T)}) \\
&= 0
\end{aligned}$$

which implies that it is also a T–Semiflow of $\mathcal{N}$.

Let us prove that minimal T–Semiflows of $\mathcal{N}_{|(P_0 \cup P_S, T)}$ are also minimal in $\mathcal{N}$. Let us consider a minimal T–Semiflow of $\mathcal{N}_{|(P_0 \cup P_S, T)}$, that is not minimal for $\mathcal{N}$. Since $\forall t \in T \,.\, \mathbf{X_F}[t] \in \{0, 1\}$, if $\mathbf{X_F}$ is non minimal, this implies that there exists another T–Semiflow, $\mathbf{X}'$, such that $\forall t \in T \,.\, \mathbf{X}'[t] \in \{0, 1\}$ and $\|\mathbf{X}'\| \subset \|\mathbf{X_F}\|$. Let us consider $t \in \|\mathbf{X_F}\| \setminus \|\mathbf{X}'\|$. Since $\mathbf{X_F}$ induces a simple circuit, ${}^{\bullet\bullet}t \cap \|X_F\| = \{t'\} \notin \|X_F\|$.

This reasoning can be iterated, allowing us to conclude that $\mathbf{X_F}' = \mathbf{0}$. □

Notice that any T–Semiflow $\mathbf{X}$, if fireable, corresponds to a firing sequence $\sigma_\mathbf{X}$ such that $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \sigma_\mathbf{X}$, and then, $\mathbf{m} = \mathbf{m_0}$. From the application point of view, minimal T–Semiflows are related to firing sequences moving a token from P_0 to P_0 following a path in the process Petri net, which corresponds to a complete processing of a part. Any minimal T–Semiflow corresponds to a possible processing sequence. Proposition 11 below shows that any T–Semiflow induces an effective production sequence for parts of the considered type, provided they can be executed in isolation.

For example, in the net in Figure 2.5, the previously presented T–Semiflow $\mathbf{X_1}$ models the complete processing of a *WP1*–part.

Note 8 *If $\mathcal{N}$ is a process Petri net, and being $\mathcal{F} = \{F \mid$ simple circuit of $\mathcal{N}$ such that it does not contain places of $P_R\}$, the set of minimal T–Semiflows is $\mathcal{X} = \{F_T \mid F \in \mathcal{F}\}$.* □

Note 9 *In a process Petri net each transition has a unique input process state place (whose weight is equal to one) and zero or more input resource places. Extending the definitions presented in [XHC96] for SU–RAS, and given a marking, $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, a transition t is said to be*

- $\mathbf{m}$–process–enabled *(or,* process–enabled *at $\mathbf{m}$) if, and only if:*

  $${}^{\bullet}t \cap (P_0 \cup P_S) \neq \emptyset, \text{ and } \mathbf{m}[{}^{\bullet}t \cap (P_0 \cup P_S)] \neq 0$$

  That is, the transition is enabled by the corresponding process place (an active process is ready to fire it.) A transition that is no $\mathbf{m}$–process–enabled is $\mathbf{m}$–process–disabled.

- $\mathbf{m}$–resource–enabled *(or,* resource–enabled *at $\mathbf{m}$) if, and only if:*

  $${}^{\bullet}t \cap P_R \neq \emptyset \text{ and } \forall r \in ({}^{\bullet}t \cap P_R) \,.\, \mathbf{m}[r] \geq Pre[r, t]$$

  That is, no resource place is preventing the firing of t. A transition that is no $\mathbf{m}$–resource–enabled is $\mathbf{m}$–resource–disabled. □

Let us now prove a lemma relating the resources used by a state place and the resource enabling condition.

Lemma 10 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ be a marked process Petri net. Let $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ and let $t \in T$ such that $\{p\} = {}^{\bullet}t \cap (P_0 \cup P_S)$ and $\{q\} = t^{\bullet} \cap (P_0 \cup P_S)$. Then, $\mathbf{m}$ enables t if and only if $\mathbf{m}[p] > 0$ and $\forall r \in P_R \,.\, \mathbf{m}[r] \geq \mathbf{Y_r}[q] - \mathbf{Y_r}[p]$.*

Moreover, if $\mathbf{m} \xrightarrow{t} \mathbf{m}'$, $\mathbf{m}'$ is as follows:

- $\mathbf{m}'[p] = \mathbf{m}[p] - 1$
- $\mathbf{m}'[q] = \mathbf{m}[q] + 1$
- $\mathbf{m}'[p'] = \mathbf{m}[p], \forall p' \in (P_0 \cup P_S) \setminus \{p, q\}$
- $\mathbf{m}'[r] = \mathbf{m}[r] + \mathbf{Y_r}[p] - \mathbf{Y_r}[q], \forall r \in P_R$

Proof

First of all, let us remember that $\forall r \in P_R \,.\, \mathbf{C}[r,t] = \sum_{p \in (P_0 \cup P_S \setminus \mathcal{H}_r)} k_r \cdot \mathbf{C}[p,t] + \sum_{p \in \mathcal{H}_r} (k_r - \mathbf{Y_r}[p]) \cdot \mathbf{C}[p,t]$ where $k_r = \max_{p \in \mathcal{H}_r} \mathbf{Y_r}[p] (> 0)$ (Lemma 5.)

Then, in this case, $\forall r \in P_R \,.\, \mathbf{C}[r,t] = k_r \cdot \mathbf{C}[p,t] + k_r \cdot \mathbf{C}[q,t] - \mathbf{Y_r}[p] \cdot \mathbf{C}[p,t] - \mathbf{Y_r}[q] \cdot \mathbf{C}[q,t]$. But $\mathbf{C}[p,t] = -\mathbf{C}[q,t] = -1$, and then: $\forall r \in P_R \,.\, \mathbf{C}[r,t] = -k_r + k_r + \mathbf{Y_r}[p] - \mathbf{Y_r}[q] = \mathbf{Y_r}[p] - \mathbf{Y_r}[q]$.

Therefore, the first part of the Lemma is a direct translation of the enabling conditions for general Petri nets to the considered class of systems, and the second part is a direct translation of the firing rule. □

Based on the previous properties, the following proposition proves that when an acceptable initial marking is considered, a part can be processed in isolation, i.e. the system is well–defined.

Proposition 11 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ be a marked process Petri net. Let $(p_0, t_1, p_1, ..., t_k, p_k, t_{k+1}, p_0)$ be a simple circuit containing p_0. Then $\mathbf{m_0} \xrightarrow{t_1 t_2 \ldots t_{k+1}} \mathbf{m_0}$*

Proof

Let us prove this result by contradiction. Let us assume that there exists $i \in \{0..k\}$ such that $\mathbf{m_0} \xrightarrow{t_1, \ldots t_i} \mathbf{m}_i$ and such that $\mathbf{m}_i$ does not enable t_{i+1}. Notice that, according to Lemma 10, $\mathbf{m}_i$ is as follows:

- $\mathbf{m}_i[p_0] = \mathbf{m_0}[p_0] - 1$; $\mathbf{m}_i[p_i] = 1$; $\mathbf{m}_i[q] = 0, \forall q \in P_S \setminus \{p_i\}$, ($i \in \{1..k-1\}$),

- $\mathbf{m}_i[p_0] = \mathbf{m_0}[p_0]$, *(*$i = 0$*);*
- $r \in P_R, \mathbf{m}_i[r] = \mathbf{m_0}[r] - \mathbf{Y_r}[p_i]$.

Since t_{i+1} *is* $\mathbf{m}$*–process–enabled but not* $\mathbf{m}$*–resource–enabled, there exists an input resource place,* $r \in {}^\bullet t$*, such that*

$$\mathbf{m}[r] < \mathbf{Pre}[r, t_{i+1}]$$

Considering the marking invariant induced by $\mathbf{Y_r}$*,*

$$\mathbf{m}[r] + \mathbf{Y_r}[p_i] \cdot \mathbf{m}[p_i] + \sum_{s \in \|\mathbf{Y_r}\| \setminus \{r, p_i\}} \mathbf{Y_r}[s] \cdot \mathbf{m}[s] = \mathbf{m_0}[r]$$

Taking into account that $\sum_{s \in \|\mathbf{Y_r}\| \setminus \{r, p_i\}} \mathbf{Y_r}[s] \cdot \mathbf{m}[s] = 0$*, that* $\mathbf{m}[p_i] \geq 1$*, and that* $\mathbf{m}[r] < \mathbf{Pre}[r, t_{i+1}]$*, the following inequality can be obtained*

$$\mathbf{m_0}[r] < \mathbf{Pre}[r, t_{i+1}] + \mathbf{Y_r}[p_i]$$

Since $r \in {}^\bullet t_{i+1}$ *and the net is self–loop free,* $\mathbf{Pre}[r, t_{i+1}] > 0$ *(in fact, let us remember that* $\mathbf{Pre}[r, t_{i+1}] = \mathbf{Y_r}[p_{i+1}] - \mathbf{Y_r}[p_i]$*), which allows us to conclude that* $\mathbf{m}[r] < \mathbf{Y_r}[p_{i+1}]$ *which contradicts the hypothesis of* $\mathbf{m_0}$ *being an acceptable marking.* □

2.2.2 Modeling the whole system: S^4PR nets.

Usually, more than one type of part must be processed in a given system. A process Petri net is used to model the processing of a type of part. When several types of parts must be manufactured, a set of process Petri nets are needed. The interactions among the different types of parts can be established in terms of the competition for the set of system resources. Therefore, the Petri net modeling a given system in which a set of types of parts must be processed will be obtained by means of the composition of the Process Petri nets modeling the different types of parts to be produced. Since the only interactions among different types of processes are due to the use of shared resources, the composition of the process Petri nets will be done by means of the fusion of places modeling these common resources: this is a S^4PR net. Let us first remember the definition of the composition of nets via shared resources ([NV86].)

Definition 12 *Let* $\mathcal{N}_i = \langle P_i, T_i, \mathbf{C_i} \rangle, i \in \{1, 2\}$*, be two generalized Petri nets. Then:*

1. $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *are composable (via fusion of places) if and only if*

   (a) $T_1 \cap T_2 = \emptyset$ *and*
   (b) $P_C = P_1 \cap P_2 \neq \emptyset$

2. *The net* $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ *where*

   (a) $P = P_1 \cup P_2$,
   (b) $T = T_1 \cup T_2$,
   (c) $$\mathbf{C}[p,t] = \begin{array}{ll} \mathbf{C_1}[p,t] & \textit{If } (p,t) \in P_1 \times T_1 \\ \mathbf{C_2}[p,t] & \textit{If } (p,t) \in P_2 \times T_2 \\ 0 & \textit{in other case} \end{array}$$

   is called the composed net of $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *and it is denoted as* $\mathcal{N} = \mathcal{N}_1 \circ \mathcal{N}_2$. □

This definition is a restricted version of the one presented in [NV86] where places and transitions could be shared. It has been adapted to feet our needs.

Definition 13 *The class of S^4PR systems is defined recursively as follows:*

1. *A process Petri net is a S^4PR.*

2. *The composition of two composable S^4PR by fusion of the common resource places is also a S^4PR.*

3. *All the S^4PR systems are generated using the previous rules.* □

The name S^4PR has been chosen to identify this class of systems. This name does not have any specific meaning. It has been chosen because this class of nets is a generalization of previously introduced classes named as S^3PR [ECM95], and $L - S^3PR$ [EGVC98b]. In fact, $L - S^3PR \subset S^3PR \subset S^4PR$.

The S^4PR class (first introduced in [TCE99]) is similar to the WS^3PSR presented in [TM95], the S^4R presented in [BA96] and the S^3PGR^2 presented in [PR00a].

The way we have used to compose the models of the different types of parts to obtain the whole model can also be applied to previous work:

- *Free sequential systems*, presented in [LT79] can be composed, obtaining a similar class to the one presented here. Unfortunately, the authors did not consider this possibility since their objective was to prevent deadlocks on one-process based systems.

- The original *process Petri nets* presented in [BK90] can be composed, obtaining a restricted version of the class presented here, called $L-S^3PR$ in [EGVC98b] able to model the TO–SU–RAS.

- *Job subnet*, presented in [HC92] are composed using a different approach to the one presented here: they merge complete *resource nets* and *job subnets*, by means of the fusion of the matching process–related parts of both kind of nets. Given the restrictions introduced in the model presented in that paper this way of merging can be compared with the merging via resource places. The resulting nets are TO–SU–RAS, as in the previous case.

- *Simple sequential processes with resources*. The S^2PR nets were introduced in [ECM95] and can be composed in the same way proposed here, obtaining a new class called S^3PR. The obtained nets are adequate to represent PO–SU–RAS.

- *Extended simple sequential processes with resources* (ES^2PR) presented in [TGVCE98] can also be composed in the way presented here. The new class obtained in this way is the ES^3PR that is able to model the same restricted version of PO–MT–RAS.

- In [XHC96] the *production sequence model* of [BK90] is used in a similar way to the one used in $L-S^2PR$ [Val99], obtaining what they call *production Petri nets*. These production Petri nets can be composed in a similar way to the one presented here obtaining a model similar to $L-S^3PR$ able to model TO–SU–RAS.

- In [PR00a] another version of the $L-S^3PR$ is used, the *systems of simple linear sequential processes with resources* (S^2LSPR) defined as an appropriately restricted version of the S^3PR to eliminate dynamic flexible routings.

All these approaches can be considered as *process–oriented*: they first describe how processes are, then the way they can use resources, and finally how they interact with other types of processes.

Note 14 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ *be a marked* S^4PR*. In the following, it will be assumed to be of the form* $\mathcal{N} = (\bigcirc_{i=1}^{n-1} \mathcal{N}_i) \circ \mathcal{N}_n$*, where* $I_{\mathcal{N}} = \{1..n\}$ *is a finite set of indices and each* $\mathcal{N}_i = \langle P_{0_i} \cup P_{S_i} \cup P_{R_i}, T_i, \mathbf{C_i} \rangle$ *is a process Petri net. As a natural extension of the terms used in process Petri nets,*

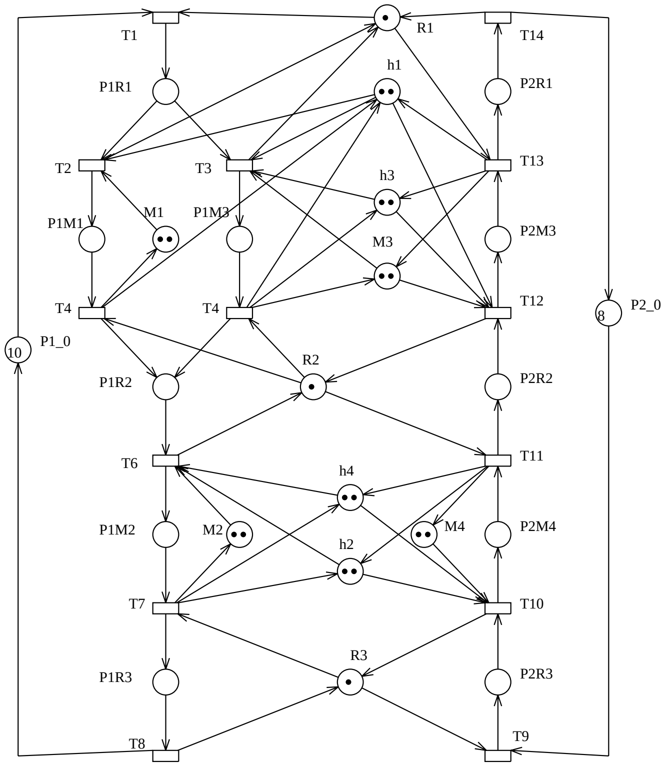


Figure 2.5: The S^4PR Petri net modeling the processing of parts in the cell of Figure 2.1

the sets $P_0 = \bigcup_{i \in I_{\mathcal{N}}} P_{0_i}$, $P_S = \bigcup_{i \in I_{\mathcal{N}}} P_{S_i}$, *and* $P_R = \bigcup_{i \in I_{\mathcal{N}}} P_{R_i}$ *will be called the sets of idle places, state places, and resource places, respectively.*

Moreover, for a given resource $r \in P_R$, $I_r = \{i \in I_{\mathcal{N}} \mid r \in P_{R_i}\}$ *denotes the set of indices corresponding to component process Petri nets using* r. □

Let us now give an initial marking to a S^4PR net.

Definition 15 *Let* $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle = \bigcirc_{i \in I_{\mathcal{N}}} \mathcal{N}_i$ *be a* S^4PR, *where* $\langle \mathcal{N}_i, \mathbf{m_0}_i \rangle$, $i \in I_{\mathcal{N}}$, *is a marked process Petri net. Then,* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *with* $\mathbf{m_0}$ *defined as follows is a* S^4PR *with an acceptable initial marking.*

1. $\forall i \in I_{\mathcal{N}} \,.\, \forall p \in P_{0_i} \cup P_{S_i} \,.\, \mathbf{m_0}[p] = \mathbf{m_{0_i}}[p]$, *and*

2. $\forall i \in I_{\mathcal{N}} \,.\, \forall r \in P_{R_i} \,.\, \mathbf{m_0}[r] = max_{i \in I_r} \mathbf{m_{0_i}}[r]$

In the composition of nets via shared resources the marking of places that are not common remains the same. For each shared place, the initial marking can be computed as the maximum of its initial markings in the different sub–systems. This approach seems reasonable since it assumes that the capacity of each resource when designing a process plan (a process Petri net) has been properly established for all the types of parts, either due to the real system constraints (the physical resources exist) or as a system parameter (used for simulation purposes, for example) to be definitively established in the future. Using that maximal value ensures a correct behavior for each component process Petri net.

In the following, when speaking about a marked S^4PR we are going to assume that its initial marking is acceptable.

Note 16 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ *be a marked* S^4PR*, and let* $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$*. The following conventions will be used:*

- *Each token in one state place will be called an* $\mathbf{m}$–active process *(or active process when no reference to the marking is needed.)*
- $|\mathbf{m}| = \sum_{p \in P_S} \mathbf{m}[p]$ *will be used to represent the number of* $\mathbf{m}$*–active processes.*
- $\mathcal{A}_{\mathbf{m}} = \{a_1 \ldots a_{|\mathbf{m}|}\}$ *will be used to identify each active process, identifying each token in a state place;*
- *the mapping* $\pi_{\mathbf{m}} : \mathcal{A}_{\mathbf{m}} \longrightarrow P_S$ *applies each* $\mathbf{m}$*–active process to the corresponding state place;*
- *finally, if* $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$*,* σ *is one of the interleavings of the subsequences* σ_{a_1}*,* σ_{a_2}*, …,* $\sigma_{a_{|\mathbf{m}|}}$*, where* σ_{a_i} *is the firing sequence carrying* $a_i \in \mathcal{A}$ *from its corresponding idle state place to* $\pi_{\mathbf{m}}(a_i)$*.* □

Figure 2.5 shows the marked S^4PR corresponding to the processing of two types of parts in the system whose layout is shown in Figure 2.1.

The following lemma establishes the conditions under which a reachable marking of a S^4PR $\mathbf{m}$ enables a transition.

Lemma 17 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ *be a marked* S^4PR*. Let* $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ *and let* $t \in T_i (\subseteq T)$ *such that* $\{p\} = {}^{\bullet}t \cap (P_0 \cup P_S)$ *and*

$\{q\} = t^{\bullet} \cap (P_0 \cup P_S)$. *Then,* $\mathbf{m}$ *enables* t *if and only if* $\mathbf{m}[p] > 0$ *and* $\forall r \in P_R \,.\, \mathbf{m}[r] \geq \mathbf{Y_r}[q] - \mathbf{Y_r}[p]$.

Moreover, if $\mathbf{m}\xrightarrow{t}\mathbf{m}'$, $\mathbf{m}'$ *is as follows:*

- $\mathbf{m}'[p] = \mathbf{m}[p] - 1$
- $\mathbf{m}'[q] = \mathbf{m}[q] + 1$
- $\mathbf{m}'[p'] = \mathbf{m}[p], \forall p' \in (P_0 \cup P_S) \setminus \{p, q\}$
- $\mathbf{m}'[r] = \mathbf{m}[r] + \mathbf{Y_r}[p] - \mathbf{Y_r}[q], \forall r \in P_R$

Proof

This is an obvious extension of Lemma 10. □

2.3 Some properties of S^4PR nets.

In this section we are going to see that previous results about minimal P–Semiflows and the correctness of the initial marking for process Petri nets can be trivially extended to S^4PR nets.

First of all, let us take a look at the structure of the incidence matrix for S^4PR nets guided by the example shown in the previous section. Table 2.1 shows the incidence matrix of the S^4PR net depicted in Figure 2.5.

The structure of this matrix can be seen as a set of boxes with the following meaning:

- The upper left one shows the flow relation for the nodes belonging to the state machine associated to the process Petri net modeling the processing of parts of type 1;
- The middle right one shows the flow relation for parts of type 2;
- The rest of the matrix corresponds to the flow relation between resource places and transitions of the two component *process Petri nets*.

Let us remark that this last box can be divided into two sub–matrices which correspond to the competition for the set of system resources.

Notice that for resources that are shared between both subsystems the corresponding row has non–zero values in both sub–matrices and that for resources that used only in one of them the row corresponding to the other subsystem is 0.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
P1_0	-1	0	0	0	0	0	0	1						
P1R1	1	-1	-1	0	0	0	0	0						
P1M1	0	1	0	-1	0	0	0	0						
P1M3	0	0	1	0	-1	0	0	0						
P1R2	0	0	0	1	1	-1	0	0						
P1M2	0	0	0	0	0	1	-1	0						
P1R3	0	0	0	0	0	0	1	-1						
P2_0									-1	0	0	0	0	1
P2R3									1	-1	0	0	0	0
P2M4									0	1	-1	0	0	0
P2R2									0	0	1	-1	0	0
P2M3									0	0	0	1	-1	0
P2R1									0	0	0	0	1	-1
R1	-1	1	1	0	0	0	0	0	0	0	0	0	-1	1
R2	0	0	0	-1	-1	1	0	0	0	0	-1	1	0	0
R3	0	0	0	0	0	0	-1	1	-1	1	0	0	0	0
M1	0	-1	0	1	0	0	0	0	0	0	0	0	0	0
M2	0	0	0	-1	-1	1	0	0	0	0	0	0	0	0
M3	0	0	-1	0	1	0	0	0	0	0	0	-1	1	0
M4	0	0	0	0	0	0	0	0	0	-1	1	0	0	0
h1	0	0	-1	0	1	0	0	0	0	0	0	-1	1	0
h2	0	0	-1	0	1	0	0	0	0	0	0	-1	1	0
h3	0	0	0	0	0	-1	1	0	0	-1	1	0	0	0
h4	0	0	0	0	0	-1	1	0	0	-1	1	0	0	0

Table 2.1: Incidence matrix of net in Figure 2.5

We are going to see which properties presented for process Petri nets are also valid for S^4PR. In this sense, let us recall a simplified version of a property presented in [NV86] relating P–Semiflows of the composed net and the ones of the individual component nets.

Theorem 18 *Let $\mathcal{N}_1 = \langle P_1, T_1, \mathbf{C_1}\rangle$ and $\mathcal{N}_2 = \langle P_2, T_2, \mathbf{C_2}\rangle$ two composable Petri nets. Let $\mathcal{N} = \mathcal{N}_1 \circ \mathcal{N}_2$ be the net obtained by composition of them by means of of the subset of common places ($P_{12} = P_1 \cap P_2$.) Let $\mathbf{Y_1}$ and $\mathbf{Y_2}$ be two P–Semiflows of $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively, and such that $\forall p \in P_{12} \,.\, \mathbf{Y_1}[p] = \mathbf{Y_2}[p]$. Then,*

$$\mathbf{Y} = \mathbf{Y_1}_{(P_1 \setminus P_{12}|P_1 \cup P_2)} + \mathbf{Y_2}_{(P_2 \setminus P_{12}|P_1 \cup P_2))} + \mathbf{Y_1}_{(P_{12}|P_1 \cup P_2)}$$

is a P–Semiflow of $\mathcal{N}$ ($\mathbf{Y} \cdot \mathbf{C} = \mathbf{0}, \mathbf{Y} \geq \mathbf{0}$.) □

The next lemma shows that minimal P–Semiflows related to state places for each *process Petri net* are also minimal P–Semiflows of the composed net.

Lemma 19 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle = \bigcirc_{i \in I_\mathcal{N}} \mathcal{N}_i$ be a S^4PR. Then, $\forall i \in I_\mathcal{N} \,.\, \mathbf{Y}_{\mathbf{S_i}(P_{0_i} \cup P_{S_i} \cup P_{R_i}|P_0 \cup P_S \cup P_R)}$ is a minimal P–Semiflow of $\mathcal{N}$.*

Proof
Clearly, according to Theorem 18, $\mathbf{Y}_{\mathbf{S_i}(P_{0_i} \cup P_{S_i} \cup P_{R_i}|P_0 \cup P_S \cup P_R)}$ is a P–Semiflow of $\mathcal{N}$.

Let us proceed by contradiction: let us suppose that $\exists j \in I_{\mathcal{N}}$ such that the P–Semiflow $\mathbf{Y_j}_{(P_{0_j} \cup P_{S_j} \cup P_{R_j} | P_0 \cup P_S \cup P_R)}$ is not minimal (we can suppose that $j = 1$ without lost of generality.)

Then, there exists $\mathbf{Y'_{S_1}}$, $\|\mathbf{Y'_{S_1}}\| \subset \|\mathbf{Y_{S_1}}_{(P_{0_1} \cup P_{S_1} \cup P_{R_1} | P_0 \cup P_S \cup P_R)}\|$, such that $\mathbf{Y'_{S_1}} \cdot \mathbf{C} = 0$. Then, $\mathbf{Y'_{S_1}}[P_{0_1} \cup P_{S_1} \cup P_{R_1}] \cdot \mathbf{C_1} = 0$ and this is a contradiction with the fact that $\mathbf{Y_{S_1}}$ is minimal in $\mathcal{N}_1$. □

The following lemma helps us to see that the use of resources is also conservative in S^4PR nets.

Lemma 20 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle = \bigcirc_{i \in I_{\mathcal{N}}} \mathcal{N}_i$ be a S^4PR. Let $r \in P_R$, and let $\mathbf{Y_{r_i}}$, $i \in I_{\mathcal{N}}$, be the minimal P–Semiflows associated to r in each $\mathcal{N}_i$. Then $\mathbf{Y_r} = \mathbf{1}_{(\{r\}|P_0 \cup P_S \cup P_R)} + \sum_{i \in I_r} \mathbf{Y_{r_i}}_{(P_{0_i} \cup P_{S_i} | P_0 \cup P_S \cup P_R)}$ is a minimal P–Semiflow of $\mathcal{N}$.*

Proof

According to Theorem 18 $\mathbf{Y_r}$ is clearly a P–Semiflow of $\mathcal{N}$. We have to prove that it is a minimal one.

Let us assume that $\mathbf{Y_r}$ is not minimal. Since $\mathbf{Y_r}[r] = 1$, this means that there exists another P–Semiflow $\mathbf{Y'_r}$ such that $\|\mathbf{Y'_r}\| \subset \|\mathbf{Y_r}\|$. Let us consider $p \in (\|\mathbf{Y_r}\| \setminus \|\mathbf{Y'_r}\|) \cap P_{S_i}$, for some $i \in I_r$. Notice that $\mathbf{Y'_r}[P_{0_i} \cup P_{S_i} \cup P_{R_i}] \cdot \mathbf{C}[P_{0_i} \cup P_{S_i} \cup P_{R_i}] = 0$, which implies that $\mathbf{Y'_r}[P_{0_i} \cup P_{S_i} \cup P_{R_i}]$ is a P–Semiflow of $\mathcal{N}_i$ such that $p \in \|\mathbf{Y_{r_i}}\| \setminus \|\mathbf{Y'_r}[P_{0_i} \cup P_{S_i} \cup P_{R_i}]\|$, which contradicts the hypothesis of $\mathbf{Y_{r_i}}$ being minimal in $\mathcal{N}_i$. □

The next proposition shows a basis of the left annuler space for the incidence matrix of the net.

Proposition 21 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle = \bigcirc_{i \in I_{\mathcal{N}}} \mathcal{N}_i$ be a S^4PR. Then, the set $\mathcal{Y} = \bigcup_{i \in I_{\mathcal{N}}} \{\mathbf{Y_{S_i}}\} \cup \{\mathbf{Y_r} \mid r \in P_R\}$ is a basis of the left annuller space*[1].

Proof

We are going to proceed in three steps:

1. *First of all, we are going to show that $rank(\mathbf{C}) = |P_S|$.*

   Each net $\mathcal{N}_{|(P_{0_i} \cup P_{S_i}, T_i)}$ is a strongly connected state machine, and the rows that model resources in each net $\mathcal{N}_i$ are linear combinations of the rows of the corresponding process places. Then looking at the structure of the matrix, we can trivially say that $rank(\mathbf{C}) = \sum_{i \in I_{\mathcal{N}}} rank(\mathbf{C_i}) = \sum_{i \in I_{\mathcal{N}}} |P_{S_i}|$.

[1] In order to define a vectorial space, we need a group, so using this terminology here is an abuse of language. We could use linear combinations with coefficients in $\mathbb{Q}^+$ (see, for example, [AT85].)

2. *Now we are going to see that the elements of $\mathcal{Y}$ are linearly independent.*

   The elements of $\{\mathbf{Y_r} \mid r \in P_R\}$ are mutually linearly independent because each one of them contains in its support an element $r \in P_R$ not belonging to the support of any other of them.

   The elements of $\{\mathbf{Y_r} \mid r \in P_R\}$ are linearly independent with respect to the ones in $\{\mathbf{Y_{S_i}} \mid i \in I_\mathcal{N}\}$ because these ones do not contain elements from P_R.

   Finally, the elements of $\{\mathbf{Y_{S_i}} \mid i \in I_\mathcal{N}\}$ are linearly independent one respect to each other because their supports have empty intersection.

3. *Let us now prove that $\mathcal{Y} = \{\mathbf{Y_{S_i}} \mid i \in I_\mathcal{N}\} \cup \{\mathbf{Y_r} \mid r \in P_R\}$ is a basis of the left annuller space.*

   Let K be the left annuller space of $\mathbf{C}$.

$$\begin{aligned} dim(K) &= \textit{number of rows of the matrix } \mathbf{C} - rank(\mathbf{C}) \\ &= |P_0| + |P_S| + |P_R| - |P_S| \\ &= \left|\bigcup_{i\in I_\mathcal{N}} \{\mathbf{Y_{S_i}}\}\right| + |\{\mathbf{Y_r} \mid r \in P_R\}| \end{aligned}$$

   Therefore, we can conclude.

□

Now, a similar result about T–Semiflows is going to be presented.

Lemma 22 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle = \bigcirc_{i\in I_\mathcal{N}} \mathcal{N}_i$ be a S^4PR.*

1. *$\forall i \in I_\mathcal{N}$, if $\mathbf{X_i}$ is a minimal T–Semiflow of $\mathcal{N}_i$, then $\mathbf{X_i}_{(T_i|T)}$ is a minimal T–Semiflow of $\mathcal{N}$.*

2. *If $\mathbf{X}$ is a minimal T–Semiflow of $\mathcal{N}$, then there exists $i \in I_\mathcal{N}$, such that $\mathbf{X} = \mathbf{X_i}_{(T_i|T)}$.*

Proof

1. *Let $\mathbf{X_i}$ be a minimal T–Semiflow of $\mathcal{N}_i$; then, $\mathbf{C_i} \cdot \mathbf{X_i} = 0$. Therefore, $\mathbf{X_i}_{(T_i|T)}$ is such that $\mathbf{C} \cdot \mathbf{X_i}_{(T_i|T)} = 0$. Let us assume that it is not minimal. Then there exists another T–Semiflow of $\mathcal{N}$ whose support is contained in the support of $\mathbf{X}_\mathbf{i}$. Since all the components corresponding to transitions not belonging to T_i are zero, it is also a T–Semiflow of $\mathcal{N}_i$ which contradicts the the hypothesis of $\mathbf{X_i}$ being minimal in $\mathcal{N}_i$.*

i	Support of the P–Semiflow	Projection over $P_{0_1} \cup P_{S_1} \cup P_{R_1}$	Projection over $P_{0_2} \cup P_{S_2} \cup P_{R_2}$
1	{ P1R1, P1M1, P1M3, P1R2, P1M2, P1R3, P1_0 }	{ P1R1, P1M1, P1M3, P1R2, P1M2, P1R3, P1_0 }	
2	{ P2M4, P2R2, P2M3, P2R1, P2R3, P2_0 }		{ P2M4, P2R2, P2M3, P2R1, P2R3, P2_0 }
3	{ P1R1, P2R1, R1 }	{ P1R1, R1 }	{ P2R1, R1 }
4	{ P1R2, P2R2, R2 }	{ P1R2, R2 }	{ P2R2, R2 }
5	{ P1R3, P2R3, R3 }	{ P1R3, R3 }	{ P2R3, R3 }
6	{ P1M1, M1 }	{ P1M1, M1 }	
7	{ P1M2, M2 }	{ P1M2, M2 }	
8	{ P2M4, M4 }		{ P2M4, M4 }
9	{ P1M3, P2M3, M3 }	{ P1M3, M3 }	{ P2M3, M3 }
10	{ P1M1, P1M3, P2M3, h1 }	{ P1M1, P1M3, h1 }	{ P2M3, h1 }
11	{ P1M2, P2M4, h2 }	{ P1M2, h2 }	{ P2M4, h2 }
12	{ P1M3, P2M3, h3 }	{ P1M3, h3 }	{ P2M3, h3 }
13	{ P1M2, P2M4, h4 }	{ P1M2, h4 }	{ P2M4, h4 }

Table 2.2: Minimal P–Semiflows of the S^4PR depicted in Figure 2.5

2. *Let $\mathbf{X}$ be a minimal T–Semiflow of $\mathcal{N}$; then $\mathbf{C} \cdot \mathbf{X} = 0$. Considering the structure of $\mathbf{C}$, $\mathbf{C_i} \cdot \mathbf{X}[T_i] = \mathbf{0}$ for each $i \in I_\mathcal{N}$, and considering that $\forall i, j \in I_\mathcal{N}, i \neq j \,.\, T_i \cap T_j = \emptyset$, it is obvious.*

□

Finally, the next proposition establishes which are the sets of minimal P– and T–Semiflows of a S^4PR net.

Proposition 23 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle = \bigcirc_{i \in I_\mathcal{N}} \mathcal{N}_i$ be a S^4PR. Then,*

1. $\mathcal{Y} = \{\mathbf{Y}_{\mathbf{S_i}(P_{0_i} \cup P_{S_i} \cup P_{R_i} | P_0 \cup P_S \cup P_R)} \mid i \in I_\mathcal{N}\} \cup \{\mathbf{Y}_{\mathbf{r}(P_{0_i} \cup P_{S_i} \cup P_{R_i} | P_0 \cup P_S \cup P_R)} | \, r \in P_R)\}$ *is the set of minimal P–Semiflows of $\mathcal{N}$.*

2. $\mathcal{X} = \{\mathbf{X_i}_{(T_i|T)} \mid i \in I_\mathcal{N}, \mathbf{X_i}$ *is a minimal T–Semiflow of $\mathcal{N}_i\}$ is the set of minimal T–Semiflows of $\mathcal{N}$.* □

Let us see these structural components for the considered example. Table 2.2 shows the list of minimal P–Semiflows for the net in Figure 2.5. The two minimal

i	Support of the T–Semiflow
1	{ T1, T3, T5, T6, T7, T13 }
2	{ T2, T4, T5, T6, T7, T13 }
3	{ T8, T9, T10, T11, T12, T14 }

Table 2.3: Minimal T–Semiflows of the S^4PR depicted in Figure 2.5

P–Semiflows numbered 1 and 2 are related to the state machines associated to each process Petri net. The others are related to the system resources. The table also shows in the third and fourth columns the minimal P–Semiflows of each one of the *process Petri nets*.

Table 2.3 shows the T–Semiflows of the net depicted in Figure 2.5. The two first T–Semiflows are associated to the *process Petri net* on the left, and they correspond to the first working plan; the other T–Semiflow is related to the *process Petri net* on the right, and it corresponds to the type $WP2$.

T–Semiflow $\mathbf{X_1} = T1 + T3 + T5 + T6 + T7 + T13$ induces the circuit whose nodes are: $X_{1_T} \cup X_{1_P} = \{P1\_0, T7, P1R1, T1, P1M1, T3, P1R2, T5, P1M2,$ $T6, P1R3, T13\}$.

These results will be used to study the behavior of S^4PR nets. In order to complete this section, let us finally present an alternative definition for the S^4PRclass of nets.

Definition 24 *Let $I_\mathcal{N}$ be a finite set of indices. A S^4PR is a connected generalized self–loop free Petri 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) $P_S = \bigcup_{i \in I_\mathcal{N}} P_{S_i}$, where for each $i \in I_\mathcal{N}, P_{S_i} \neq \emptyset$, and for each $i, j \in I_\mathcal{N}, i \neq j$, $P_{S_i} \cap P_{S_j} = \emptyset$.

   (b) $P_0 = \bigcup_{i \in I_\mathcal{N}} \{p_{0_i}\}$.

   (c) $P_R = \{r_1, r_2, \ldots, r_n\}$, $n > 0$.

2. *$T = \bigcup_{i \in I_\mathcal{N}} T_i$, where for each $i \in I_\mathcal{N}, T_i \neq \emptyset$, and for each $i, j \in I_\mathcal{N}, i \neq j$, $T_i \cap T_j = \emptyset$.*

3. *For each $i \in I_\mathcal{N}$, the subnet $\mathcal{N}_{|(P_{0_i} \cup P_{S_i}, T_i)}$ 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 $\{r\} = \|\mathbf{Y_r}\| \cap P_R$, $P_0 \cap \|\mathbf{Y_r}\| = \emptyset$, $P_S \cap \|\mathbf{Y_r}\| \neq \emptyset$, and $\mathbf{Y_r}[r] = 1$.*

5. $P_S = \bigcup_{r \in P_R} (\|\mathbf{Y_r}\| \setminus \{r\})$.

□

It is easy to see that this definition is equivalent to the one presented previously in a constructive way.

2.4 Liveness Analysis of S^4PR Models

One of the desirable properties of the systems we are considering is that the processing of each part, once started, will finish. When talking about concurrent systems this is related to the deadlock freeness property. Since each started processing will terminate, the initial state of the system (the idle state) can be reached from any reachable state. Moreover, since only acceptable initial markings are considered (adequate initial states from which every transition is fireable), it is clear that the behavioral property needed from the Petri net point of view is liveness. Liveness in systems modeled by means of S^4PR nets also implies that, provided that new raw materials arrive, their processing will be also possible.

In this section some important behavioral properties of S^4PR nets are presented. Having to deal with this class of nets, one would feel inclined to use similar results to the ones introduced in previous analogous approaches [ECM95, TGVCE98]. In [ECM95] the S^3PR class was presented and for it, empty siphons were used to detect deadlock problems. Another approach based on siphons for a class of ordinary nets are *process nets with resources* ([JXH00, PR01].) An alternative approach, based on transforming a weighted Petri net in an ordinary one (at least for the arcs that are output of the places of the net) and using the empty siphon characterization [LR96, IMA02].

However, since most of the previous work are applied to nets whose arc weights are always one, they use the same liveness characterization as in [ECM95]: a deadlock situation is related to some empty siphon. Since the structure of S^4PR nets is similar to the S^3PR, the question is whether siphons are useful in order to characterize deadlock problems or not.

S^4PR nets, and in general, nets with weighted arcs present some differences when dealing with deadlock problems: as it will be shown, it is possible to have deadlock problems with no related empty siphons. This introduces a new dimension: the distribution of tokens in places related with the siphons must be considered. Several definitions have been introduced in the literature studying siphons

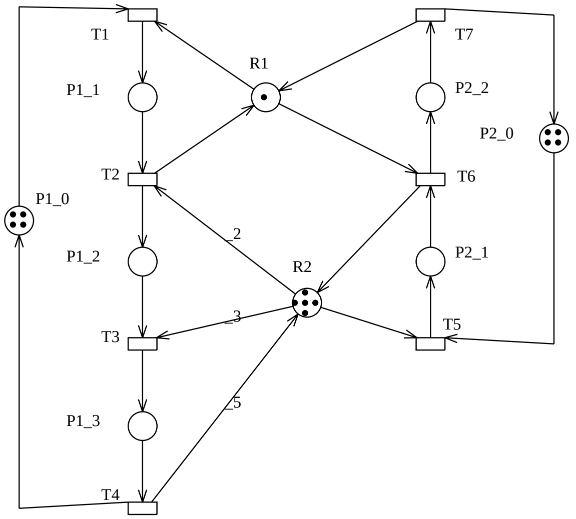


Figure 2.6: A S^4PR with deadlock problems.

(structural component) and the token distribution in places related to them (behavioral information) to relate deadlock situations and siphons in weighted Petri nets:

- In [Bra83] the concept of empty siphon in ordinary Petri nets is extended to the notion of *insufficiently marked siphon*: a siphon, D, is insufficiently marked at marking $\mathbf{m}$ if $\forall t \in D^\bullet . \exists p \in ({}^\bullet t \cap D) . \mathbf{m}[p] < \mathbf{Pre}[p,t]$. This definition extends one of the most important behaviorral properties based on siphons (a total deadlock in an ordinary Petri net implies an empty siphon, while a total deadlock in a weighted Petri net implies an insufficiently marked siphon.) This approach seemed promising when moving from S^3PR (if an active process cannot terminate, an empty siphon is reachable) to S^4PR. However, this property is not enough. Let us take a look at the S^4PR in Figure 2.6: marking $2 \cdot P1\_2 + R1 + R2 + 2 \cdot P1\_0 + 4 \cdot P2\_0$ is reachable; for it, transitions $T2$ and $T3$ are dead, but no siphon insufficiently marked exists.

- [BPP96, AE98] used siphons to deal with deadlock problems in S–RAS, giving a sufficient condition to ensure that no deadlock can occur. The objective is to keep all the minimal siphons "marked". A siphon D is said to be *marked* if and only if $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) \,.\, \exists p \in D \,.\, \mathbf{m}[p] \geq \max_{t \in p^\bullet} \mathbf{C}[p,t]$, that is, a siphon is "marked" if there exists a place enabling all of its output transitions. As it will be shown later this property is too strong, and reducing the requirements more permissive approaches can be obtained.

- [TCE99] presented a necessary condition for deadlock situations in terms of siphons for S^4PR nets. A deadlock prevention algorithm was proposed controlling the system so that the necessary condition cannot hold in the controlled system, obtaining live controlled systems. The proposed solution was similar to the one presented here but it has some inefficiencies that have been removed here.

- In [PR01] the idea of *resource–induced deadly marked siphons* for S^4PR nets is proposed: a siphon D is a resource–induced deadly marked siphon at $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ when each transition $t \in {}^\bullet D$ is disabled by some place belonging to D. In order to forbid deadlock problems no resource–induced deadly marked siphon should be allowed. They also presented a liveness characterization for S^4PR nets based on siphons but they do not use it for deadlock prevention.

In the rest of the chapter we are going to present a set of liveness characterizations for S^4PR nets. The first one (Theorem 26) does not use siphons, but concentrates on states where circular wait situations appear. The second one (Theorem 28), obtained from the first one, characterizes deadlock problems in terms of siphons and some related markings. Finally, the last one (Theorem 32) is also based on siphons, but establishes in a more clear way how deadlocked processes can be located around siphon components.

We will see that all the proposed characterizations are equivalent and also equivalent to the one proposed in [PR01]. The main advantage of the one proposed in Theorem 32 is that, as shown in the next chapter, it induces an efficient way of preventing deadlocks in S^4PR nets.

Let us present a lemma proving that the activation of a new process at a given reachable marking cannot increment the number of available resources. Later, a liveness characterization for S^4PR nets (Theorem 26) will be introduced.

Lemma 25 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^4PR. Let $\mathbf{m}, \mathbf{m'} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that $\forall p \in P_S \,.\, \mathbf{m'}[p] \geq \mathbf{m}[p]$. Then, $\forall r \in P_R \,.\, \mathbf{m'}[r] \leq \mathbf{m}[r]$.*

Proof
Let $r \in P_R$. The invariant relation induced by $\mathbf{Y_r}$ and the fact that $\forall p \in P_S \,.\, 0 \leq \mathbf{m}[p] \leq \mathbf{m}'[p]$ allows us to write

$$\mathbf{m}[r] = \mathbf{m_0}[r] - \sum_{p \in P_S} \mathbf{m}[p] \cdot \mathbf{Y_r}[p] \geq \mathbf{m_0}[r] - \sum_{p \in P_S} \mathbf{m}'[p] \cdot \mathbf{Y_r}[p] = \mathbf{m}'[r]$$

□

The following theorem presents a liveness characterization for S^4PR nets in terms of a property of circular waits.

Theorem 26 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^4PR. The system is non–live if and only if there exists a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ 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
$\Rightarrow$) If $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non–live, there exists at least a transition, t, that is dead at a marking $\mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$. Let $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ obtained by moving forward all the active processes (firing transitions of $T \setminus P_0{}^{\bullet}$) until no process enabled transition can fire. At this marking, $\mathbf{m}[P_S] \neq 0$ holds (i.e. there are some active processes and, in consequence, some $\mathbf{m}$–process–enabled transitions.) On the contrary, $\mathbf{m_0} = \mathbf{m}$, and then $\mathbf{m_0}$ could be reached from $\mathbf{m}'$. But, since the system is well defined, any minimal T–Semiflow containing t would be fireable from $\mathbf{m_0}$ (Lemma 7 and Proposition 11) and, in consequence, a firing sequence containing t would exist from $\mathbf{m}'$, which is a contradiction with t being dead at $\mathbf{m}'$. Therefore, any transition $t \in \pi_{\mathbf{m}}(a)^{\bullet}$ for any $a \in \mathcal{A}_{\mathbf{m}}$ is $\mathbf{m}$–process–enabled and $\mathbf{m}$–resource–disabled.
$\Leftarrow$) Let $t \in \pi_{\mathbf{m}}(a)^{\bullet}$ for $a \in \mathcal{A}_{\mathbf{m}}$. In order to fire t some more tokens are needed in some places belonging to $P_R \cap {}^{\bullet}t$. Since $\mathbf{m}$–active processes cannot progress, the only way to change the marking of such resources is by moving other processes, and the only possibility is to activate some idle processes. Let $ET = \bigcup_{a \in \mathcal{A}_{\mathbf{m}}} \{\pi_{\mathbf{m}}(a)^{\bullet}\}$ denote the set of $\mathbf{m}$–process–enabled transitions and $AP = \bigcup_{a \in \mathcal{A}_{\mathbf{m}}} \{\pi_{\mathbf{m}}(a)\}$ denote the set of state places with some $\mathbf{m}$–active-process, and let $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$. We are going to prove, by induction over the length of σ that:

1. *$\|\sigma\| \cap ET = \emptyset$.*
2. *$\forall p \in AP \,.\, \mathbf{m}'[p] \geq \mathbf{m}[p]$.*

Doing so, and since $\mathbf{m}[P_S \setminus AP] = 0$, it can be deduced that $\forall p \in P_S \,.\, \mathbf{m}'[P_S] \geq \mathbf{m}[P_S]$. Taking into account Lemma 25, $\forall r \in P_R \,.\, \mathbf{m}'[r] \leq \mathbf{m}[r]$. 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, $t \notin ET$. On the other hand, if $t \notin {}^\bullet AP$, $\forall p \in AP \,.\, \mathbf{m}'[p] = \mathbf{m}[p]$. If $t \in {}^\bullet AP$, let $t^\bullet \cap P_S = \{q\} \in AP$. In this case $\mathbf{m}'[q] = \mathbf{m}[q] + 1$ and the equality holds for the marking of the rest of places belonging to AP. In both cases, point 2 holds.*
- *General case. $\mathbf{m} \xrightarrow{\sigma''} \mathbf{m}'' \xrightarrow{t} \mathbf{m}'$, where $\sigma'', \mathbf{m}''$ verify the induction hypothesis: $\|\sigma''\| \cap ET = \emptyset$ and $\forall p \in AP \,.\, \mathbf{m}'[p] \geq \mathbf{m}[p]$. Applying Lemma 25 (taking also into account that $\mathbf{m}[P_S] > 0$) we can conclude that $t \notin ET$, and point 1 holds. Moreover, the fact that $t \in ET$ implies that $\forall p \in AP \,.\, \mathbf{m}'[p] \geq \mathbf{m}''[p] \geq \mathbf{m}[p]$, and we can conclude.*

□

In the example of Figure 2.6, at marking $\mathbf{m_1} = 2 \cdot P1\_2 + R1 + R2 + 2 \cdot P1\_0 + 4 \cdot P2\_0$, $T3$ is the only $\mathbf{m_1}$–process–enabled transition, which is disabled by $R2$. Therefore, it is dead. Resource $R2$ has only one token at $\mathbf{m_1}$, so $T3$ is a $\mathbf{m_1}$–resource–disabled transition.

Note 27 *A marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ verifying the conditions of Theorem 26 will be called a* deadlocked marking. *The term* bad marking *will also be used.* □

Theorem 26 relates non–liveness to the existence of a marking where 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 exist a set of circular waits between the blocked processes.

This concept of circular waits can be captured 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 shows that, when a bad marking as in Theorem 26 exists, a related siphon can be constructed; the reverse is also true. This establishes the bridge between behavior and model structure.

Theorem 28 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^4PR. The net is non–live if, and only if, there exists a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, and a siphon D such that $\mathbf{m}[P_S] > 0$ and the firing of each $\mathbf{m}$–process–enabled transition is prevented by a set of resource places belonging to D.*

Moreover, the siphon D is such that:

1. *$D_R = D \cap P_R = \{r \in P_R \mid \exists t \in r^\bullet$ such that $\mathbf{m}[r] < \mathbf{Pre}[r, t]$ and $\mathbf{m}[{}^\bullet t \cap P_S] > 0\} \neq \emptyset$;*

2. $D_S = D \cap P_S = \{p \in \mathcal{H}_{D_R} \mid \mathbf{m}[p] = 0\} \neq \emptyset$;

Proof

$\Rightarrow$) According to Theorem 26, $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non–live if and only if $\exists \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that the set of $\mathbf{m}$–process–enabled transitions is non–empty and each one of these transitions is a $\mathbf{m}$–resource–disabled transition. Let us construct D as follows:

- Let $D_R = \{r \in P_R \mid \exists t \in r^\bullet$ such that $\mathbf{m}[r] < \mathbf{Pre}[r, t]$ and $\mathbf{m}[{}^\bullet t \cap P_S] > 0\}$ be the set of resources disabling the $\mathbf{m}$–process–enabled transitions.

  $D_R \neq \emptyset$ because there is at least one $\mathbf{m}$–process–enabled transition.

- Let $D_S = \{p \in \mathcal{H}_{D_R} \mid \mathbf{m}[p] = 0\}$ be the set of holders of D_R unmarked at $\mathbf{m}$.

  We are going to prove that $D_S \neq \emptyset$ and $D_S \subset \mathcal{H}_{D_R}$. Let us suppose that $D_S = \emptyset$. Let $\mathbf{X}$ be a minimal T–Semiflow such that $\|\mathbf{X}\| \cap {}^\bullet D_R \neq \emptyset$. Let F_X be the directed path defined as $F_X = p_0 t_0 p_1 t_1 \ldots p_k t_k$ such that $\{t_0, t_1 \ldots t_k\} = \|\mathbf{X}\|$, $\forall i \in \{1, \ldots, k\} \,.\, p_i \in {}^\bullet t_i \cap P_S$ and $p_0 \in {}^\bullet t_0 \cap P_0$.

  This is a well–defined path because in S^4PR nets $\forall t_i \in T \,.\, {}^\bullet t_i \cap (P_0 \cup P_S)$ contains only a place. Let t be the last transition in the directed path F_X such that $t \in {}^\bullet D_R$. Let $r \in t^\bullet \cap D_R$. (Figure 2.7 sketches this situation.)

  Since $P_S \cap {}^\bullet t \in \mathcal{H}_r$ and $D_S = \emptyset$, $\mathbf{m}[P_S \cap {}^\bullet t] > 0$, i.e. t is a $\mathbf{m}$–process–enabled transition. Since $t \notin D_R{}^\bullet$ (if $t \in D_R{}^\bullet$, and taking into account that the net is self–loop free, 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 $D_S = \mathcal{H}_{D_R}$, since $\mathbf{m}[D_S] = 0$, $\forall r \in D_R \,.\, \mathbf{m}[r] = \mathbf{m_0}[r]$ which makes impossible for r to prevent the firing of any transition ($\mathbf{m_0}$ is acceptable.) Then, $D_S \subset \mathcal{H}_{D_R}$.

Let us now prove that $D = D_R \cup D_S$ is a siphon. Let $t \in {}^\bullet D$; two cases must be checked:

1. $t \in {}^\bullet D_R$. Let $r \in D_R$ 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 $p \in D_S$, and $t \in D_S{}^\bullet$.
   - If $\mathbf{m}[p] > 0$, since t is not enabled, there must exist a place $r' \in (P_R \cap {}^\bullet t)$ such that $\mathbf{m}[r'] < \mathbf{Pre}[r', t]$ i.e. disabling t. Then $r' \in D_R$, and in consequence, $t \in D_R{}^\bullet$.

2. $t \notin {}^\bullet D_R$. Then there exists $p \in D_S$ such that $t \in {}^\bullet p$ and $\exists r' \in D_R$ such that $p \in \mathcal{H}_{r'}$. If $\exists r \in ({}^\bullet t \cap D_R)$, $t \in D^\bullet$ and we can conclude.

   Let us now suppose that ${}^\bullet t \cap D_R = \emptyset$. In this case, t cannot be $\mathbf{m}$–process–enabled; if it was, by Theorem 26, t has to be $\mathbf{m}$–resource–disabled, and then, there would exist $r \in {}^\bullet t \cap D_R$.

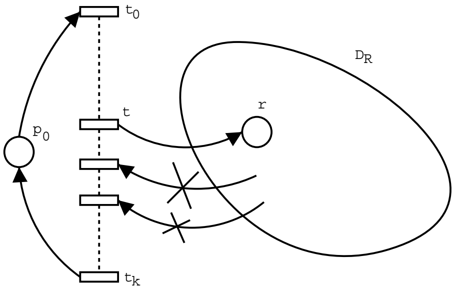


Figure 2.7: An abstract representation of the path and the last transition selected in the proof of Theorem 28

> *Let $\{q\} = {}^\bullet t \cap P_S$ (this place exists because $p \in \mathcal{H}_{r'}$ and ${}^\bullet t \cap D_R = \emptyset$.) 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 $q \in D_S$ (q is not marked), and $t \in {D_S}^\bullet$.*

Finally, notice that the firing of each $\mathbf{m}$–process–enabled transition is prevented by some resource places belonging to D (by construction.)

$\Leftarrow$) If each $\mathbf{m}$–process–enabled transition is prevented by a set of resource places (belonging to D), Theorem 26 allows us to conclude. □

In the example of Figure 2.6, at marking $\mathbf{m_1} = 2 \cdot P1\_2 + R1 + R2 + 2 \cdot P1\_0 + 4 \cdot P2\_0$, transition $T3$ is dead and the siphon $D1 = \{P1\_3,\ P2\_1,\ R2\}$ ($D1_R = \{R2\}$, $D1_S = \{P1\_3, P2\_1\}$) fulfills conditions stated in previous theorem: $R2$ is preventing the firing of $T3$, which is process–enabled, and all the places in $D1_S$ have zero tokens.

Note 29 *A siphon D and a marking, $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, verifying the properties of Theorem 28 will be said to be a* bad siphon *and a D–*deadlocked marking*, respectively.*

For a given bad siphon D, in the following the next notation will be used:

$$\forall p \in P_S \,.\, \mathbf{Y_{D_R}}[p] = \sum_{r \in D_R} \mathbf{Y_r}[p]$$

Notice that $\mathbf{Y_{D_R}}$ is the total amount of resource units belonging to D (in fact, to D_R) used by each active process in p. □

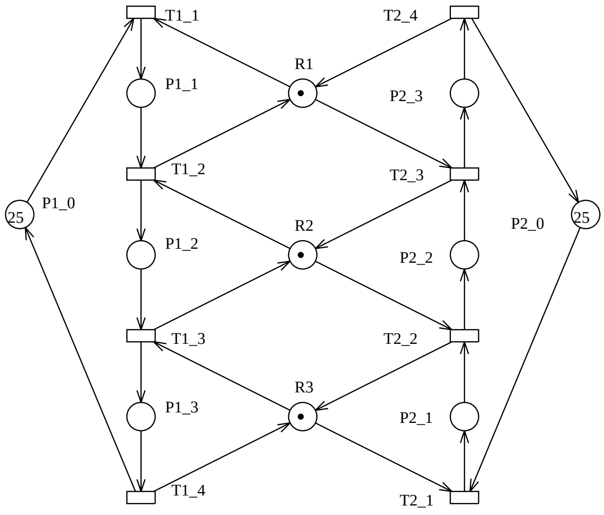


Figure 2.8: A S^4PR with deadlock problems.

It is important to remark that Theorem 28 does not state that if a transition t is dead at marking $\mathbf{m}$ this marking verifies the theorem's conditions, but that a reachable marking exists verifying them. In order to show this property, let us consider the S^4PR in Figure 2.8. At marking $\mathbf{m} = P1\_1 + P2\_1 + R2 + 24 \cdot P1\_0 + 24 \cdot P2\_0$ transitions $T1\_3$, $T1\_4$, $T2\_3$ and $T2\_4$ are dead, but no D–deadlocked siphon exists at $\mathbf{m}$. However, considering $\mathbf{m}\xrightarrow{T1\_2}\mathbf{m}' = P1\_2 + P2\_1 + R1 + 24 \cdot P1\_0 + 24 \cdot P2\_0$, this marking verifies conditions in Theorem 28 for the siphon $D' = \{R2, R3, P1\_3, P2\_2\}$.

One of the constraints imposed to process Petri nets composing a S^4PR was that each cycle must contain the corresponding idle place. Let us see that Theorem 28 is not true if this restriction is withdrawn. Let us consider the net shown in Figure 2.9, which is 'almost' a S^4PR: the cycle whose support is $\{P1\_1$, $T3$, $P1\_2'$, $T5\}$ does not contain place $P1\_0$. At the reachable marking $\mathbf{m} = P1\_1 + P2\_1 + r3 + 2 \cdot P1\_0 + 2 \cdot P2\_0$, transition $T2$ is dead. However, no reachable marking exists verifying the conditions of Theorem 28. In Chapter 4 a more general class of nets (S^*PR) allowing the modeling of systems with internal part recirculation will be presented, and a different approach to control them will

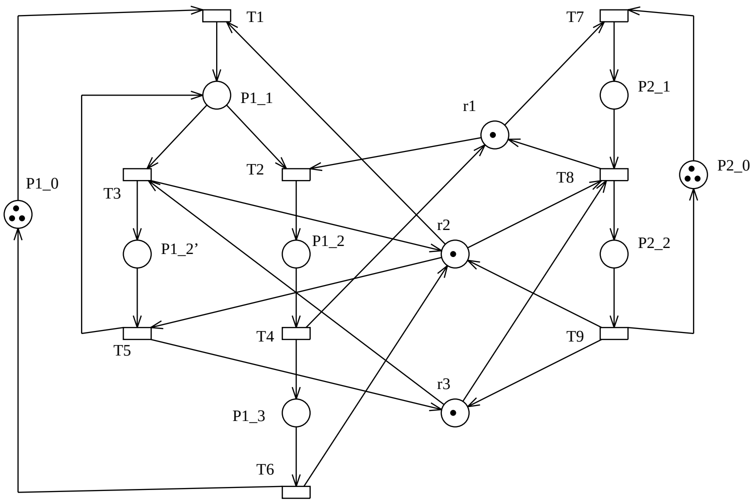


Figure 2.9: A net that is not a S^4PR (notice the arc from $T5$ to $P1\_1$.)

be proposed.

Another important comment to do related to this theorem is that there is not an alternative liveness characterization based on minimal siphons, as it can be shown using the S^4PR of Figure 2.10. Siphon $D = \{P1\_21, P1\_22, P2\_2, R2, R1\}$ is the only siphon that has associated markings as in Theorem 28 (for example, marking $\mathbf{m} = P1\_1 + P2\_1 + 3 \cdot R1 + 4 \cdot P1\_0 + 4 \cdot P2\_0$.) However, it is not minimal (in fact, this siphon contains a minimal one, $D3 = \{P1\_21, P1\_22, P2\_2, R1\}$.)

Definition 30 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, *be a marked* S^4PR. *Let* D *be a siphon of* $\mathcal{N}$. *Then,* $\mathcal{T}h_D = \mathcal{H}_{D_R} \setminus D_S$ *is* set of thieves *of* D[2]. □

The utility of this set will be understood later; for now, it should be clear that it represents places of the net that use resources of the siphon and do not belong to it.

Let us now introduce an alternative liveness characterization that shows that in a non–live system it is always possible to reach a deadlocked marking with all

[2] We will use sometimes in the following $\mathcal{T}h_{D_R}$ to show the relation among these two sets.

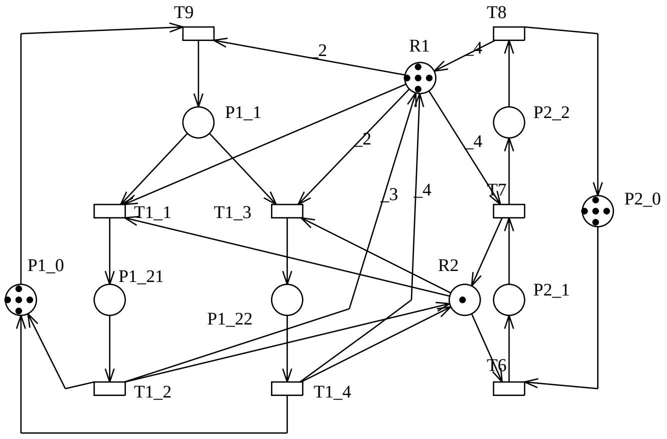


Figure 2.10: A different S^4PR with deadlock problems. It has no minimal siphon as in Theorem 28

.

the active processes using resources from an associated siphon. That is, the following characterization establishes where active processes can stay if deadlocked situations are possible.

For this we will need a technical result about reachable markings. The idea is very simple: given any reachable marking, we can select one of the active processes; the marking where all the processes except the selected one are at the same state as in the original one, and the selected one has not been activated, is reachable.

Proposition 31 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^4PR. Let $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, $\mathbf{m_0} \neq \mathbf{m}$. Let $q \in P_{S_i}$ such that $\mathbf{m}[q] > 0$, and $\mathbf{m}'$ the token distribution defined as follows:*

- $\mathbf{m}'[q] = \mathbf{m}[q] - 1$;
- $\forall p \in P_S \setminus \{q\} \,.\, \mathbf{m}'[p] = \mathbf{m}[p]$;
- $\mathbf{m}'[p_{0_i}] = \mathbf{m}[p_{0_i}] + 1$;
- $\forall p_0 \in P_0 \setminus \{p_{0_i}\} \,.\, \mathbf{m}'[p_0] = \mathbf{m}[p_0]$;
- $\forall r \in P_R \,.\, \mathbf{m}'[r] = \mathbf{m}[r] + \mathbf{Y_r}[q]$.

Then, $\mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$.

Proof

Let us assume that $\mathbf{m_0}\xrightarrow{\sigma}\mathbf{m}$, *being* σ *one interleaving of* σ_{a_1}, σ_{a_2}, ..., $\sigma_{a_{|\mathbf{m}|}}$. *Let us consider an* $\mathbf{m}$*–active process* a_i *such that* $\pi_{\mathbf{m}}(a_i) = q$ *(such process exists since we are considering that* $\mathbf{m}[q] > 0$.*) Let us also assume that* $\sigma = \sigma_1\sigma_{q1}\sigma_2\sigma_{q2}\ldots\sigma_l\sigma_{ql}\sigma_{l+1}$, *with* $\sigma_{a_i} = \sigma_{q1}\sigma_{q2}\ldots\sigma_{ql}$, *and let us denote the intermediate markings as*

$$\mathbf{m_0}\xrightarrow{\sigma_1}\mathbf{m_1}\xrightarrow{\sigma_{q1}}\overline{\mathbf{m_1}}\ldots\xrightarrow{\sigma_l}\mathbf{m_l}\xrightarrow{\sigma_{ql}}\overline{\mathbf{m_l}}\xrightarrow{\sigma_{l+1}}\mathbf{m}$$

Let us now prove that $\mathbf{m_0}\overset{\sigma_1\sigma_2\ldots\sigma_l\sigma_{l+1}}{\longrightarrow}\mathbf{m}'$ *as in the proposition statement.*

Let us concentrate on marking $\mathbf{m_1}$. *From it, the firing of* σ_{q1} *corresponds to the introduction in the system (and partial evolution) of a new process (the one identified as* a_i*), while the rest of processes active at* $\mathbf{m_1}$ *remain at the same state as in* $\mathbf{m_1}$. *From the point of view of resources,* $\forall r \in P_R \,.\, \mathbf{m_1}[r] \geq \overline{\mathbf{m_1}}[r]$. *Therefore, since* σ_2 *moves only processes different than* a_i, *and it is fireable from* $\overline{\mathbf{m_1}}$, *it must also be fireable from* $\mathbf{m_1}$ *where more resources are available (*$\mathbf{m_1}\xrightarrow{\sigma_2}\mathbf{m_2}'$.*) Moreover, since all the* $\mathbf{m_2}'$*–active processes are in the same state as in markings* $\mathbf{m_2}$ *and* $\overline{\mathbf{m_2}}$, *except* a_i *that remains in its idle state place,* $\forall r \in P_R \,.\, \mathbf{m_2}'[r] \geq \mathbf{m_2}[r]$, *and* $\forall r \in P_R \,.\, \mathbf{m_2}'[r] \geq \overline{\mathbf{m_2}}[r]$, *which makes* σ_3 *fireable from* $\mathbf{m_2}'$. *This reasoning can be inductively applied to* $\sigma_4, \ldots \sigma_{l+1}$.

Notice that at $\mathbf{m}'$ *the following things are true:* $\mathcal{A}_{\mathbf{m}'} = \mathcal{A}_{\mathbf{m}} \setminus \{a_i\}$; $\forall a \in \mathcal{A}_{\mathbf{m}'} \,.\, \pi_{\mathbf{m}'}(a) = \pi_{\mathbf{m}}(a)$, *while the token corresponding to the process* a_i *in* $\mathbf{m}$ *is in* p_{0_i} *at* $\mathbf{m}'$. *Then,*

- $\forall r \in P_R \,.\, \mathbf{m}'[r] = \mathbf{m}[r] + \mathbf{Y_r}[\pi_{\mathbf{m}}(a_i)] = \mathbf{m}[r] + \mathbf{Y_r}[q]$;
- $\forall p \in P_0 \setminus \{p_{0_i}\} \,.\, \mathbf{m}'[p] = \mathbf{m}[p]$;
- $\mathbf{m}'[p_{0_i}] = \mathbf{m}[p_{0_i}] + 1$;
- $$\begin{aligned} \mathbf{m}'[q] &= |\{a \in \mathcal{A}_{\mathbf{m}'} \mid \pi_{\mathbf{m}'}(a) = q\}| \\ &= |\{a \in \mathcal{A}_{\mathbf{m}} \setminus \{a_i\} \mid \pi_{\mathbf{m}}(a) = q\}| \\ &= \mathbf{m}[q] - 1; \end{aligned}$$
- $$\begin{aligned} \forall p \in P_S \setminus \{q\} \,.\, \mathbf{m}'[p] &= |\{a \in \mathcal{A}_{\mathbf{m}'} \mid \pi_{\mathbf{m}'}(a) = p\}| \\ &= |\{a \in \mathcal{A}_{\mathbf{m}} \setminus \{a_i\} \mid \pi_{\mathbf{m}}(a) = p\}| \\ &= \mathbf{m}[p]. \end{aligned}$$

□

The following liveness characterization establishes that when a S^4PR is not live, there exists a deadlocked marking such that all the active processes are "stealing" tokens from the set of resources of an associated siphon. This alternative characterization is useful to generate a deadlock prevention solution, allowing us to concentrate on siphon and their thieves, "forgetting" those active processes that are not related to the siphons, and giving better computational results when controlling the system.

Theorem 32 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^4PR. The net is non–live if, and only if, there exists a siphon D, and a marking $\mathbf{m}_D \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, such that:*

1. $\mathbf{m}_D[P_S] > 0$.
2. $\mathbf{m}_D[P_S \setminus \mathcal{T}h_D] = 0$.
3. $\forall p \in \mathcal{T}h_{D_R}$ *such that* $\mathbf{m}_D[p] > 0$, *the firing of each* $t \in p^\bullet$ *is prevented by a set of resource places belonging to* D.

Proof

⇒) According to Theorem 28, the net is non–live if, and only if, there exists a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, *with* $\mathbf{m}[P_S] > 0$, *and a siphon* $D = D_S \cup D_R$ *such that the firing of each* $\mathbf{m}$*–process–enabled transition is prevented by a set of resource places belonging to* D.

Now, we are going to prove that there exists a marking $\mathbf{m}_D$ *that characterizes the non–liveness of the net system being all the active processes in the set of thieves of* D_R, *i.e.* $\mathbf{m}_D[P_S \setminus \mathcal{T}h_D] = 0$.

Since $D_S \subset \mathcal{H}_{D_R}$, *and* $\mathbf{m}[D_S] = 0$, $\mathbf{m}[\mathcal{T}h_D] > 0$ *(if* $\mathbf{m}[\mathcal{T}h_D] = 0$, $\mathbf{m}[\mathcal{T}h_D \cup D_S] = 0$*; then, no process is using resources of* D_R *(then,* $\forall r \in D_R \,.\, \mathbf{m}[r] = \mathbf{m_0}[r]$*), while* D_R *is preventing the firing of some* $\mathbf{m}$*–process–enabled transitions which is a contradiction with the fact of being* $\mathbf{m_0}$ *an acceptable initial marking.)*

Let us now consider the following partition of the set of $\mathbf{m}$*–marked state places:*

$$\{p \in P_S \mid \mathbf{m}[p] > 0\} = \{p \in \mathcal{T}h_D \mid \mathbf{m}[p] > 0\} \cup \{p \in P_S \setminus \mathcal{T}h_D \mid \mathbf{m}[p] > 0\}$$

Notice that:

- *The first subset is non–empty.*
- *No process of the second subset uses resources of* D_R.

Now, we can apply Proposition 31 to each one of the processes of the second subset. In this way we will obtain a marking $\mathbf{m}_D$ *such that:*

- $\forall p \in D \cup \mathcal{T}h_D \,.\, \mathbf{m}_D[p] = \mathbf{m}[p]$;
- $\forall r \in P_R \setminus D_R \,.\, \mathbf{m}_D[r] = \mathbf{m_0}[r]$;
- $\forall p \in P_S \setminus (D_S \cup \mathcal{T}h_D) \,.\, \mathbf{m}_D[p] = 0$; *(there are no active processes out of the siphon related state places.)*
- $\forall i \in I_{\mathcal{N}}$, $\mathbf{m}_D[p_{0_i}] = \mathbf{m}[p_{0_i}] + \sum_{p \in P_{S_i} \setminus \mathcal{T}h_D} \mathbf{m}[p]$ *(the active processes in state places not related to the siphon are at the corresponding idle state.)*

Then, $\mathbf{m}_D$ verifies $\mathbf{m}_D[P_S] = \mathbf{m}_D[\mathcal{T}h_D] > 0$; since the set of $\mathbf{m}_D$–process–enabled transitions is a subset of the set of $\mathbf{m}$–process–enabled transitions and the resources of D_R were preventing the firing of the $\mathbf{m}$–process–enabled transitions of the first subset, and $\forall r \in D_R \,.\, \mathbf{m}_D[r] = \mathbf{m}[r]$, clearly each $\mathbf{m}_D$–process–enabled transition is still disabled at $\mathbf{m}_D$ by resources belonging to D_R.
$\Leftarrow$) Since $\mathbf{m}_D[P_S] > 0$ and $\forall p \in \mathcal{T}h_{D_R}$ such that $\mathbf{m}_D[p] > 0$, the firing of each $t \in p^\bullet$ is prevented by a set of resource places belonging to D, $\mathbf{m}_D$ and D verify conditions of Theorem 26, which is sufficient to conclude that $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is non–live. □

This liveness characterization directly relates bad markings with system states in which all the active processes stay in thief places of a bad siphon. This will be specially useful when trying to control the system in order to ensure a live behaviour since the potencial problems are located around siphons. This aspect will be developed in the following chapter.

2.5 Conclusions

In this chapter, the class of S^4PR nets has been introduced. The syntactic characteristics of the nets of such class are derived from the domain to which they are devoted: sequential resource allocation systems, in which a set of sequential processes must share a set of (conservative) resources. It has been shown how the model properties (Petri net properties, in this case) map into domain characteristics. In this sense, token conservation laws induced by (minimal) P–Semiflows correspond to the conservative nature of system resources or to the closed structure of the processes running in the system. On the other side, executable (minimal) T–Semiflows correspond to complete possible executions of the involved processes. Using this well defined mapping into model and system structures, the deadlock problem analysis has been studied from the Petri net model structural point of view, leading to a complete liveness characterization.

From the domain point of view, the S^4PR nets allowed to remove some of the restrictions (syntactic from the Petri net point of view, but related to the number and variety of real systems able to deal with, from the application point of view) imposed by previous classes of Petri net models devoted to the analysis and control of S–RAS. In this sense, the only constraint still maintained in S^4PR nets is that no inner cycle is allowed in the behavior of a process.

In the second part of the chapter a liveness study for S^4PR systems has been presented, showing a characterization for deadlock problems. This characterization is based on the existence of circular waits (involving resource places) related to the existence of problematic system states, re-establishing a classical result related to deadlock situations for the class of models considered here. Later, the relation of problematic markings with some special siphons has been shown, providing a way to study these bad markings in terms of some net siphons. As it will be shown in the next chapter, the siphon–based characterizations can be used to prevent deadlock problems.

Chapter 3

Deadlock Prevention Policies for S^4PR nets

Abstract
One of the advantages of the use of formal models as S^4PR is that the model can be used both to analyze the system behavior, and to control it. In the present chapter we are going to use the liveness characterizations introduced in Chapter 2 in order to prevent deadlock problems.

3.1 Introduction

The objective of the present chapter is to concentrate on how to add control to the system to ensure that no deadlock situations can happen. For this task the departure state is promising: the system is modeled by means of a S^4PR, and for this class some liveness characterizations have been introduced in the previous chapter. In consequence, we are going to concentrate on how to use these characterizations to reach the objective. We need to ensure not only that no deadlock will be reachable but also that the resulting system is as permissive as possible. Permissiveness here is related to the number of reachable states in the controlled system.

The quality (based on the permissiveness) of a prevention approach relies in two main aspects:

- A good identification (a characterization if possible) of deadlock related states.

- A good control method able to forbid the deadlock related states (without forbidding too many of the good ones).

Considering the first point, most of the previously proposed methods lack of a full liveness characterization, providing only necessary conditions. Then, the prevention approach for them will need to ensure that such conditions cannot occur. This is the case of [LT79, ECM95, TM95, KTJK97, TGVCE98, AE98, TE99, GL01]. With respect to the second one, the proposed methods usually apply the control at the level of process activation; that is, if the activation of an idle process could lead to deadlock, such activation is forbidden. This kind of prevention is easy to implement, but it usually gives controlled systems with poor use of the system resources ([IM80, Sin89b]). From this point of view the objective would be to look for a way to control the system in such a way that only deadlock related states are forbidden. In this way, the undesired states would be eliminated, while allowing as many concurrency as possible in the execution of processes in the controlled system. As it will be shown, the key issue will be to look for "bad states" that are "around" the siphons of the Petri net model, obtaining quite good solutions.

3.2 What a maximally permissive control policy should do?

The final objective of a deadlock prevention policy is to constrain the allowable firing sequences so that only non–deadlocked states are reachable. A way of doing that consists in the addition of new preconditions to the firing of transitions by means of new places and arcs, with an adequate initial marking.

Let us give some intuition about this using the reachability graph of the S^4PR of Figure 2.6 (page 54), which is depicted in Figure 3.1. The reachable states can be classified into three categories:

1. The first one (type 1) contains those markings from which $\mathbf{m_0}$ is reachable. These markings are not involved in deadlock problems (the shadowed states in Figure 3.1).

2. The second class (type 2) is composed of those markings that are not D–deadlocked for any siphon, and such that $\mathbf{m_0}$ is not reachable from them.

3. Finally, the third class (type 3) is composed of those markings that are D–deadlocked for some siphon D (depicted as black boxes in the Figure).

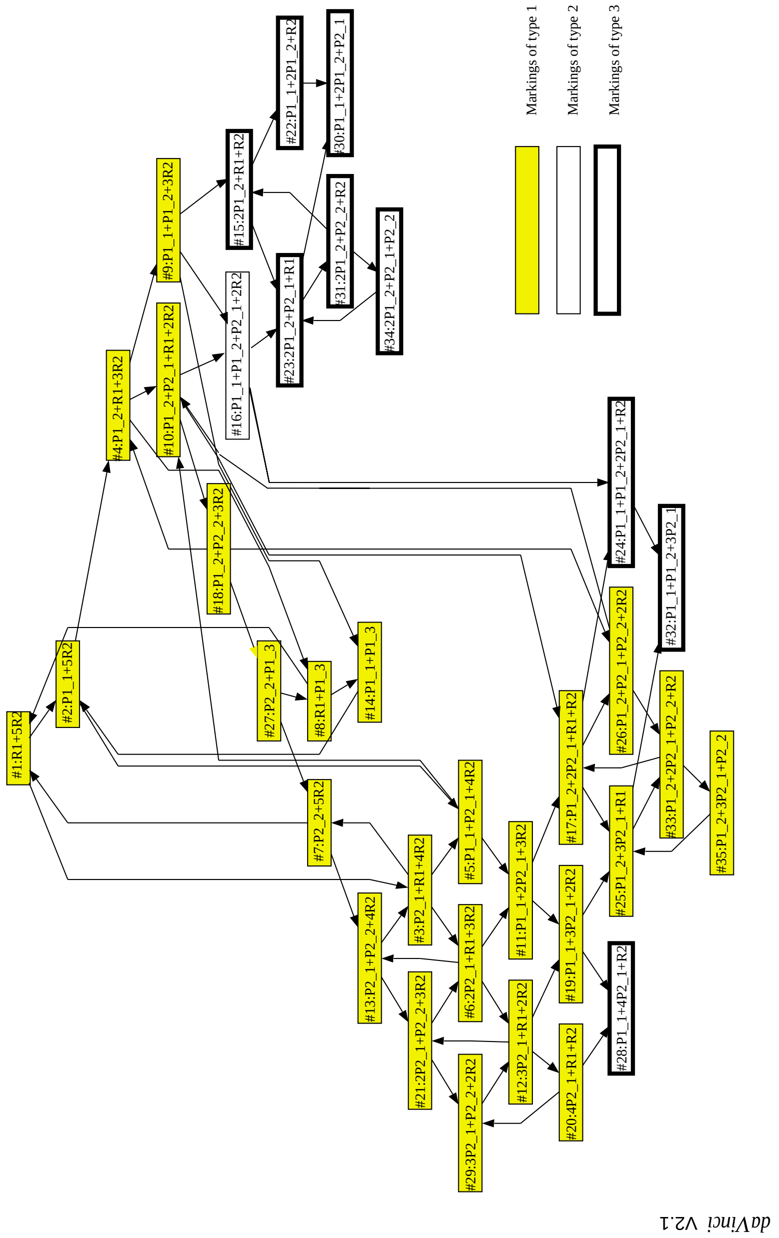


Figure 3.1: Reachability graph of the net of Figure 2.6 (marking of the idle places not shown for the sake of clarity)

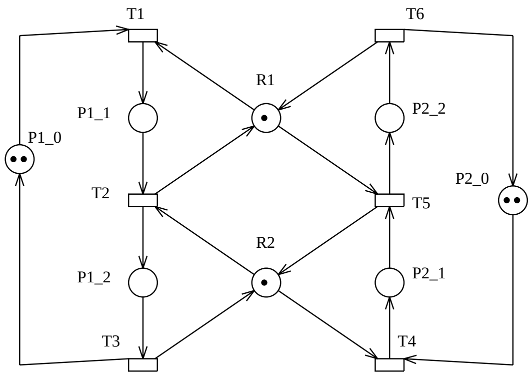


Figure 3.2: S^4PR net that can reach a deadlocked marking

For example, marking $\#32 : P1\_1 + P1\_2 + 3 \cdot P2\_1 + 2 \cdot P1\_0 + P2\_0$ is a $D2$–deadlocked marking, where the siphon is $D2 = \{P1\_3, P2\_2, R1, R2\}$, while the marking $\#16 : P1\_1 + P1\_2 + P2\_1 + 2 \cdot R2 + 2 \cdot P1\_0 + 3 \cdot P2\_0$ is not D–deadlocked, for any siphon D. Nevertheless, from marking $\#16$ we will reach in an inevitable way a D–deadlocked marking (both successor markings $\#23$ and $\#24$ are D–deadlocked).

Let us concentrate on how a bad situation can be controlled and all the related bad markings forbidden. Let us now consider the net in Figure 3.2. Its reachability graph is depicted in Figure 3.3. There, marking $\#5 : P1\_1 + P2\_1 + P1\_0 + P2\_0$ is a deadlock. This deadlock occurs because the process in place $P\mathit{1\_1}$ is waiting for the resource $R\mathit{2}$ which is being held by process in place $P\mathit{2\_1}$; this process is waiting for resource $R\mathit{1}$ that is being held by process in place $P\mathit{1\_1}$. In this case, it is easy to modify the system in such a way that the marking $\#\mathit{5}$ is not reachable anymore: a place, PC_1 (Figure 3.4), establishing a mutual exclusion between places $P\mathit{1\_1}$ and $P\mathit{2\_1}$ in such a way that they cannot be simultaneously marked solves the problem. This place will be called *control place*. While in this example the addition of just one place is enough to have a controlled system, this will not be always the case. For other systems, more control places will be needed. This way of adding control is related to the *generalized mutual exclusion constraints* (GMEC) introduced in [GDS92].

The problem with adding control for each deadlocked state is that too many control places could be needed: in the worst case, one for each marking of types one and two. Fortunately, Theorems 28 and 32 relate bad states with some special

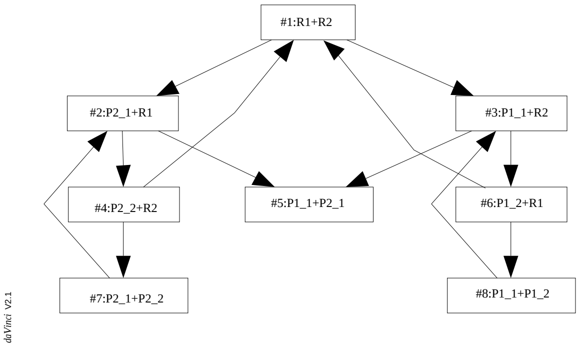


Figure 3.3: Reachability graph of net of Figure 3.2 (the markings of the idle state places is not shown for the sake of brevity).

siphons, allowing to establish a middle point between the control based exclusively on the information provided by deadlocked markings and the control based on process activation.

Another important question is the way in which bad markings are forbidden; in the example it is obvious that the addition of PC_1 strictly avoids the problematic marking; the general case will be much more complicated.

The addition of a place and its corresponding arcs introduces a new row in the incidence matrix, that is, a linear restriction. We are interested in control policies based on the addition of Petri net components: places and arcs (which we will use to forbid some bad states). The main reason for this is that the control via the addition of Petri net elements will allow us the study of the new system in the same terms of the original one. In consequence, the approach proposed in the following sections corresponds to the addition of a set of *linear restrictions* to the initial system, "cutting" a set of markings of the reachability set. As an example of this, notice that the net of Figure 3.4, resulting from the addition of the control place to the net in Figure 3.2 is a new S^4PR net: the added place PC_1 follows the restrictions about resource places.

The use of linear restrictions has a drawback: in some cases they forbid not only the desired bad states but also some good ones. This fact has been previously shown in [Val99, GVTCE00]. A maximally permissive control policy should prevent all the bad markings (types 2 and 3), forbidding no good markings.

Forbidding deadlocked markings by means of the addition of linear restrictions can be done according to the following two main perspectives:

- First approach: to cut as few states as possible; that is, once the states that

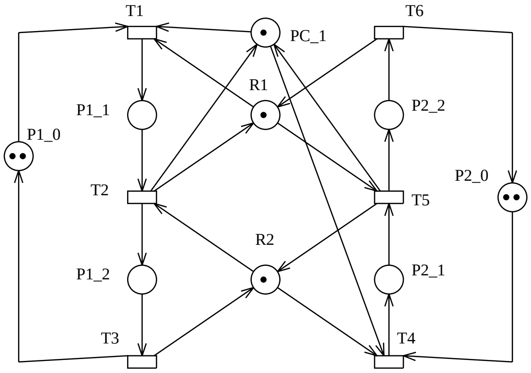


Figure 3.4: Controlled net

are D–deadlocked have been characterized, a linear restriction can be added that avoids just these bad states.

The problem with this approach is that it is not able to deal with the markings of type three (remember that they are not D–deadlocked). Then, once the original system has been controlled, a new system is obtained (the old one plus a linear restriction) that can have deadlock problems: further work is needed to obtain a “good” one. For this reason this approach will be called *in several steps*.

This approach is local, oriented to the control of the transitions closely related to bad siphons. In this way, new linear restrictions can be added in consecutive iterations in order to get a live system.

- Second approach: to cut ‘enough’ states to avoid all the problems; that is, for each siphon that can have D–deadlocked markings, the set of linear restrictions to be added can be computed in such a way that neither D–deadlocked markings nor markings that in an inevitable way conduct to D–deadlocked ones can be reached.

  This way of controlling is usually closely related to the idea of process activation. The objective is to cut all the D–deadlocked markings (markings of type 2), all the ones of type 3 and, perhaps, some of the good ones (without introducing new problems). We can expect that this way of controlling the system will be more restrictive but more simple and faster to compute.

Let us comment on the main previous prevention solutions based on the addition of linear restrictions.

- In [LT79], a resource–oriented approach is considered: for each resource, as many control places as output transitions are added (except for the last one). They control the system in such a way that once a part starts requesting tokens from a given resource place (that is, the firing of one of its output transitions occurs), there will be enough available tokens in the resource to grant all the future requests and terminate.

  This is one of the first control policies that proposed the modification of existing nets with the addition of linear restrictions in order to get live models. The main drawbacks are that it is based on the control of each resource and that it is not easy to generalize the policy to deal with systems with several different concurrent processes.

- In [ECM95] the control policy is based on the detection of deadlock problems by means of siphons. It is also an approach in one step. The way to apply the control was to add a control place for each minimal siphon constraining the system evolution in such a way that each part entering into the system makes a ‘reservation’ of enough copies of the resources related to the siphon to be sure that it will not reach an empty state. There are some further evolutions of this control policy for more general classes of nets in [TM95, TGVCE98, TE99].

- In [BCZ97, AE98] the class of nets is similar to the one used here. It is an approach in several steps. The control is based on siphons, and the way to add the control places (one for each siphon) has similarities to the one used in [LT79], but considering the siphon as a whole; that is, control place arcs reproduce the total number of tokens that the process has withdrawn from the resource places belonging to the siphon. The method guarantees that at each reachable marking there is always a place in the siphon with enough tokens to fire any of its output transitions.

- In [PR01] the approach in one step is followed. It is based on the division of the processing paths in zones (neighborhoods) defined by means of the imposition of a partial order in the set of resources. The authors claim that it is very efficient and the policy is proposed as an avoidance policy; however, it can be implemented as a prevention one. The proposed solution is too constraining.

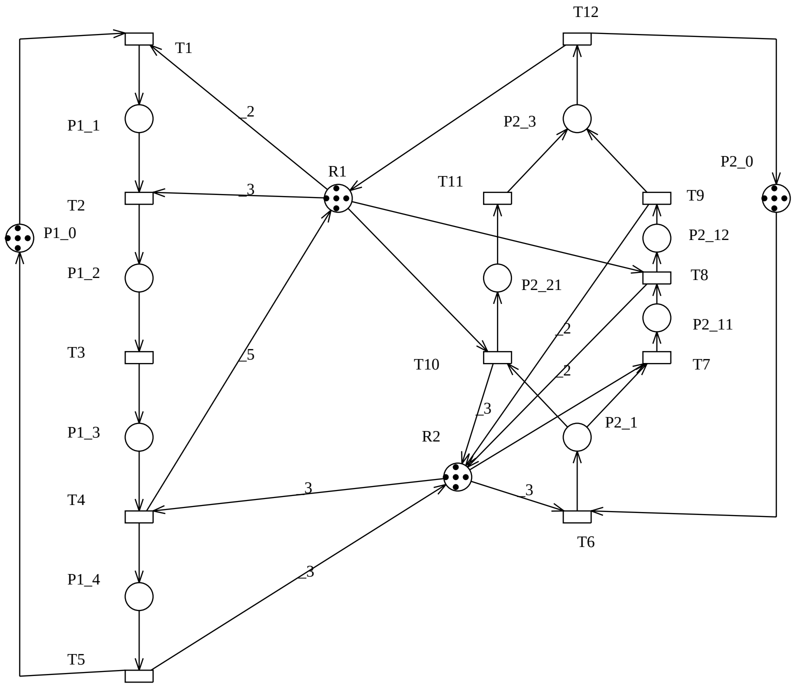


Figure 3.5: A net with some deadlock problems

3.3 An iterative control policy

Let us present the proposed control policy, implemented in several steps. For this, the characterizations of Theorem 28 and Theorem 32 will be used, together with the net state equation.

Since the problematic markings can be identified, some restrictions can be added to the system in order to make them unreachable. These restrictions should forbid as few states as possible in order to avoid just the detected bad markings (markings of type 2). This has a drawback: if we try to avoid just the bad markings for a siphon, there can appear, as sketched in the previous introductory section, “new” problems. The next objective would be to forbid another bad markings related to another siphon. A way to do that will be to add some linear constraints, implemented as new (virtual) resource places.

This method has a drawback. As proved in [Val99, GVTCE00] in some cases it is not possible to forbid a given marking (or set of markings), and just this given

	Places of the siphon
D1	P2_21 P1_2 P1_3 P2_3 P2_12 R1
D2	P2_21 P2_3 P1_4 P2_12 R1 R2
D3	P1_4 P2_11 P2_12 P2_1 R2

Table 3.1: Minimal siphons related to resources of the net in Figure 3.5

marking, by means of the addition of linear constraints. It will be shown later that for any given marking it will be always possible to add some linear constraint forbidding it, but this will usually forbid some other markings, perhaps even good ones.

The advantage of forbidding bad markings by means of linear constraints is that it can be implemented as virtual resources. This means that, once a given marking has been forbidden (by means of the addition of the adequate control place), the resulting system still belongs to the S^4PR class. Therefore, the method can continue looking for a new bad marking, forbidding it, and so on, in a iterative way. Since the number of (potential) states in the sequence of a partially controlled system is strictly decreasing, the process terminates. Moreover, in each iteration some new resource is added, and then, the language corresponding to the final model is an strict subset of the language of the initial one. In consequence, by construction, no firing sequence can lead to a deadlocked state.

Let us know introduce in the following sections the proposed approach, first by means of an example, and then in a more formal way.

3.3.1 An intuitive approach to the $\mathbb{D}$–deadlocked markings computation

As a first approach, let us assume that a bad siphon is already known (later it will be shown how to compute one of such siphons). Let us consider, for instance, siphon $D2$ shown in Table 3.1, which is a bad siphon of the net of Figure 3.5. According to the notations used in Chapter 2, we can write:

- $D2_S = \{P2\_21, P2\_3, P1\_4, P2\_12\}$,
- $D2_R = \{R1, R2\}$,
- $D2 = D2_S \cup D2_R$,
- $\mathcal{T}h_{D2} = \{P1\_1, P1\_2, P1\_3, P2\_1, P2\_11\}$.

Now, $D2$–deadlocked markings have to be considered. Theorem 32[1] proved that there exits a $D2$–deadlocked markings, $\mathbf{m}_{D2}$, verifying the following set of restrictions.

1. $\mathbf{m}_{D2}[P_S] > 0$:

   $$\mathbf{m}_{D2}[P1\_1] + \mathbf{m}_{D2}[P1\_2] + \mathbf{m}_{D2}[P1\_3] + \mathbf{m}_{D2}[P2\_1] + \mathbf{m}_{D2}[P2\_11] > 0$$

2. $\mathbf{m}_{D2}[P_S \setminus \mathcal{T}h_{D2}] = 0$:

   $\mathbf{m}_{D2}[P2\_21] = 0 \wedge \mathbf{m}_{D2}[P2\_3] = 0 \wedge \mathbf{m}_{D2}[P2\_12] = 0 \wedge \mathbf{m}_{D2}[P1\_4] = 0$

3. $\forall p \in \mathcal{T}h_{D2}$ such that $\mathbf{m}_{D2}[p] > 0$, the firing of each $t \in p^\bullet$ is prevented by a set of resource places belonging to $D2$. These restrictions are:

   $$\begin{array}{lll} A) & \mathbf{m}_{D2}[P1\_1] & > 0 \longrightarrow \quad \mathbf{m}_{D2}[R1] < 3 \\ B) & \mathbf{m}_{D2}[P1\_3] & > 0 \longrightarrow \quad \mathbf{m}_{D2}[R2] < 3 \\ C) & \mathbf{m}_{D2}[P2\_11] & > 0 \longrightarrow \quad \mathbf{m}_{D2}[R1] < 1 \\ D) & \mathbf{m}_{D2}[P2\_1] & > 0 \longrightarrow \quad \mathbf{m}_{D2}[R1] < 1 \ \wedge \ \mathbf{m}_{D2}[R2] < 1 \end{array}$$

In consequence, in a $D2$–deadlocked marking, the following expression must be true:

$(\mathbf{m}_{D2}[P_S \setminus \mathcal{T}h_{D2}] = 0) \ \wedge \ (\mathbf{m}_{D2}[D2_S] = 0) \ \wedge \ (\mathbf{m}_{D2}[P_S] > 0) \ \wedge \ (A \ \wedge \ B \ \wedge \ C \ \wedge \ D)$

For example, for the $D2$–deadlocked marking $\mathbf{m2} = 4{\cdot}P1\_0 + P1\_3 + 4{\cdot}P2\_0 + P2\_11 + R2$, the following facts hold:

1. Since $(\mathbf{m2}[P1\_3] > 0) \wedge (\mathbf{m2}[P2\_11] > 0)$, then $\mathbf{m2}[P1\_1] + \mathbf{m2}[P1\_2] + \mathbf{m2}[P1\_3] + \mathbf{m2}[P2\_1] + \mathbf{m2}[P2\_11] > 0$.

2. $P_S \setminus \mathcal{T}h_{D2} = D2_S = \{P2\_21, P2\_3, P1\_4, P2\_12\}$, and $\mathbf{m2}[P_S \setminus \mathcal{T}h_{D2}] = 0$.

3. Finally, $\mathbf{m2}[P1\_3] > 0 \wedge \mathbf{m2}[R2] < 3$, and $\mathbf{m2}[P2\_11] > 0 \wedge \mathbf{m2}[R1] < 1$

[1] We can use this theorem here because the siphon is known. We will see later that it is more convenient to use Theorem 28 when the siphon is not known.

A deadlock situation relates a bad marking and a bad siphon. It is usual that more than one bad marking can be associated to the same siphon. In order to add less control elements, we are looking for a method able to control at once all the bad markings associated to a given bad siphon. Deadlock situations can be viewed from two different points of view:

1. Too many siphon resources are used simultaneously.

   In $\mathbf{m2}$ there are not enough available resources in $D2_R$ to allow processes in $\mathcal{T}h_{D2}$ to advance. There is just one free copy of $R2$, while $P1\_3$ needs three free copies of $R2$ in order to evolve, and $P2\_11$ needs one free copy of $R1$. The computation of the maximal number of total tokens in $D2_R$ for $D2$–deadlocked markings like $\mathbf{m2}$ will give a bound for the number of tokens allowed to be used by places of $\mathcal{T}h_{D2}$; let us call m_{D2}^{max} such number.

   No $D2$–deadlocked marking can occur if the active processes in $\mathcal{T}h_{D2}$ are using less than $\mathbf{m_0}[D] - m_{D2}^{max}$ tokens of $D2_R$ (that is, they leave more than m_{D2}^{max} free resources in $D2_R$). Notice that m_{D2}^{max} is finite (each place in a S^4PR is in at least one P–Semiflow). In consequence, if the system evolutions are controlled in such a way that processes in $\mathcal{T}h_{D2}$ always leave more free resources than this number, there will be no $D2$–deadlocked markings.

   The value of m_{D2}^{max} can be computed as the solution of the following integer linear programming problem (ILPP):

$$
\begin{aligned}
m_{D2}^{max} = \text{ Maximize } & \sum_{r \in D2_R} \mathbf{m}[r] \\
\text{Subject to } & \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \overline{\boldsymbol{\sigma}} \\
& \overline{\boldsymbol{\sigma}} \geq 0, \mathbf{m} \geq 0, \\
& \mathbf{m}[P_S] > 0 \ \wedge \ \mathbf{m}[P_S \setminus \mathcal{T}h_{D2}] = 0, \\
& A \ \wedge \ B \ \wedge \ C \ \wedge \ D
\end{aligned}
$$

   Notice that since the set of reachable markings is not known (and we do not want to have to compute it explicitly) we will use the set of potentially reachable markings (the solutions of the state equation). The value m_{D2}^{max} obtained in this way could be strictly greater than the real (reachable) one. But since the objective is to find an upper bound of the number of tokens that remain in the siphon for $D2$–deadlocked markings, the obtained value can be used to control the system.

2. Too many active processes are using the siphon resources.

   Alternatively, the problem can be seen as the existence of too many active processes in $\mathcal{T}h_{D2}$ using resources of $D2$. For example, at marking $\mathbf{m2}$ the two active processes are at $P1\_3$ and $P2\_11$, respectively. $P1\_3$ is using five copies of $R1$, and $P2\_11$ is using four copies of $R2$.

   The computation of the minimal number of total tokens in $\mathcal{T}h_{D2}$ for D–deadlocked markings like $\mathbf{m2}$ will give a bound for the number of processes allowed to enter in this part of the system; let us call m_{D2}^{min} such minimum.

   It is now clear that if the system evolutions are restricted in such a way that the total number of tokens in $\mathcal{T}h_{D2}$ is always strictly less than m_{D2}^{min}, no $D2$–deadlocked markings will be reachable. The value of m_{D2}^{min} can be obtained as the solution of the following ILPP:

$$\begin{aligned} m_{D2}^{min} = \text{ Minimize } & \sum_{p \in \mathcal{T}h_{D2}} \mathbf{m}[p] \\ \text{Subject to } & \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \overline{\boldsymbol{\sigma}} \\ & \overline{\boldsymbol{\sigma}} \geq 0, \mathbf{m} \geq 0, \\ & \mathbf{m}[P_S] > 0 \;\wedge\; \mathbf{m}[P_S \setminus \mathcal{T}h_{D2}] = 0, \\ & A \;\wedge\; B \;\wedge\; C \;\wedge\; D \end{aligned}$$

From the integer linear programming problems presented above, a solution for m_{D2}^{max} (alternatively, m_{D2}^{min}) will be obtained.

The token conservation properties induced by the set of P–Semiflows clearly show a relationship between both points of view, even though they are different and lead to different solutions.

Remember that in a $D2$–deadlocked marking, an active process in place $p \in \mathcal{T}h_{D2}$ uses $\mathbf{Y_{D2_R}}[p]$ $(= \sum_{r \in D2_R} \mathbf{Y_r}[p])$ tokens of $D2_R$. This fact can be used in two different ways to prevent $D2$–deadlocked markings.

1. The control policy will need to ensure that at each reachable state,the places of $\mathcal{T}h_{D2}$ are not using too many resources of $D2$. For this, a new place $pD2\_R$ can be added, in such a way that its marking is equal to the number of tokens of $D2_R$ engaged by processes in $\mathcal{T}h_{D2}$. For example, each active process at place $P1\_1$ uses two units of resource $R1$, while each active process in $P1\_2$ and $P1\_3$ uses five units of this resource. On the other side, each process in $P2\_1$ uses three units of $R2$, and each process in $P2\_11$, four.

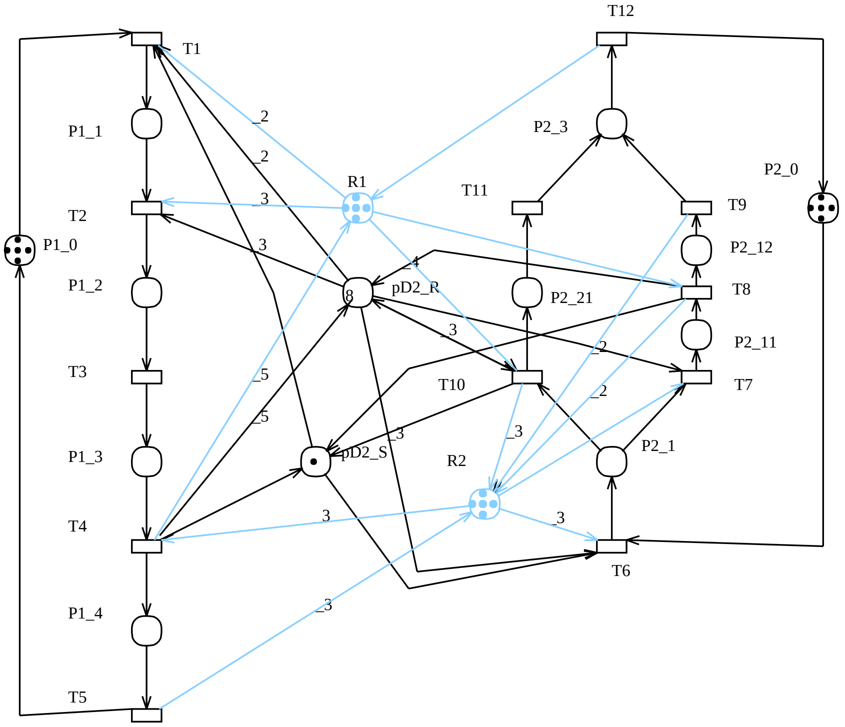


Figure 3.6: $pD2\_R$ and $pD2\_S$ are two possible control places for siphon $D2$

The initial marking for this place will be $\mathbf{m_0}[D_R] - (m_{D2}^{max} + 1)$ to ensure that these places are not holding too many copies of the resources. Technically speaking, it can be said that $pD2\_R$ and $\mathcal{T}h_{D2}$ induce an invariant relation (established by means of a new P–Semiflow) constraining the system evolution so that the number of tokens of $D2_R$ used by places of $\mathcal{T}h_{D2}$ is less than $\mathbf{m_0}[D_R] - (m_{D2}^{max} + 1)$; that is, active processes in $\mathcal{T}h_{D2}$ are not using enough resources of $D2_R$ to reach a $D2$–deadlocked marking.

In the example, since $m_{D2}^{max} = 1$, $\mathbf{m_0}[D] - \sum_{p \in D2} \mathbf{m}[p] \cdot \mathbf{Y_{D2_R}}[p] \geq 2$ can be imposed for any reachable marking. To do that, place $pD2\_R$ can be added with initial marking equal to $10 - 2 = 8$. (Definition 43 will show how this place is computed).

The addition of $pD2\_R$ generates the following marking invariant:

$$\begin{aligned}\mathbf{m_0}[pD2\_R] = 8 \\ &= \mathbf{m}[pD2\_R] \\ &\quad + 2 \cdot \mathbf{m}[P1\_1] + 5 \cdot (\mathbf{m}[P1\_2] + \mathbf{m}[P1\_3]) \\ &\quad + 3 \cdot \mathbf{m}[P2\_1] + 4 \cdot \mathbf{m}[P2\_11]\end{aligned}$$

This P–Semiflow is forbidding the previously cited marking $\mathbf{m2}$. The reason is that, at $\mathbf{m2}$,

$$5 \cdot \mathbf{m2}[P1\_3] + 4 \cdot \mathbf{m2}[P2\_11] = 5 \cdot 1 + 4 \cdot 1 = 9 > 8 (= \mathbf{m_0}[pD2\_R])$$

In the same way, any other $D2$–deadlocked markings is also forbidden.

2. When considering the processes point of view, we will need to ensure that there are not "too many" active processes in $\mathcal{T}h_{D2}$. For this, place $pD2\_S$ can be added, in such a way that its marking is equal to the number of such active processes. This is a more simple approach, since counting the number of tokens that enter in relevant places is quite easy to implement.

   The initial marking for this place will be $m_{D2}^{min} - 1$, generating an invariant relation similar to the one shown in the previous case, ensuring that no more than this number of parts can enter $\mathcal{T}h_{D2}$.

   In the example, since $m_{D2}^{min} = 2$, we want to impose that for each marking, $\sum_{p \in \mathcal{T}h_{D2}} \mathbf{m}[p] < 2$. To do that, place $pD2\_S$ is added with marking 1. (Definition 43 establishes how this place is added.)

   This place generates the following new marking invariant:

$$\begin{aligned}\mathbf{m_0}[pD2\_S] = 1 \\ &= \mathbf{m}[pD2\_S] \\ &\quad + \mathbf{m}[P1\_1] + \mathbf{m}[P1\_2] + \mathbf{m}[P1\_3] \\ &\quad + \mathbf{m}[P2\_1] + \mathbf{m}[P2\_11]\end{aligned}$$

   This P–Semiflow is forbidding marking $\mathbf{m2}$. This is so because at $\mathbf{m2}$,

$$\mathbf{m2}[P1\_3] + \mathbf{m2}[P2\_11] = 1 + 1 = 2 > 1$$

   Of course, it also forbids any other $D2$–deadlocked marking.

Given a siphon, $D2$, two possible control places have been suggested. Obviously, just one of them is needed for each bad siphon. Let us present some final remarks to this intuitive introduction:

- A control place (using any of the two alternative approaches) is computed from a given bad siphon. Then, to control the whole system, a first direct solution would consist in computing all the bad siphons and then to control each one of them. This can be very time consuming. A different approach is going to be used: a bad siphon is computed and controlled, then a second one, and so on. In general, this approach will reach a controlled system in a faster way, since it is possible that controlling one bad siphon, other siphons become also controlled at the same time.

- Controlling all the bad siphons of the original net can be insufficient to ensure a live behavior. As previously stated, markings of type 2 can be the source of new problems. From the structure point of view this fact will be shown by (new) bad siphons involving some of the added control places. Luckily, each time a control place is added, the controlled net also belongs to S^4PR class. This allows to follow an iterative approach. Moreover, since the addition of a control place forbids some (potentially) reachable marking and the set of (potentially) reachable markings is finite, it is ensured that the iterative method terminates.

- Two alternative ways of controlling a bad siphon have been presented. They both are adequate to control a given siphon, but they are not equivalent, since the set of reachable markings (after the addition of the control place) can be different. Let us consider again $\mathbf{m2}$. If $D2$ is controlled using the resource point of view, the obtained system can reach 138 states. If it is controlled using the process point of view, only 98 states are reachable in the resulting system.

- The method starts computing and controlling a bad siphon, using an integer linear programming problem. If such problem has no solution, the system is live and no control is needed. This is an important advantage of the proposed deadlock prevention method, when it is compared with any deadlock avoidance method.

In the following all these ideas will be presented in a formal way.

3.3.2 Computation of deadlocked markings

The following proposition relates liveness with the existence of a solution for the presented system of inequalities. The systems is a linear representation of a bad marking given a known bad siphon introduced in the statement of Theorem 28.

Proposition 33 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^4PR. 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:*

$$\begin{cases} \mathbf{m}[P_S] > 0 & \\ \forall t \in T \setminus {P_0}^\bullet, & \text{being } \{p\} = {}^\bullet t \cap P_S, \\ & \mathbf{m}[p] \geq e_t \\ & e_t \geq \mathbf{m}[p]/\mathbf{sb}[p] \\ \forall r \in D_R, & \\ \forall t \in r^\bullet \setminus {P_0}^\bullet, & \mathbf{m}[r]/\mathbf{Pre}[r,t] \geq e_{rt} \\ & e_{rt} \geq (\mathbf{m}[r] - \mathbf{Pre}[r,t] + 1)/(\mathbf{m_0}[r] - \mathbf{Pre}[r,t] + 1) \\ \forall r \in P_R \setminus D_R, & \forall t \in r^\bullet \setminus {P_0}^\bullet, e_{rt} = 1 \\ \forall t \in T \setminus {P_0}^\bullet, & \sum_{r \in {}^\bullet t \cap P_R} e_{rt} < |{}^\bullet t \cap P_R| + 1 - e_t \\ \forall t \in T \setminus {P_0}^\bullet, & e_t \in \{0,1\} \\ \forall r \in P_R, & \forall t \in r^\bullet \setminus {P_0}^\bullet, e_{rt} \in \{0,1\} \end{cases} \tag{3.1}$$

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

Proof

First of all, let us make some comments about the variables used in these inequalities.

1. *For each $t \in T \setminus {P_0}^\bullet$, e_t indicates whether t is $\mathbf{m}$–process–enabled or not. It follows immediately from the following facts:*
   - *since $\mathbf{sb}[p] > 0$, $\mathbf{m}[p] > 0$ if, and only if, $\mathbf{m}[p]/\mathbf{sb}[p] > 0$, which is equivalent to state that $e_t = 1$ (remember that $e_t \in \{0,1\}$)*
   - *$\mathbf{m}[p] = 0$ if, and only if, $e_t = 0$*
2. *Let $r \in D_R$, and $t \in r^\bullet \setminus {P_0}^\bullet$. Let us prove that e_{rt} indicates whether t is enabled by r at $\mathbf{m}$:*
   - *If t is enabled by r at $\mathbf{m}$($\mathbf{m}[r] \geq \mathbf{Pre}[r,t]$), $\mathbf{m}[r]/\mathbf{Pre}[r,t] \geq 1$ and $1 \geq (\mathbf{m}[r] - \mathbf{Pre}[r,t] + 1)/(\mathbf{m_0}[r] - \mathbf{Pre}[r,t] + 1) > 0$; therefore, e_{rt} must be 1.*

- *If t is not enabled by r ($\mathbf{m}[r] < \mathbf{Pre}[r,t]$), $\mathbf{m}[r]/\mathbf{Pre}[r,t] < 1$ and $(\mathbf{m}[r] - \mathbf{Pre}[r,t] + 1)/(\mathbf{m_0}[r] - \mathbf{Pre}[r,t] + 1) \leq 0$; then, e_{rt} must be 0.*

3. *If $r \in P_R \setminus D_R$, and $t \in r^\bullet \setminus {P_0}^\bullet$, $e_{rt} = 1$ (that is, resources not belonging to the siphon enable their output transitions).*

4. *The system of inequalities without the last one has always a solution, and the value of variables e_t, 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 $t \in T \setminus {P_0}^\bullet$ is not $\mathbf{m}$–process–enabled, $e_t = 0$ and the inequality for t is trivially fulfilled, because $\sum_{r \in {}^\bullet t \cap D_R} e_{rt} \leq |{}^\bullet t \cap D_R|$.*
   - *If $t \in T \setminus {P_0}^\bullet$ is $\mathbf{m}$–process–enabled, $e_t = 1$ and the inequality becomes $\sum_{r \in {}^\bullet t \cap D_R} e_{rt} < |{}^\bullet t \cap D_R|$.*
     Therefore, there is a solution if, and only if, $\exists r \in D_R \cap {}^\bullet t$ such that t is not enabled by r.

Let us use these points in order to show the truth of the Theorem.

$\Longrightarrow$) If the net is non–live, Theorem 28 ensures that there exists a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, with $\mathbf{m}[P_S] > 0$, and a siphon D such that the firing of each $\mathbf{m}$–process–enabled transition is prevented by a set of resource places belonging to D.

This means that there exist places with $e_t = 1$. Since each one of these transitions is prevented by a set of resource places belonging to D, there exists $r \in {}^\bullet t \cap D_R$ such that $\mathbf{m}[r] < \mathbf{Pre}[r,t]$, and then, $e_{rt} = 0$.

In consequence, for these transitions, $\sum_{r \in {}^\bullet t \cap P_R} e_{rt} < |{}^\bullet t \cap P_R| + 1 - e_t$, is true.

$\Longleftarrow$)

Let us consider D, $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, and the set of variables $\{e_r \mid r \in P_R\}$ and $\{e_{rt} \mid t \in r^\bullet \setminus {P_0}^\bullet\}$, solutions of the set of inequalities.

Since $\mathbf{m}[P_S] > 0$, let $p \in P_S$ such that $\mathbf{m}[p] > 0$. For each $t \in p^\bullet$, $e_t = 1$, and then, $\sum_{r \in {}^\bullet t \cap P_R} e_{rt} < |{}^\bullet t \cap P_R| + 1 - e_t = |{}^\bullet t \cap P_R|$. Therefore, there must exist $r \in {}^\bullet t \cap P_R$ such that $e_{rt} = 0$ which means that t is $\mathbf{m}$–resource disabled. Moreover, $r \in D_R$ since for each $r \in P_R \setminus D_R$, each $t' \in r^\bullet$, $e_{rt'} = 1$. Therefore, any $\mathbf{m}$–process enabled transition is disabled by a resource place belonging to D_R and Theorem 28 allows us to conclude. □

The existing bad siphons and their related bad markings need to be computed in order to control the system. Our next goal is to reformulate the above system of inequalities in order to be able to obtain a bad siphon, together with its related bad markings. The characterization presented in Theorem 28 allows a simple reformulation of these equations. To do that, an algebraic characterization of siphon is necessary. In [AT85, Sil85] a characterization of this kind is given for traps. It is straightforward to adapt it to the case of siphons.

The result establishes that each solution of the following set of inequalities:

$$v_p \leq \sum_{q \in {}^\bullet t} v_q,\ v_p \in \{0,1\},\ \forall p \in P,\ \forall t \in {}^\bullet p,$$

is a siphon (whose components are those places such that the value of variable v_p equal to 1). As it will become clear later, this result is not adequate in this original form, and it has to be transformed into an equivalent form using negated logic (this approach is similar to the one proposed in [Sil85] and also in [XJ99].) A siphon is the set of places whose associated variables in the following set of inequalities is 0:

$$v_p \geq \sum_{q \in {}^\bullet t} v_q - |{}^\bullet t| + 1,\ v_p \in \{0,1\},\ \forall p \in P,\ \forall t \in {}^\bullet p$$

In order to compute a bad siphon, conditions of Proposition 33 can be completed by the addition of the following equations:

- A set of constraints representing the siphon,

$$v_p \geq \sum_{q \in {}^\bullet t} v_q - |{}^\bullet t| + 1,\ v_p \in \{0,1\},\ \forall p \in P,\ \forall t \in {}^\bullet p$$

- A restriction that avoids the whole net as solution:

$$\sum_{p \in P \setminus P_0} v_p < |P \setminus P_0|$$

- A set of restrictions relating resource places that are avoiding the firing of a process–enabled transition and the siphon. For this, e_t, e_{rt}, as in previous proposition are used together with the new introduced variables.

Let us show how this extension can be used to compute bad siphons and related bad markings.

Proposition 34 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^4PR. 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 system of inequalities (3.2) has a solution with $D = \{p \in P_S \cup P_R \mid v_p = 0\}$:*

$$\left\{\begin{array}{ll}
\multicolumn{2}{l}{\forall p \in P \setminus P_0, \forall t \in {}^{\bullet}p, v_p \geq \sum_{q \in {}^{\bullet}t} v_q - |{}^{\bullet}t| + 1} \\
\multicolumn{2}{l}{\sum_{p \in P \setminus P_0} v_p < |P \setminus P_0|} \\
\multicolumn{2}{l}{\mathbf{m}[P_S] > 0} \\
\forall t \in T \setminus {P_0}^{\bullet}, & \textit{being } \{p\} = {}^{\bullet}t \cap P_S, \\
 & \mathbf{m}[p] \geq e_t \\
 & e_t \geq \mathbf{m}[p]/\mathbf{sb}[p] \\
\forall r \in P_R, & \\
\forall t \in r^{\bullet} \setminus {P_0}^{\bullet}, & \mathbf{m}[r]/\mathbf{Pre}[r,t] + v_r \geq e_{rt} \\
 & e_{rt} \geq (\mathbf{m}[r] - \mathbf{Pre}[r,t] + 1)/(\mathbf{m_0}[r] - \mathbf{Pre}[r,t] + 1) \\
 & e_{rt} \geq v_r \\
\forall t \in T \setminus {P_0}^{\bullet}, & \sum_{r \in {}^{\bullet}t \cap P_R} e_{rt} < |{}^{\bullet}t \cap P_R| + 1 - e_t \\
\forall p \in P \setminus P_0, & v_p \in \{0,1\} \\
\forall t \in T \setminus {P_0}^{\bullet}, & e_t \in \{0,1\} \\
\forall r \in P_R, & \forall t \in r^{\bullet} \setminus {P_0}^{\bullet}, e_{rt} \in \{0,1\}
\end{array}\right. \tag{3.2}$$

Proof

The truth of this proposition is immediate taking into account Proposition 33 and the following considerations:

1. *The two first inequalities, and the fact that $v_p \in \{0,1\}$, define a non–empty siphon as stated before. Let D be such siphon.*
2. *The inequality related to the marking of P_S is the same as in Proposition 33.*
3. *The inequalities related to e_t are the same as in Proposition 33. Remember that $e_t = 1$ if, and only if, t is $\mathbf{m}$–process–enabled.*
4. *The inequalities involving e_{rt} are:*
    - *The same as in Proposition 33 when $v_r = 0$; that is, if resource r belongs to the siphon D. In this case, restriction $e_{rt} \geq v_r$ becomes $e_{rt} \geq 0$, which is redundant.*
    - *If resource $r \in P_R$ does not belong to D, ($v_r = 1$), these inequalities make e_{rt} to be 1. They become:*
        - *$A \geq e_{rt}$, with $A \geq 1$*
        - *$e_{rt} \geq B$, with $B \in [(-Pre[r,t] + 1)/(\mathbf{m_0}[p] - Pre[r,t] + 1), 1]$, and*

- $e_{rt} \geq 1$

  These inequalities always have a unique solution, $e_{rt} = 1$. (Notice that in Proposition 33, e_{rt} was explicitly made equal to 1 for $r \in P_R \setminus D_R$, because the siphon was known a priori.)

5. *The last inequality is the same as in Proposition 33, and its meaning is exactly the same (see the proof of the previous proposition for details).*

□

The characterization introduced in this proposition is not directly applicable to control the system, since a reachable marking is needed and we do not want to use reachable markings (our goal is to avoid the enumeration of the set of reachable markings). Therefore, we are going to propose an alternative approach using the set of potentially reachable markings (markings obtained as solutions of the state equation).

Proposition 35 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^4PR. If net is non–live, there exists a marking $\mathbf{m} \in \mathrm{PRS}(\mathcal{N}, \mathbf{m_0})$, with $\mathbf{m}[P_S] > 0$, and a siphon D such that the following system of inequalities has, at least, one solution with $D = \{p \in P_S \cup P_R \mid v_p = 0\}$:*

$$\begin{cases} \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \overline{\boldsymbol{\sigma}} \\ \mathbf{m} \geq 0,\ \overline{\boldsymbol{\sigma}} \in \mathbb{Z}_+^{|T|} \\ System\ (3.2) \end{cases} \tag{3.3}$$

□

This proposition does not provide a complete characterization (as it was the case in Proposition 34). It only provides a necessary condition for deadlock. The reason is the existence of spurious solutions: markings that are solution of the state equation but are not reachable. This is not a problem when the objective is to control the system looking for a live system: the only consequence can be that control places also forbid some marking which are not reachable. In this way, a system with more control than needed can be obtained which will be, in any case, live.

Finally, it must also be pointed out that, when possible, adding non necessary or redundant control should be avoided, since some good markings can be eliminated together with the bad ones, which is not desirable.

Note 36 *A siphon and the corresponding marking fulfilling conditions in Proposition 35 will be called a* potential bad siphon *and a* potential D–deadlocked marking, *respectively. However, and for the sake of simplicity, they will be called bad siphon and D–deadlocked marking.* □

The approach we are going to propose does not obtain all the solutions of the system of Proposition 35. The considered method will obtain a bad siphon, for later controlling it by means of the addition of the adequate place, and iteratively continue computing and controlling new bad siphons. The reason for this is clear: the added control will modify the system behavior and some bad markings associated to another siphons can be forbidden (also some good states). The obtained system will have different deadlock problems than the original one.

To do that we are going to transform the system of equations into another one that will obtain just one siphon as solution. This raises the question of how to decide which siphon to control. The proposed approach selects the siphon with a minimal number of places in the hope that controlling first smaller siphons may help to avoid the control of the bigger ones. The following corollary introduces the problem.

Corollary 37 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^4PR. 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 at least, one solution with $D = \{p \in P_S \cup P_R \mid v_p = 0\}$:*

$$\begin{cases} \textit{maximize} & \sum_{p \in P \setminus P_0} v_p \\ \textit{s.t.} & System\ (3.3) \end{cases} \tag{3.4}$$

□

The solution of this problem is a bad siphon, D, and a D–deadlocked marking, $\mathbf{m}$. No special consideration has been done about the D–deadlocked marking associated to the siphon, while some restrictions about minimality have been done for D.

Nevertheless, we do not want to avoid only just this D–deadlocked marking but also all the deadlocked markings related to the siphon. In consequence, a new problem needs to be solved: once the siphon is known, which are the deadlocked markings for it? This question has a clear theoretical interest but, if we look at it carefully, we discover that its practical application can be very expensive, depending on the number of such markings.

The approach considered here is to compute some selected 'representative' markings that can be used to avoid all the related D–deadlocked markings. This will be accomplished here in two alternative ways, as presented in the intuitive introduction:

- Looking at the maximal number of resources available at D–deadlocked markings.
- Looking at the minimal number of active processes at D–deadlocked markings.

For this, the characterization of Theorem 32 (page 64) and Proposition 31 (page 62) will be used, that allows us concentrate on different markings, once a bad siphon is known. In this sense, it will be useful to return to Proposition 33 (page 33). The equations presented there were constructed supposing that the siphon was known. Let us use them in order to construct the associated D–restrictions. The restriction $\mathbf{m}[P_S \setminus \mathcal{T}h_D] = 0$ from Theorem 32 can be added since the siphon is now known.

Definition 38 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^4PR. Let D be a bad siphon. The* set of D–restrictions *is:*

$$\begin{cases} \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \overline{\boldsymbol{\sigma}} \\ \mathbf{m} \geq 0,\ \overline{\boldsymbol{\sigma}} \in \mathbb{Z}_{+}^{|T|} \\ \mathbf{m}[P_S \setminus \mathcal{T}h_D] = 0 \\ System\ (3.1) \end{cases} \tag{3.5}$$

□

These restrictions represent the conditions related to the ones shown in Theorem 32 once the bad siphon is known. With them, we can now select the adequate bad markings.

Definition 39 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^4PR. Let D be a bad siphon, m_D^{max} and m_D^{min} are defined as follows:*

$$\begin{array}{llll} ILPP_D^{max}: & m_D^{max} = & \textit{maximize} & \sum_{r \in D_R} \mathbf{m}[r] \\ & \textit{s.t.} & \textit{restrictions}\ (3.5) & \end{array}$$

$$\begin{array}{llll} ILPP_D^{max}: & m_D^{min} = & \textit{minimize} & \sum_{p \in \mathcal{T}h_D} \mathbf{m}[p] \\ & \textit{s.t.} & \textit{restrictions } (3.5) & \end{array}$$

□

Note 40 *These two problems are, in some way, equivalent: either both have solution or none of them has solution: they search for deadlocked markings, concentrating on different points of view. That is, while m_D^{max} looks at the number of tokens in D_R at deadlocked markings, m_D^{min} looks at the number of active processes in places belonging to $\mathcal{T}h_D$ that are "stealing" tokens from D at deadlocked markings.*

When referring to a particular $ILPP$ problem of the ones presented in Definition 39, $ILPP_D^{max}$ or $ILPP_D^{min}$ will be used. When referring to any of them $ILPP_D$ will be used. □

The way to control these systems in order to avoid deadlock problems is based on the addition of control places, as sketched in the intuitive introduction. Let us present some terminology to deal with this.

Note 41 *Once a bad siphon D has been computed, it can be controlled using $ILPP_D^{max}$ or $ILPP_D^{min}$ in order to prevent D–deadlocked markings in two different ways:*

- *Adding one control place ensuring that processes in $\mathcal{T}h_D$ are not using more resources than $\mathbf{m_0}[D_R] - m_D^{max}$. If this is the adopted approach (called the* D–resource approach*), the system will be said to be D–*resource–controlled.
- *Adding a control place ensuring that there will be no more than $m_D^{min} - 1$ active process in places belonging to $\mathcal{T}h_D$. If this is the adopted approach (called the* D–process approach*), the system will be said to be D–*process–controlled.

*If the adopted method is not specified, the resulting system will be said to be D–*controlled. □

Let us now present a basic property that will be needed later in order to see that the added control is correct. The result is related to the minimal number of active processes at a deadlocked state.

Lemma 42 *Let* $\langle\mathcal{N},\mathbf{m_0}\rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$*, be a non–live* S^4PR*, and let* D *and* $\mathbf{m}_D$ *as in Proposition 35. Then,* $\mathbf{m}_D[\mathcal{T}h_D] > 1$.

Proof

Since $\mathbf{m}_D[\mathcal{T}h_D] \geq 0$*, it is enough to see that it cannot be neither 0 nor 1.*

If $\mathbf{m}_D[\mathcal{T}h_D] = 0$*, then* $\mathbf{m}_D[P_S] = 0$ *which is a contradiction with Theorem 32.*

If $\mathbf{m}_D[\mathcal{T}h_D] = 1$*,* $\mathbf{m}_D$ *is a potentially reachable marking with just one token in* P_S*. This marking must be reachable, and cannot be blocked since the initial marking is acceptable.* □

Obviously, this property is also true for reachable markings.

The following definition shows how to add a control place that prevents D–deadlocked markings for a given bad siphon D.

Definition 43 *Let* $\langle\mathcal{N},\mathbf{m_0}\rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$*, be a non–live* S^4PR*. Let* D *be a bad siphon, and* m_D^{max} *and* m_D^{min} *as in Definition 39. Then,*

- *The associated* D*–resource place,* p_D*, is defined by means of the addition of the following incidence matrix row and initial marking:*

$$\begin{array}{rcl} \mathbf{C^{p_D}}[p_D,T] & = & -\sum_{p\in\mathcal{T}h_D}\mathbf{Y_{D_R}}[p]\cdot\mathbf{C}[p,T] \\ \mathbf{m_0}^{p_D}[p_D] & = & \mathbf{m_0}[D]-(m_D^{max}+1) \end{array}$$

- *The associated* D*–process place,* p_D*, is defined by means of the addition of the following incidence matrix row and initial marking:*

$$\begin{array}{rcl} \mathbf{C^{p_D}}[p_D,T] & = & -\sum_{p\in\mathcal{T}h_D}\mathbf{C}[p,T], \\ \mathbf{m_0}^{p_D}[p_D] & = & m_D^{min}-1 \end{array}$$

□

To exemplify the previous definition, let us come back to the S^4PR in Figure 2.6, whose incidence matrix is shown in Table 3.2. $D1 = \{P1\_3,\ P2\_1,\ R2\}$ was a bad siphon.

According to Definition 43, two different control places can be added:

- $D1$–resource place:

	T1	T2	T3	T4	T5	T6	T7
P1_1	1	-1	0	0	0	0	0
P1_2	0	1	-1	0	0	0	0
P1_3	0	0	1	-1	0	0	0
P2_1	0	0	0	0	1	-1	0
P2_2	0	0	0	0	0	1	-1
P1_0	-1	0	0	1	0	0	0
P2_0	0	0	0	0	-1	0	1
R1	-1	1	0	0	0	-1	1
R2	0	-2	-3	5	-1	1	0

Table 3.2: Incidence matrix of net of Figure 2.6

  - $$\begin{aligned}\mathbf{C}^{pD1\_R}[pD1\_R,T] &= -\textstyle\sum_{p\in\{P1\_2\}} 2\cdot\mathbf{C}[P1\_2,T] = \\ &= -2\cdot[0,1,-1,0,0,0,0] = \\ &= [0,-2,2,0,0,0,0]\end{aligned}$$
  - $$\begin{aligned}\mathbf{m_0}^{pD1\_R}[pD1\_R] &= \mathbf{m_0}[D]-(m^{max}_{D1}+1) \\ &= 5-(1+1) \\ &= 3\end{aligned}$$

- $D1$–process place
  - $$\begin{aligned}\mathbf{C}^{pD1\_S}[pD1\_S,T] &= -\textstyle\sum_{p\in\{P1\_2\}}\mathbf{C}[P1\_2,T] = \\ &= -[0,1,-1,0,0,0,0] = \\ &= [0,-1,1,0,0,0,0]\end{aligned}$$
  - $$\begin{aligned}\mathbf{m_0}^{pD1\_S}[pD1\_S] &= m^{min}_{D1}-1 \\ &= 2-1 \\ &= 1\end{aligned}$$

Figure 3.7 shows these places.

Now, two important properties need to be proved for the added places. First, we are going to show that the initial markings for D–control places are non–negative (this is needed to ensure that the D–controlled net is a well–defined Petri net). As a second step, it will be shown that the added place can be seen as a new (virtual) resource (this is needed in order to apply an iterative method). For this

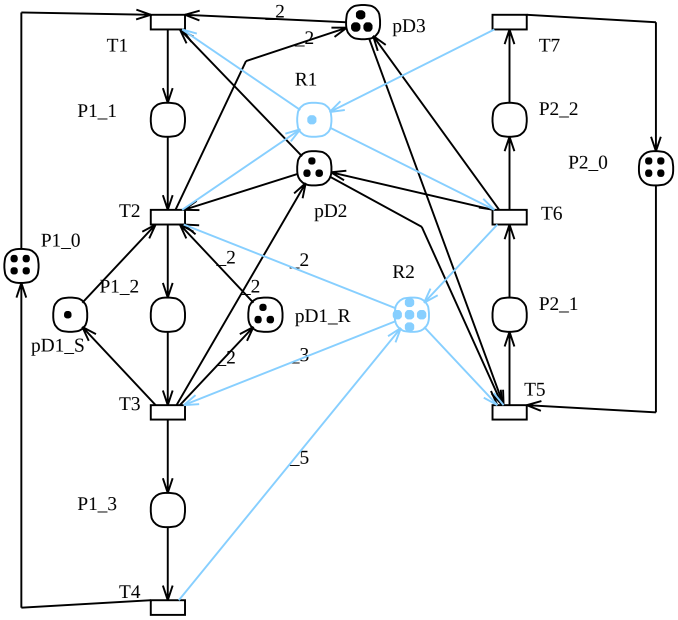


Figure 3.7: Control places proposed for the bad siphon $D1$ of the net in Figure 2.6.

second property, two things are needed: p_D must verify structure conditions to be a resource, and the (extended) marking must be acceptable in the resulting S^4PR.

Lemma 44 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a non–live S^4PR. Let D be a bad siphon, and m_D^{max} and m_D^{min} as in Definition 39. Let $\langle \mathcal{N}^{p_D}, \mathbf{m_0}^{p_D} \rangle$, $\mathcal{N}^{p_D} = \langle P_0 \cup P_S \cup P_R \cup \{p_D\}, T, \mathbf{C}^{p_D} \rangle$ be the net obtained by the addition of the D–process place or the D–resource place. Then, $\mathbf{m_0}^{p_D}[p_D] > 0$.*

Proof

We must check two cases.

- *Addition of a D–resource place.* $\mathbf{m_0}^{p_D}[p_D] = \mathbf{m_0}[D] - (m_D^{max} + 1) > 0$

  Using Lemma 42, at any D–deadlocked marking, $\mathbf{m}_D[\mathcal{T}h_D] > 1$.

  Then, $\mathbf{m}_D[D_R] < \mathbf{m_0}[D_R] - 1$ *(each place of $\mathcal{T}h_D$ is holder of, at least, one resource of D_R). In consequence,* $m_D^{max} < \mathbf{m_0}[D] - 1$.

  Then, $\mathbf{m_0}^{p_D}[p_D] = \mathbf{m_0}[D] - (m_D^{max} + 1) > \mathbf{m_0}[D] - (\mathbf{m_0}[D] - 1 + 1) = 0$.

- *Addition of a D–process place.* $\mathbf{m_0}^{p_D}[p_D] = m_D^{min} - 1$

  Using Lemma 42, at any D–deadlocked marking, $\mathbf{m}_D[\mathcal{T}h_D] > 1$*. In consequence,* $m_D^{min} > 1$*, and then,* $\mathbf{m_0}^{p_D}[p_D] = m_D^{min} - 1 > 0$.

□

Lemma 44 proves that the initial markings computed for a D–resource or a D–process control place are non–negative. Let us consider the net depicted in Figure 3.8, for which $D1 = \{P1\_2, P2\_2, P3\_2, R1, R2\}$ is a bad siphon.

Let us first try to apply the D–resource–control approach. For this siphon, $m_{D1}^{max} = 1$. This would give a control place that induces the marking invariant $2 \cdot \mathbf{m}[P1\_1] + \mathbf{m}[P2\_1] + \mathbf{m}[P3\_1] + \mathbf{m}[p_{D1}] = mo[p_{D1}] = 1$, and this initial marking is not acceptable, since $max_{p \in \mathcal{H}_{p_{D1}}}(\mathbf{Y_{p_{D1}}}[P1\_1]) = 2 > 1$.

However, using the D–process–control approach, $m_{D1}^{min} = 2$. This would give the control place that induces the invariant $\mathbf{m}[P1\_1] + \mathbf{m}[P2\_1] + \mathbf{m}[P3\_1] + \mathbf{m}[p_{D1}] = \mathbf{m_0}[p_{D1}] = 1$. In this case, the marking is acceptable.

This example shows that the initial marking computed for a D–resource place can be non–acceptable. The question is to know which conditions ensure that such markings are acceptable. In Lemma 45 we are going to prove the following facts:

- The initial marking computed for a D–process place is always acceptable. Any D–process place will be also called D–process control place since it is always possible to use it to control the system.

- The initial marking computed for a D–process place will be acceptable if and only if $\mathbf{m_0}^{p_D}[p_D] \geq \max_{p \in \mathcal{T}h_D} \{\mathbf{Y_{D_R}}[p]\}$. Those D–resource places for which the initial marking computed in Definition 39 is acceptable will be called D–resource control places.

Considering this, we can now prove that under these restrictions, the added places are valid from the class definition point of view, and that the resulting system is a marked S^4PR.

Lemma 45 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a non–live S^4PR. Let D be a bad siphon, and m_D^{max} and m_D^{min} as in Definition 39. The net $\langle \mathcal{N}^{p_D}, \mathbf{m_0}^{p_D} \rangle$, $\mathcal{N}^{p_D} = \langle P_0 \cup P_S \cup P_R \cup \{p_D\}, T, \mathbf{C}^{p_D} \rangle$, obtained by the addition of a D–process control place or a D–resource control place is a marked S^4PR.*

Proof

The structure of the net has not changed except for the addition of the D–resource place (resp. D–process place). Let us prove that this added place behaves like a new resource, according to Definition 13. First of all, for each $r \in P_R$, and $p_{0_i} \in P_0$, let us denote $\mathbf{Y_r}^{p_D}$ and $\mathbf{Y_i}^{p_D}$ the immersions of $\mathbf{Y_r}$ and $\mathbf{Y_i}$ of the original net into the controlled system so *that* $\forall p \in P_0 \cup P_S \cup P_R, \mathbf{Y_r}^{p_D}[p] = \mathbf{Y_r}[p]$ *and* $\mathbf{Y_i}^{p_D}[p] = \mathbf{Y_i}[p]$, *while* $\mathbf{Y_r}^{p_D}[p_D] = 0$, *and* $\mathbf{Y_i}^{p_D}[p] = 0$. *We will not use the super–index in order to alleviate the notation.*

Let us see now that the added place, p_D, *has also an associated P–Semiflow. Two cases need to be studied:*

1. *Let* p_D *be a* D*–resource place. Let us define the following non–negative integer vector,* $\mathbf{Y_{p_D}}$*:*

$$\begin{cases} \mathbf{Y_{p_D}}[p_D] & = 1 \\ \mathbf{Y_{p_D}}[p] & = \mathbf{Y_{D_R}}[p], \quad p \in \mathcal{T}h_D \\ \mathbf{Y_{p_D}}[p] & = 0, \qquad\quad p \notin \mathcal{T}h_D \cup \{p_D\} \end{cases}$$

We have that

$$\mathbf{Y_{p_D}} \cdot \mathbf{C^{p_D}} = \mathbf{Y_{p_D}}[p_D] \cdot \mathbf{C^{p_D}}[\mathbf{p_D}, \mathbf{T}] + \sum_{p \in \mathcal{T}h_D} \mathbf{Y_{p_D}}[p] \cdot \mathbf{C^{p_D}}[\mathbf{p}, \mathbf{T}]$$

By Definition 43, we can substitute $\mathbf{C^{p_D}}[\mathbf{p_D}, \mathbf{T}]$, *obtaining:*

$$- \sum_{p \in \mathcal{T}h_D} \mathbf{Y_{D_R}}[p] \cdot \mathbf{C}[p, T] + \sum_{p \in \mathcal{T}h_D} \mathbf{Y_{p_D}}[p] \cdot \mathbf{C^{p_D}}[\mathbf{p}, \mathbf{T}] = 0$$

2. *Let* p_D *be a* D*–process place. Then, let us define the following non–negative integer vector,*

$$\begin{cases} \mathbf{Y_{p_D}}^{p_D}[p_D] & = 1 \\ \mathbf{Y_{p_D}}^{p_D}[p] & = 1, \quad p \in \mathcal{T}h_D \\ \mathbf{Y_{p_D}}^{p_D}[p] & = 0, \quad p \notin \mathcal{T}h_D \cup \{p_D\} \end{cases}$$

For it:

$$\mathbf{Y_{p_D}}^{p_D} \cdot \mathbf{C}^{p_D} = \mathbf{Y_{p_D}}^{p_D}[p_D] \cdot \mathbf{C}^{p_D}[p_D, T] + \sum_{p \in \mathcal{T}h_D} \mathbf{Y_{p_D}}^{p_D}[p] \cdot \mathbf{C}^{p_D}[p, T]$$

By Definition 43, we can substitute $\mathbf{C^{p_D}}[\mathbf{p_D}, \mathbf{T}]$, *obtaining:*

$$- \sum_{p \in \mathcal{T}h_D} \mathbf{C}[p, T] + \sum_{p \in \mathcal{T}h_D} \mathbf{C}^{p_D}[p, T] = - \sum_{p \in \mathcal{T}h_D} \mathbf{C}[p, T] + \sum_{p \in \mathcal{T}h_D} \mathbf{C}[p, T] = 0$$

Therefore, in both cases there exists a P–Semiflow associated to the new added place. Moreover, the chosen P–Semiflow is also minimal. Notice that $\mathbf{Y}_{\mathbf{p_D}}$ *is the only P–Semiflow that has 1 in the component associated to the place* p_D*, so it cannot be described as a linear combination of the other P–Semiflows. Furthermore, it is the only one that contains this place (if there exist two such P–Semiflows, the difference would be also a P–Semiflow and would be strictly included in* P_S *which is not possible). Finally, by the way it is defined,* $\mathbf{Y}_{\mathbf{p_D}}[p_D] = 1$.

Finally, it is important to remark that Lemma 44 and the restrictions imposed on the hypothesis make the chosen initial marking acceptable. □

As a consequence, the added place can be considered as a new (virtual) resource whose holders are $\mathcal{H}_{p_D} = \mathcal{T}h_D$. Let us prove that the addition of any of the considered control places strictly reduces the size of the potentially reachable set.

Lemma 46 *Let* $\langle \mathcal{N}, \mathbf{m_0}\rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$*, be a non–live* S^4PR*. Let* D *be a bad siphon, and* m_D^{max} *and* m_D^{min} *as in Definition 39. Let* $\langle \mathcal{N}^{p_D}, \mathbf{m_0}^{p_D}\rangle$, $\mathcal{N}^{p_D} = \langle P_0 \cup P_S \cup P_R \cup \{p_D\}, T, \mathbf{C}^{p_D}\rangle$*, the* S^4PR *obtained by the addition of the* D*–process place or the* D*–resource place. Then,* $|\mathrm{PRS}(\mathcal{N}^{p_D}, \mathbf{m_0}^{p_D})| < |\mathrm{PRS}(\mathcal{N}, \mathbf{m_0})|$

Proof

Since $\mathrm{PRS}(\mathcal{N}^{p_D}, \mathbf{m_0}^{p_D})$ *corresponds to the solutions of* $\mathrm{PRS}(\mathcal{N}, \mathbf{m_0})$ *plus a new resource added in a conservative way,* $|\mathrm{PRS}(\mathcal{N}^{p_D}, \mathbf{m_0}^{p_D})| \leq |\mathrm{PRS}(\mathcal{N}, \mathbf{m_0})|$*. Moreover,*

- *being* p_D *the* D*–resource place, let* $\mathbf{m}$ *be a* D*–deadlocked marking; let us remember Theorem 32, where it was shown that for any* D*–deadlocked marking,* $\mathbf{m}$*, there exists a marking* $\mathbf{m}_D$ *such that it has the same tokens in* $\mathcal{T}h_D$*, and* $\mathbf{m}_D[D_S] = 0$ *(among other properties). Let us concentrate on these markings, with* $\mathbf{m}[D_S] = 0$.

$$\begin{aligned}
\mathbf{m}[D_R] &= \mathbf{m_0}^{p_D}[D_R] - \sum_{p \in \mathcal{T}h_D} \mathbf{m}[p] \cdot \mathbf{Y_{D_R}}[p] \\
&= \mathbf{m_0}^{p_D}[p_D] + m_D^{max} + 1 - \sum_{p \in \mathcal{T}h_D} \mathbf{m}[p] \cdot \mathbf{Y_{D_R}}[p] \\
&= \mathbf{m}[p_D] + m_D^{max} + 1 \\
&\geq m_D^{max} + 1
\end{aligned}$$

In consequence, no reachable marking with $\mathbf{m}[D_S] = 0$ *will have less than* $m_D^{max} + 1$ *tokens in* D_R*. If the programming problems have solution, that means that such markings exist in the original system, so the size of the reachability set decreases with the addition of the control place.*

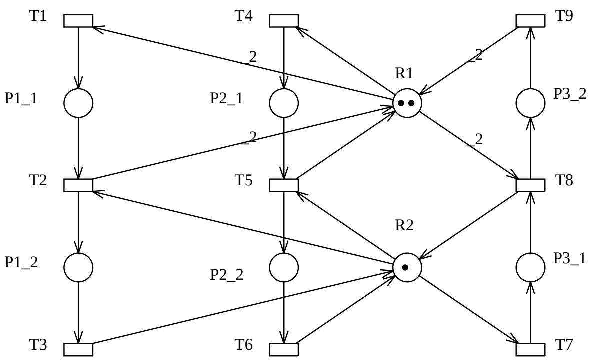


Figure 3.8: A (partial) S^4PR with a bad siphon that is no controllable using the D–resource approach (idle places have been omitted for the sake of clarity).

- *If p_D is a D–process place,*

$$\begin{aligned}\mathbf{m}^{p_D}[\mathcal{T}h_D] &= \mathbf{m_0}^{p_D}[p_D] - \mathbf{m}^{p_D}[p_D]\\ &= m_D^{min} - 1 - \mathbf{m}^{p_D}[p_D]\\ &< m_D^{min}\end{aligned}$$

In consequence, no D–deadlocked marking can exists in $\mathrm{PRS}(\mathcal{N}^{p_D}, \mathbf{m_0}^{p_D})$, since at least one D–deadlocked marking existed in $\mathrm{PRS}(\mathcal{N}, \mathbf{m_0})$ □

In our experience, the use of the D–resource approach gives more permissive controlled systems. In consequence, the approach we are going to propose will try to first apply the D–resource–control; if the resulting initial marking is not acceptable, then apply the D–process–control.

Notice that no potential reachable marking in $\mathrm{PRS}(\mathcal{N}^{p_D}, \mathbf{m_0}^{p_D})$ can be D–deadlocked, which implies the same property for any reachable marking, since $\mathrm{RS}(\mathcal{N}^{p_D}, \mathbf{m_0}^{p_D}) \subseteq \mathrm{PRS}(\mathcal{N}^{p_D}, \mathbf{m_0}^{p_D})$.

3.4 Preventing deadlock problems in S^4PR

We have concentrated on the prevention of the bad markings related to a given bad siphon. An iterative algorithm is going to be proposed to control the whole system. It is structured in the following steps:

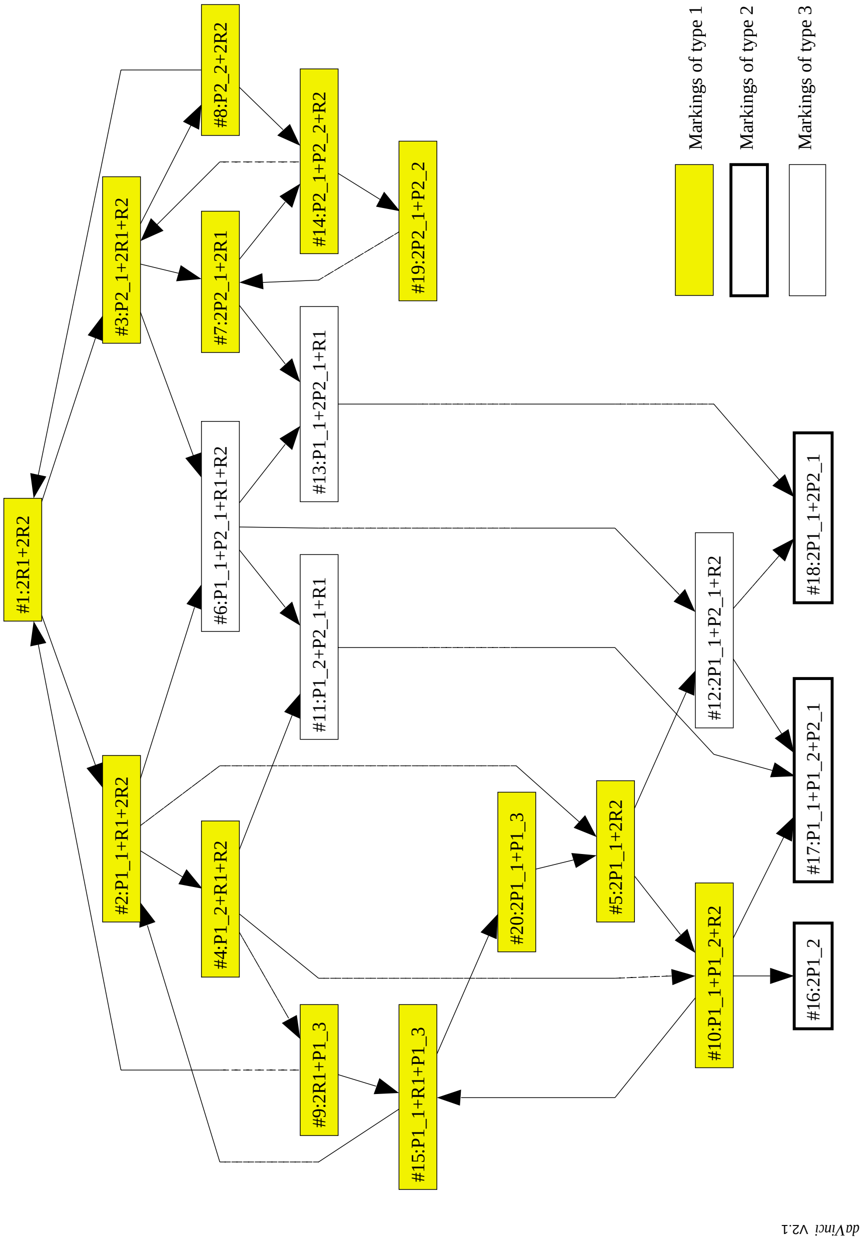


Figure 3.9: Reachability graph of the first net of Figure 3.10

Algorithm 3.1

```
 Function controlNet(In $\langle \mathcal{N}, \mathbf{m_0} \rangle$: a marked $S^4PR$) Return $\langle \mathcal{N}^{\mathtt{C}}, \mathbf{m_0}^{\mathtt{C}} \rangle$
——Pre: TRUE
——Post: $\langle \mathcal{N}^{\mathtt{C}}, \mathbf{m_0}^{\mathtt{C}} \rangle$ is a live $S^4PR$ obtained controlling $\langle \mathcal{N}, \mathbf{m_0} \rangle$
Begin
    $\langle \mathcal{N}^{\mathtt{C}}, \mathbf{m_0}^{\mathtt{C}} \rangle$ := $\langle \mathcal{N}, \mathbf{m_0} \rangle$
    Repeat
            Compute a bad siphon for $\langle \mathcal{N}^{\mathtt{C}}, \mathbf{m_0}^{\mathtt{C}} \rangle$, $\mathtt{D}$, using system (3.4)
            If $\exists$ $\mathtt{D}$, solution Then
                Compute $m_D^{max}$ as stated in Definition 39
                If $m_D^{max}$ is acceptable Then
                    Add the corresponding resource–control–place
                    as stated in Definition. 43
                Else
                    Compute $m_D^{min}$ as stated in Def. 39
                    Add the corresponding process–control–place
                    as stated in Definition. 43
                End If
            End If
    Until No new control place is added
End
```

1. Compute a bad siphon.
2. Compute m_D^{max}.
    - If the corresponding control place has an acceptable marking, go to the following step.
    - If not, compute m_D^{min}.
3. Add the control place.
4. Go to the first step, taking as input the partially controlled system, until no bad siphons exist.

Algorithm 3.1 corresponds to a more detailed implementation of these ideas. The following theorem proves the correctness of the proposed algorithm.

Theorem 47 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, *be a marked* S^4PR

- *The Algorithm 3.1 applied to* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *terminates.*
- *The resulting controlled system,* $\langle \mathcal{N}^C, \mathbf{m_0}^C \rangle$*, is live.*

Proof

- *Termination: it is a direct consequence of the following facts.*
  1. *If* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *is a marked* S^4PR*,* $|\mathrm{PRS}(\mathcal{N}, \mathbf{m_0})|$ *is finite.*
  2. *When a siphon is controlled, the resulting system is a* S^4PR*(Lemma 45).*
  3. *The addition of a control place strictly decreases the number of potentially reachable states of the controlled system (Lemma 46).*
- *When the algorithm terminates the controlled net system* $\langle \mathcal{N}^C, \mathbf{m_0}^C \rangle$ *is a* S^4PR *with an acceptable initial marking (by Lemma 45) and it has no bad siphon. Then, no marking* $\mathbf{m} \in \mathrm{PRS}(\mathcal{N}^C, \mathbf{m_0}^C)$ *can be* D*–deadlocked for any bad siphon and then, no marking* $\mathbf{m}' \in \mathrm{RS}(\mathcal{N}^C, \mathbf{m_0}^C)$ *can have a dead transition (Theorem 28).*

  Moreover, since the initial marking is acceptable, $\left|\mathrm{PRS}(\mathcal{N}^C, \mathbf{m_0}^C)\right| > 1$*.*

□

Example 3 *Let us use the* S^4PR *in Figure 3.10 to see how the Algorithm 3.1 works. In order to see the effect of the control policy, its reachability graph is depicted in Figure 3.9. The figure shows the deadlocked states (#16, #17, #18), the ones that lead in an inevitable way to them (#6, #11, #12, #13), and the ones a maximally permissive control policy should left (the rest). Notice that the original system has 20 reachable markings from which the policy should left 13. The markings forbidden by each restriction are shown in the figure by means of lines labeled with the name of the control place: the control place forbids the markings under the corresponding line.*

- **Iteration 1:** *The first bad siphon computed is* $D = \{P1\_3, P2\_1, R2\}$*. Solving the associated ILPP problem,* $m_D^{max} = 0$*, which generates the control place* $RCP1D$ *whose associated invariant is:*

$$P1\_2 + RCP1D = \mathbf{m_0}[D] - (m_D^{max} + 1) = 1$$

  It has an acceptable initial marking and it is added to the system.

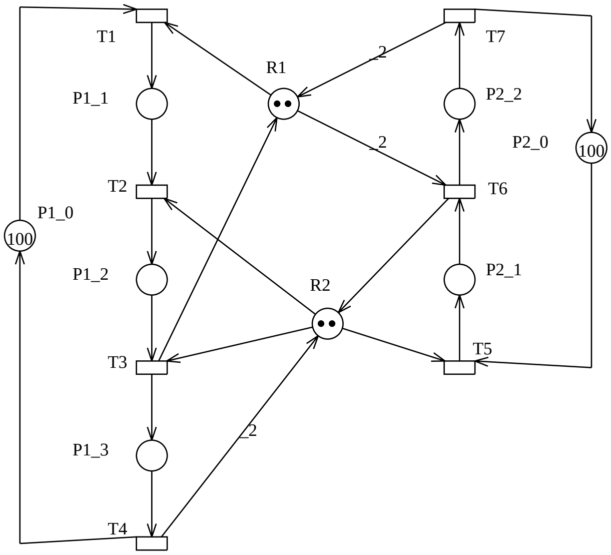


Figure 3.10: A S^4PR net.

- **Iteration 2:** *System 3.4 obtains the siphon $D = \{P1\_3,\ P2\_3,\ R1,\ R2\}$. Solving the associated ILPP problems, $m_D^{max} = 0$, which generates the control place $RCP2D$ whose associated invariant is:*

  $$P1\_1 + 2 \cdot P1\_2 + P2\_1 + RCP2D = \mathbf{m_0}[D] - (m_D^{max} + 1) = 2$$

  It has an acceptable initial marking and it is added to the system.

- **Iteration 3:** *A new siphon obtained using System 3.4 is $D = \{P1\_2,\ P2\_1,\ RCP2D\}$. Solving the associated ILPP problems, $m_D^{max} = 0$, which generates the control place $RCP3D$ whose associated invariant is:*

  $$P1\_1 + RCP3D = \mathbf{m_0}[D] - (m_D^{max} + 1) = 1$$

  It has an acceptable initial marking and it is added to the system.

- **Iteration 4:** *The next siphon obtained solving System 3.4 is $D = \{P1\_2,\ P2\_2,\ R1,\ RCP2D\}$. Solving the associated ILPP problems, $m_D^{max} = 1$, which generates the control place $RCP4D$ whose associated invariant is:*

  $$2 \cdot P1\_1 + P2\_1 + RCP4D = \mathbf{m_0}[D] - (m_D^{max} + 1) = 2$$

  It has an acceptable initial marking and it is added to the system. No new bad siphon appears and the algorithm terminates.

This system has 10 reachable states. To control the original S^4PR, four siphons have been computed and four new places have been added. In Figure 3.11 we can see the resulting S^4PR. In Figure 3.12 we can see the effect of the four control places added.

3.5 A comparison

This section introduces a set of empirical results in which a set of different control policies solve the problem of controlling a S^4PR net. The two versions of the control policy presented in this chapter (process-oriented and resource-oriented) are compared with two control policies able to deal with this general class of systems from a prevention point of view, have been implemented. These policies were introduced in [BCZ97], and in [EH93], respectively.

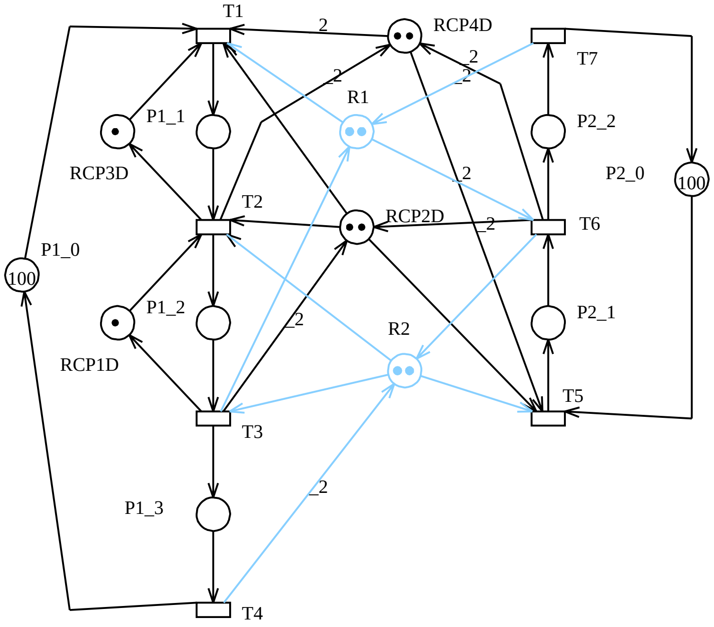


Figure 3.11: The S^4PR net obtained by controlling the system in Figure 3.10

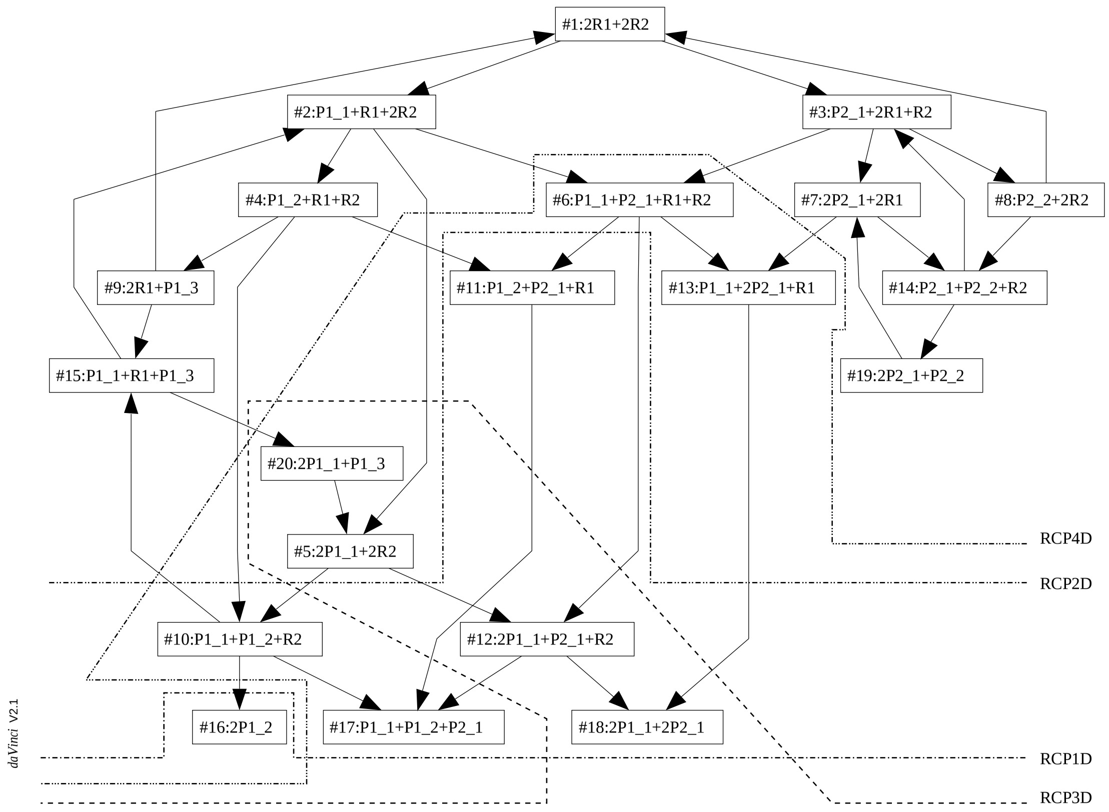


Figure 3.12: Reachability set of the net in Figure 3.10 . The states under the lines are prevented by the addition of the respective control places

			[BCZ97]	[EH93]	D–resource	D–process
	NM	NB	NMC	NMC	NMC	NMC
Net 3	471	427	94.38%	88.29%	100.00%	96.25%
Net 4	140	124	91.94%	78.23%	79.03%	79.03%
Net 5	47	42	100.00%	50.00%	100.00%	100.00%
Net 6	151	143	100.00%	47.55%	100.00%	100.00%
Net 7	1200	1149	94.34%	94.34%	94.34%	76.50%
Net 8	696	653	90.66%	86.68%	90.96%	71.82%

Table 3.3: Number of states and percentage of states left after application of control policies for the selected nets.

The first one is based on siphons, and it adds restrictions to the net such that at each reachable marking it is guaranteed that there will exist a resource of each siphon enabling all its output transitions.

The second one was originally introduced as an avoidance approach for a more restricted class of nets, but it can be implemented using the prevention point of view and can be applied to S^4PR. It is based on establishing a set of control points in the processes. When a process is going to leave one of these control points, it looks if the multi–set of resources it needs in order to reach one of its closest control points is available.

In order to carry out the comparison, the methods have been run with a set of nets. The examples presented here have been chosen in order to show a variety of results, and cannot be considered as a true statistical sample. Table 3.3 shows the results corresponding to the nets in Figures 3.10,3.13(a)–3.13(f). The columns of the table are as follows.

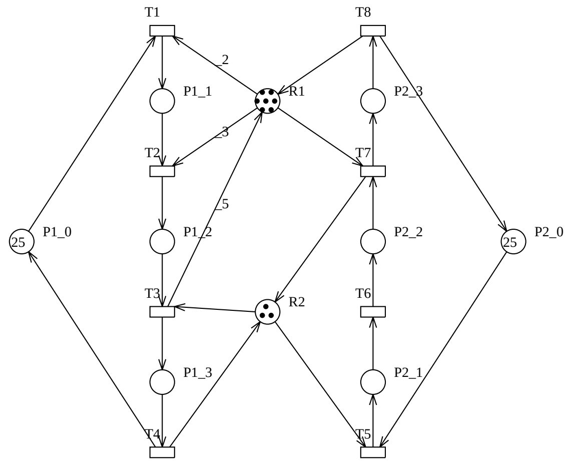
NM: is the number of states of the uncontrolled system;

NB: is the number of states that a maximally permissive control policy would allow (that is, the number of elements of the strongly connected component that contains the initial marking);

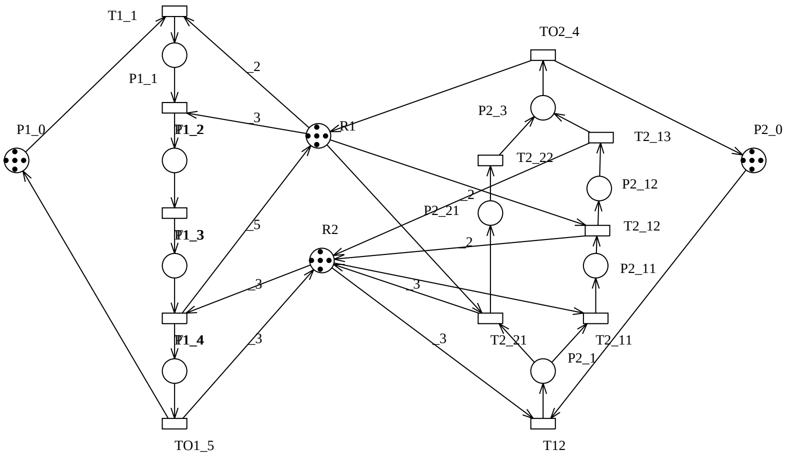
NMC: is the percentage states allowed by the control policy with respect to the number of states of the maximally permissive policy.

We would like to point out the following remarks;

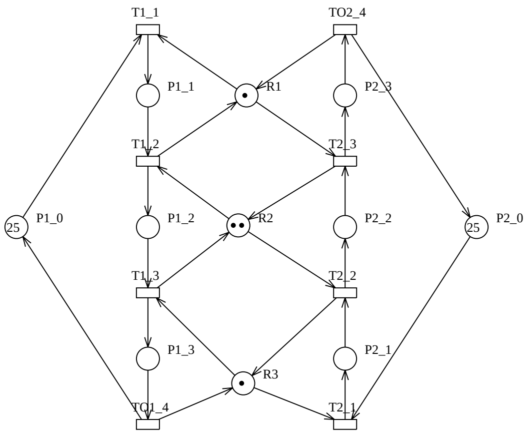
- The new control policies proposed here have a good balance between computational cost and permissiveness. In fact, their behavior is as permissive as the approach in [BCZ97], but with a clear advantage: this methods requires, at each iteration, the computation and control of each minimal siphon, which implies the necessity of computing all the minimal siphons at each iteration.


T1
T8
_2
P1_1
R1
P2_3
_3
T2
T7
_5
P1_0
25
P1_2
P2_2
25
P2_0
T3
R2
T6
P1_3
P2_1
T4
T5

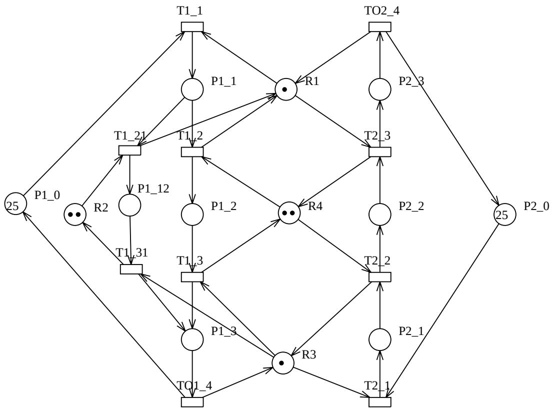
(a) Net 3


T1_1
TO2_4
_2
P1_1
_3
P2_3
P1_0
P1_2
R1
T2_13
P2_0
T2_22
P2_12
_2
P2_21
_5
R2
P1_3
T2_12
_2
P2_11
_3
_3
_3
_3
P1_4
T2_21
T2_11
P2_1
TO1_5
T12

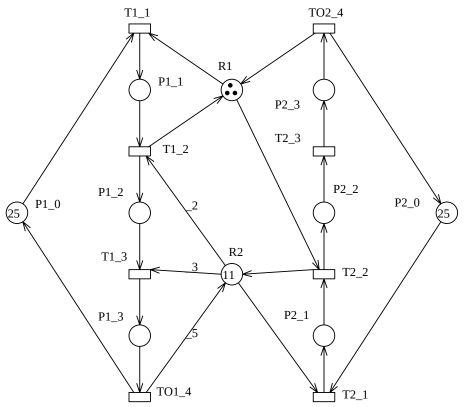
(b) Net 4


T1_1
TO2_4
P1_1
R1
P2_3
T1_2
T2_3
25
P1_0
P1_2
R2
P2_2
25
P2_0
T1_3
T2_2
P1_3
P2_1
R3
TO1_4
T2_1

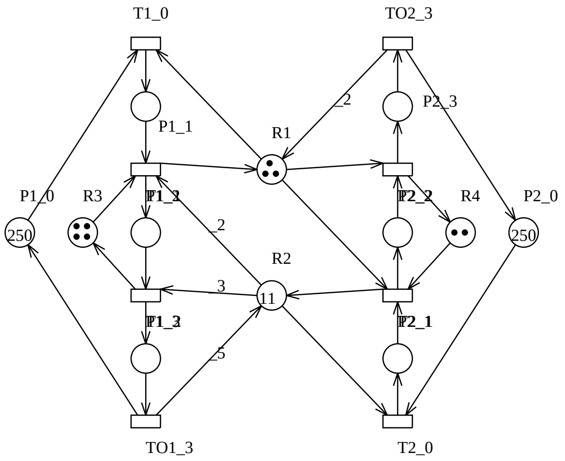
(c) Net 5


T1_1
TO2_4
P1_1
R1
P2_3
T1_21
T1_2
T2_3
P1_12
25
P1_0
R2
P1_2
R4
P2_2
25
P2_0
T1_31
T1_3
T2_2
P1_3
P2_1
R3
TO1_4
T2_1

(d) Net 6


T1_1
TO2_4
R1
P1_1
P2_3
T1_2
T2_3
P1_2
_2
P2_2
P1_0
25
P2_0
25
R2
T1_3
3
11
T2_2
P1_3
_5
P2_1
TO1_4
T2_1

(e) Net 7


T1_0
TO2_3
_2
P2_3
P1_1
R1
P1_0
R3
T1_1
P1_2
T2_2
P2_2
R4
P2_0
250
_2
250
R2
3
11
T1_2
P1_3
T2_1
P2_1
_5
TO1_3
T2_0

(f) Net 8

This can be an enormous effort. The method proposed here requires to obtain just one siphon at each iteration, which makes this approach more suitable for real applications.

- The D–resource approach is the most permissive one in most of the examples; in the other cases no other control policy has a better result than it. Our feeling is that this is due to the way the control is added: it takes into account not only the problems of the original system but also the control places added to the system, trying to add less control places.

- The D–process approach is as permissive as the D–resource one in some cases, but there are several cases where it is even worse than the previously existing approaches.

About the computational cost, we can also present some conclusions. The presented approaches are, in general, more suitable than other siphon based policies, since they only compute the needed siphons in order to reach a live system. In this sense, the D–process approach should be faster, since it never 'fails' when computing D–deadlocked markings (remember that there can exists D–resource places with non–acceptable initial markings.)

The policies presented in this chapter can be qualified as structural, that is, based on the structure of the system, in the sense that they do not need to compute the reachability set to obtain the solution. Anyway, it is obvious that the approach is not purely structural, because it depends on the initial marking (available resources).

3.6 Conclusions

In this chapter the liveness characterizations for S^4PR nets introduced in the previous chapter have been used to control the initial system, obtaining a final live controlled system. The control is based on the use of siphons of the Petri net model and it has the following characteristics:

- it adopts an structural point of view. This has the advantage of avoiding the computation of the system reachability set, avoiding the state explosion problem.

- it has been implemented in an iterative way, applying the control in an incremental way: at each iteration, one (potentially) bad siphon is computed and controlled. Therefore, the method does not need to compute at each iteration the set of minimal siphons (as is the case in [BCZ97], for instance). This is a very interesting feature of the proposed method, since it is possible for a S^4PR net to have an exponential number of (minimal) siphons, which makes very hard (if not unaffordable) the complete enumeration of all the minimal siphons.

- from the permissiveness point of view, it has been shown to have a good behavior compared to others methods able to control the class of MT-PO-RAS.

Chapter 4

Deadlock avoidance policies for S^*PR nets.

Abstract
This chapter concentrates on solving the deadlock problem for S–RAS adopting a deadlock avoidance perspective. The deadlock control policies developed are based on the Banker's algorithm, and take advantage of the process structure in order to produce more permissive control policies. Banker's-like approaches are based on a decision procedure to allow the evolution of active processes, using information about the maximal needs of resources that a process can request in order to ensure termination. Banker's algorithm considers only static information that processes must provide at the beginning of the process–execution. A framework to deal with a family of Banker's–like control policies will be presented, where the maximal need of resources is defined as a function that depends on the resources needed to ensure termination from the current state, and also on the set of sequences that must be preserved by the control policy. Several functions are presented in order to illustrate this approach. Later, an efficient way to solve the problem is presented. Finally, some examples are presented and compared looking at their "permissiveness".

4.1 Introduction

In this chapter we propose a model that covers the previously introduced subclasses of S–RAS removing all the previously enumerated constraints (except, obviously, the one related to the conservation/reusability of the resources), the S^*PR class of nets. This class extends S^4PR in the sense that no constraint is imposed to the process structure: any strongly connected state machine is allowed.

A deadlock avoidance approach based on Banker's algorithm [Dij65, Hab75] will be used. In the case of manufacturing systems, a Banker's approach has already been applied in the following works:

- In [LRF98a] a control policy is obtained for the SU–RAS class. The authors present an adapted version of the Banker's algorithm, obtaining an efficient solution. The improvement is not only based on the knowledge of the process structure, but also on the concept of "partially ordered set of active processes": it is not necessary to find an ordered termination for all the active processes, but just for some subsets of them.

- For the same class of RAS, [KTJK97] evaluates, from a performance point of view, a set of different methods for manufacturing systems with Automated Guided Vehicles (AGV), one of which is the classical Banker's approach.

- In [Rev98] and [Law00] an adaptation of the Banker's method is presented, obtaining polynomial solutions for an extension of the SU–RAS class, where each processing step can be executed in any resource from a given set.

- Finally, [Rev00] removes some of the constraints usually imposed to the process structure, developing a Banker's solution for AGV systems where controlled recirculation is allowed in the routing of guided vehicles.

Outside the scope of FMS the work in [Lan99] extended the Banker's approach to a class of systems where a multi–set of resources can be used at each processing step and flexible routing is allowed. However, in order to obtain a polynomial solution, the process structure is constrained so that the set of states has a tree structure.

In a deadlock avoidance approach, before the evolution of a process is allowed, the first step is to check if such authorization will lead to a "safe" state, i.e., a state from which all the parts being processed can terminate. In this context, the decision procedure of Banker's algorithm needs to know for each active process its maximal needs of resources along all its life. This information is static and is used together with the dynamic information about the resources assigned to each process and the set of available resources in order to determine if an ordering for the sequential termination of the active processes exists, where sequential termination means that a process is able to terminate if the rest of active processes do not move from their current states. If such an ordering can be found, the decision procedure concludes that the state is safe and the resource request can be granted.

In this chapter we define a general framework to develop Banker's-like control policies for deadlock avoidance taking advantage of the knowledge about the structure of the processes (for more restricted classes of systems, see [LRF98b, Lan99]). Two approaches are going to be considered: one that is computable in a static way, and a second one where only dynamic computations are feasible. The concept of (global) maximal needs for a whole process is transformed into the concept of maximal needs of resources related to a process state. The maximal needs of resources of a process are defined as a function depending on the needs of resources to terminate the process execution from the current state, and also on the set of execution sequences that must be preserved by the control policy. As it will be shown, in most cases this approach will lead to more permissive controlled systems.

The chapter is organized as follows. Section 4.2 introduces the class of systems and models we are considering. Section 4.3 gives an intuitive presentation of Banker's algorithm. Section 4.4 is devoted to the main results, presenting the general framework for Banker's-like algorithms for deadlock avoidance together with some particular solutions. In Section 4.5 some empirical results about the application of different Banker's–like algorithms are presented. Finally, some conclusions are presented.

4.2 The class of S^*PR nets

Let us introduce the class of S^*PR nets in a formal way. For an intuitive presentation of the main ideas, let us recall the one presented in Chapter 2, where S^4PR nets were introduced in an informal way. Here, we are going to present the class in a formal way. Then, we will show the differences with S^4PR. First of all, the structure of individual processes is defined.

Definition 48 *A* extended process Petri net *is a generalized strongly connected self–loop free Petri net* $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ *where:*

1. P *is a partition as follows:* $P = \{p_0\} \cup P_S \cup P_R$.

2. *The subnet generated by* $\{p_0\} \cup P_S \cup T$, $\mathcal{N}_{|(\{p_0\}\cup P_S,T)}$, *is a strongly connected state machine.*
3. $\forall r \in P_R$, *there exists a unique minimal P–Semiflow* $\mathbf{Y_r} \in \mathbb{N}^{|P|}$ *such that* $\{r\} = \|\mathbf{Y_r}\| \cap P_R$, $\{p_0\} \cap \|\mathbf{Y_r}\| = \emptyset$, $P_S \cap \|\mathbf{Y_r}\| \neq \emptyset$ *and* $\mathbf{Y_r}[r] = 1$.
4. $P_S = \bigcup_{r\in P_R} (\|\mathbf{Y_r}\| \setminus \{r\})$.

□

An *extended process Petri net* is a simple specification of the processing of a type of part. Notice that the only difference with a process Petri net as presented in Definition 1 (Chapter 2, page 32) is that there can exist circuits that do not contain place p_0. With this, the modeling of more complex systems providing tools to represent, for example, unlimited recirculation of parts. We will show this later but, in order to complete the modeling of the dynamics of an extended process Petri net, the introduction of an initial marking is needed. Only *acceptable initial markings*, as defined in the following, will be considered. Notice that this definition and the following results are the same as in S^4PR nets.

Definition 49 *Let* $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$ *be an extended process Petri net. An initial marking* $\mathbf{m_0}$ *is* acceptable *for* $\mathcal{N}$ *if and only if:*

1. $\mathbf{m_0}[p_0] > 0$;
2. $\forall p \in P_S \,.\, \mathbf{m_0}[p] = 0$;
3. $\forall p \in P_S \,.\, \forall r \in P_R \,.\, \mathbf{m_0}[r] \geq \mathbf{Y_r}[p]$. □

In the following, when considering a marked extended process Petri net we will assume $\mathbf{m_0}$ to be acceptable for it.

Some basic structural properties of extended process Petri nets

The structural properties of process Petri nets in Chapter 2 are also true for extended process Petri nets. In this section we will only remark the more important ones. The results shown in Proposition 4 (page 37) and Lemma 7 (page 7) presented in Chapter 2 can be trivially extended for this class of nets. They are not reproduced here.

Let us present here the definition of the S^*PR class. It is analogous to the one presented in Definition 24 (page 52), the only difference being the underlying process structure.

Definition 50 *The class of S^*PR systems is defined recursively as follows:*

1. *An extended process Petri net is a S^*PR.*

2. *The composition of two S^*PR by fusion of the common resource places is also a S^*PR.*

3. *All the S^*PR systems are generated using the previous rules.* □

The classical Banker's algorithm will be presented in terms of multi–sets. For this reason, let us comment about some well–known concepts in terms of this formalism.

Let us recall the net shown in Figure 2.9 (page 61) and the FMS shown in Figure 1.4(b)-(b) whose production routes are depicted in Figure 1.5(b) (pages 15 and 16, respectively). There we can see the main difference between S^4PR and S^*PR nets: as introduced in Section 1.3.1, the circuits that do not contain places of P_0 are related to the part recirculation capabilities. For example, in Figure 2.9 we can see that parts reaching place $P1\_1$ can be processed by resource $r3$, moving to the state represented by place $P1\_2'$. At this state, they can return to place $P1\_1$, and this sequence of steps can be repeated as many times as needed.

All the results concerning the structure of S^4PR nets presented in Chapter 2 are directly extended to this new class. However it is not the case for the liveness characterizations, as it was shown there by means of the counterexample in Figure 2.9 (page 61).

The term S^*PR does not have any specific meaning. It has been chosen since this class of nets is a generalization of previous introduced classes named as S^3PR [ECM95], $L-S^3PR$ [EGVC98b] and S^4PR [TCE99]. In fact, $L-S^3PR \subset S^3PR \subset S^4PR \subset S^*PR$. Let us also remark the fact that S^*PR is the most general class of S–RAS.

4.3 The Banker's algorithm for deadlock avoidance

A deadlock avoidance algorithm controls the system evolutions in such a way that only *safe sequences* are allowed. A sequence is *safe* if each state reached during its execution is safe. A state is considered safe if all active processes can finish.

The classical Banker's algorithm is, perhaps, the best known algorithm adopting the deadlock avoidance approach. It is based on the following idea: at the activation moment, each process must declare the maximal number of instances of each type of resource it may need during its execution (the multi–set of maximal needs).

In order to ensure that a state is safe, the Banker's algorithm uses the following sufficient condition: a state is considered to be safe if an ordering in the "sequential termination" of the active processes can be found such that the needs of each active process could be granted using the current free resources and those released by the previously terminated processes (previously terminated according to the ordering selected). Sequential termination means that it is possible to find an ordering for the set of active processes in such a way that if we freeze the system and let them evolve alone according to this order, each process can terminate with the available resources plus the ones released by the processes that have finished before.

Let us consider a state $\mathbf{m} = \{a_1, a_2, ..., a_{|\mathbf{m}|}\}$, which has to be tested for safeness and let $|\mathbf{m}|$ be the number of active processes. The method uses the following data structures:

$\boldsymbol{Available}$**:** The multi–set of available resources at the considered state.

$\mathbf{Available_m}(j) = k$ means that there are k available copies of resource type j.

$\mathbf{Max}$**:** A $|\mathbf{m}|$–indexed vector of multi–sets of resources. For each active process a_i, $\mathbf{Max}[a_i]$ is the multi–set of maximal needs of the process, declared at the process activation moment. $\mathbf{Max}[a_i](j) = k$ means that the process a_i may request at most k copies of resource j.

$\mathbf{Allocation}$**:** A $|\mathbf{m}|$–indexed vector of multi–sets of resources. For each active process, a_i, $\mathbf{Allocation_m}[i](j) = k$, is the number of copies of the resource j it is using at the state $\mathbf{m}$.

$\mathbf{Need}$**:** A $|\mathbf{m}|$–indexed vector of multi–sets of resources, representing, for each process, the resources that it would need in the future from the current state $\mathbf{m}$. For each process a_i, $\mathbf{Need_s}[a_i] = \mathbf{Max}[a_i] - \mathbf{Allocation_m}[a_i]$.

The Banker's algorithm considers that a given state is safe when there exists an ordering of the active processes allowing all of them to terminate in the following way: the first one can terminate with the resources it is holding plus the $\boldsymbol{Available}_\mathbf{m}$ ones; the second one should be able to terminate when the first one terminates, increasing $\boldsymbol{Available}_\mathbf{m}$ with the resources allocated to the first process (which are assumed to be freed once the first process terminates), and so on for the rest of active processes. Formally, it can be formulated as follows: let

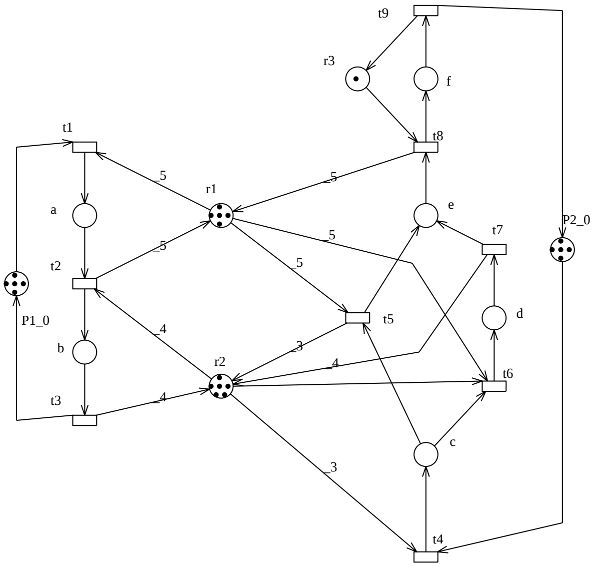


Figure 4.1: A S^*PR that will be used to show the policies

$\mathbf{m}$ be a system state with $|\mathbf{m}|$ active processes, $\{a_1 \ldots a_{|\mathbf{m}|}\}$. The Banker's algorithm considers the state $\mathbf{m}$ to be safe if and only if there exists an ordering function (a bijective mapping) $\alpha_s : \{p_1 \ldots p_{|\mathbf{m}|}\} \longrightarrow \{1 \ldots |\mathbf{m}|\}$ such that, for each $i \in \{1 \ldots |\mathbf{m}|\}$,

$$\boldsymbol{Available}_{\mathbf{m}} + \mathbf{Allocation}_{\mathbf{m}}[a_i] + \sum_{\substack{j \in \{1 \ldots |\mathbf{m}|\} \\ \alpha_{\mathbf{m}}(a_j) < \alpha_{\mathbf{m}}(a_i)}} \mathbf{Allocation}_{\mathbf{m}}[a_j] \geq \mathbf{Max}[a_i]$$

Let us now relate the Banker's algorithm and the S^*PR models. In the algorithm, each process has to be identified. As stated before, in a S^*PR model an active process is a token in a state place. If we consider, for instance, the S^*PR in Figure 4.1, marking $2 \cdot c + a + r3 + 4 \cdot P1\_0 + 3 \cdot P2\_0$ can be described as the set of processes $\{c_1, c_2, a_1\}$, where c_1, c_2 correspond to the processes modeled by the tokens in the state corresponding to place c and a_1 to the one modeled by the token in a.

To conclude this introduction, let us see these values for the net of Figure 4.1. At marking $\mathbf{m} = 2 \cdot c + a + r3 + 4 \cdot P1\_0 + 3 \cdot P2\_0$ we have:

$$\begin{aligned}
\boldsymbol{Available}_{\mathbf{m}} &= \quad r3 \\
\mathbf{Max}[c_1] &= \quad 5 \cdot r1 + 4 \cdot r2 + r3 \\
\mathbf{Max}[c_2] &= \quad 5 \cdot r1 + 4 \cdot r2 + r3 \\
\mathbf{Max}[a_1] &= \quad 5 \cdot r1 + 4 \cdot r2 \\
\mathbf{Allocation}_{\mathbf{m}}[c_1] &= \quad 3 \cdot r2 \\
\mathbf{Allocation}_{\mathbf{m}}[c_2] &= \quad 3 \cdot r2 \\
\mathbf{Allocation}_{\mathbf{m}}[a_1] &= \quad 5 \cdot r1 \\
\mathbf{Need}_{\mathbf{m}}[c_1] &= \quad 5 \cdot r1 + r2 + r3 \\
\mathbf{Need}_{\mathbf{m}}[c_2] &= \quad 5 \cdot r1 + r2 + r3 \\
\mathbf{Need}_{\mathbf{m}}[a_1] &= \quad 4 \cdot r2
\end{aligned}$$

Let us consider a system with $|\mathbf{m}|$ active processes, each one with a multi–set of r types of resources (representing the global maximal needs). As proved in [Gol78], to test if such a state is safe for the original version of the Banker's algorithm is $O(|\mathbf{m}| \cdot \log(|\mathbf{m}|) \cdot r)$.

4.3.1 A general schema for Banker's like algorithms

We are going to present a general framework to study algorithms that are similar to the classical Banker's approach. For this, let us introduce the Algorithm 4.2. It shows the general schema of the 'core' of what we will consider here Banker's based method: the algorithm to decide if a given state is considered safe or not.

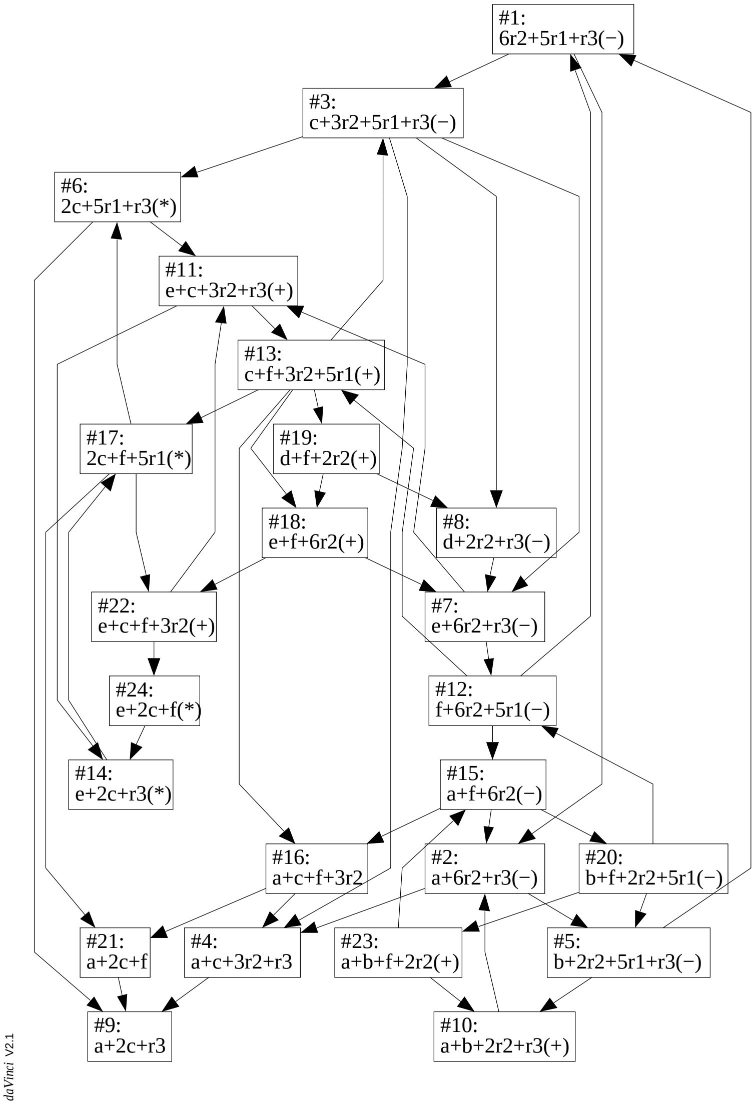


Figure 4.2: Reachability graph of the net in Fig. 4.1.

Depending on the function $isTerminable$, different policies will be obtained; in this way a family of Banker's like algorithms can be represented. In this chapter we will explore several alternative proposals for the function $isTerminable$, comparing them in terms of cost and taking advantage of the special structure of the S^*PR nets. Let us first study the complexity of the Algorithm 4.2 in terms of the complexity of the function $isTerminable$.

The number of *while* iterations is bounded by $|P_S|$ (it corresponds to the worst case: the state with at least one token in each state place). The cost of each iteration is dominated by the statement looking for a terminable process among those in $\mathcal{A}$[1]. Therefore, the cost is $O(\sum_{i=1}^{|P_S|} CLT_i)$, where CLT_i is the cost of looking for a terminable process among i different processes. Moreover, if c is a bound for the cost of checking if any process is terminable with a given set of free resources, $isSafe$ is $O(\sum_{i=1}^{|P_S|} i \cdot c) = O(c \cdot |P_S|^2)$.

In the case of S^*PR nets it is important to remark the following fact: *all the tokens (processes) in the same state place can be considered as "equivalent"*. This means that once it can be guaranteed that one of them can terminate (adopting a Banker's strategy), it is obvious that all these processes represented by tokens in the same state place can terminate one after the other, without additional checking. Notice that the presented algorithm uses this property: when it finds a $terminable$ process, it eliminates all the processes that are at the same state from the set of processes pending in the ordering process. Then, given a set of active processes, the algorithm only needs to check for the ones that are essentially "different". This means that, at a reachable marking $\mathbf{m}$, when looking for an ordering $\alpha_{\mathbf{m}}$ we will have to "order" as much items as marked state places ($\leq |P_S|$).

4.4 Several different "Banker's–like" approaches

The original Banker's algorithm was applied to a class of systems where each process had not a known structure representing the set of its possible execution paths mainly because the algorithm was conceived for operating systems where the possible execution sequences can depend on external values, and then can hardly be known a priori.

This means that, when a a process needs to move to a successor state, the controller has to take the decision of allowing or not the state change based on:

- the set of available system resources,

[1] Let us recall the definitions in Note 16 (page 46), where this notation was introduced for S^4PR. They can be extended in the obvious way to S^*PR.

Algorithm 4.2

```
 Function isSafe(In $\langle \mathcal{N}, \mathbf{m_0} \rangle$: a marked $S^4PR$;
                 In $\mathbf{m} \in \mathcal{RS}(\mathcal{N}, \mathbf{m_0}) \setminus \{\mathbf{m_0}\})$ Return (iS:boolean)
——Pre: TRUE
——Post: $iS =$ Is $\mathbf{m}$ a safe state?
Begin
    $\mathcal{A} := \mathcal{A}_{\mathbf{m}}$; $\mathbf{m}' := \mathbf{m}$; $iS :=$ TRUE
    While $iS \ \wedge \ \mathcal{A} \neq \emptyset$
        look for a process $a \in \mathcal{A}$ s.t. isTerminable($\mathcal{N}$,$\mathbf{m}'$,a)
        If such $a$ exists Then
            For Each $b \in \mathcal{A}$ s.t. $\pi_{\mathbf{m}'}(b) = \pi_{\mathbf{m}'}(a)$
                $\mathcal{A} := \mathcal{A} \setminus \{b\}$
                $\mathbf{m}' := \mathbf{m}'^{b}$
                —— $\mathbf{m}'^{b}$ represents a state where the process b has terminated
                —— and the others remain at the same state as in $\mathbf{m}'$
                —— (Notation in Definition 64)
            End For
        Else
            $iS :=$ FALSE
        End If
    End While
    Return (iS)
End
```

- the set of busy system resources when the demand arrives,
- the maximal set of resources each process declared when it was started.

The sequence of steps corresponding to the execution of an active process, from its current state until termination, is not known. In consequence, the controller must be able to ensure that the maximal claim of resources (declared before the process activation) will be granted, if needed (notice that, in fact, it is possible that some of these maximal claims were used by the process in the past, and they will not be needed anymore in the rest of the process execution).

In a S^*PR system the structure of the executions that each active process can follow is known: at a given reachable marking $\mathbf{m}$, $\pi_{\mathbf{m}}(a_i)$ represents the state place where the process a_i stays. Then, the maximal future needs of this process can be known looking at each path joining $\pi_{\mathbf{m}}(a_i)$ and the idle state in the corresponding process Petri net. In consequence, changing the notion of global maximal needs (along all the life of the process) to a notion of local maximal needs associated to each state we can obtain (depending on how the "local" term is interpreted) a family of *Banker's–like* algorithms.

4.4.1 Some improvements presented in an intuitive way

Let us present in an intuitive way the improvements considered in this work when the process structure is known.

The first one corresponds to the a natural extension of the original version using the process plan structure: *for a process, we do not need to guarantee at each step that the (global) maximal needs will be available during all of its working life, but only the maximal needs along the paths from the current state of the process to its termination state (the idle state)*. Let us explain this idea by means of an example.

Figure 4.2 depicts the reachability graph of the net in Figure 4.1. In order to show the differences between the original version and this one let us consider state $\#11$, represented by marking $\mathbf{m_{11}} = e + c + 3 \cdot r2 + r3 + 4 \cdot P1\_0 + 4 \cdot P2\_0$. In this state, there are two active processes: the tokens in places c (let us name it c_1) and e (named e_1). For these two processes the maximal needs are as follows:

$$\mathbf{Max}[c_1] = \mathbf{Max}[e_1] = 5 \cdot r1 + 4 \cdot r2 + r3$$

It is easy to see that this state is not reachable if the original Banker's approach is used: Neither

$$(3 \cdot r2 + r3) + 3 \cdot r2 \geq \mathbf{Max}[c_1]$$

nor

$$(3 \cdot r2 + r3) + 5 \cdot r1 \geq \mathbf{Max}[e_1]$$

This is so because we are assuming that both, c_1 and e_1 could need in the future $\mathbf{Max}[c_1]$, which is not true. This can be solved attaching to each state place the multi–set of resources that can be needed in the future (as the original version proposes) but taking advantage of the knowledge about the states a process can reach during its evolution from its processing until termination. Then, using this information it is clear that it is possible to terminate e_1 (since $r3$ is free) and then terminate c_1 (once e_1 terminates, $5 \cdot r1 + 3 \cdot r2 + r3$ resources are free, which is enough to ensure that c_1 could also terminate).

In the previous example, we have ensured that each process will eventually finish following no matter which path from its current state to the completion of the processing. But it is enough to find an ordering in such a way that a given process can finish, which is equivalent to ensure that resources can be granted in such a way that *at least* one path from the actual state to the idle one can be followed: if the maximal needs differ depending on the processing path (completion of a processing path), we do not need to guarantee the maximal needs but only the adequate needs to finish following one of the processing paths. Let us concentrate on the state $\#14$, represented by the marking $\mathbf{m_{14}} = e + 2 \cdot c + r3 + 5 \cdot P1\_0 + 2 \cdot P2\_0$. For each process represented by tokens in c there are two different maximal needs, depending on which path is chosen for termination. For the path corresponding to transitions $\{t4, t5, t8, t9\}$, the multi–set of maximal needs is

$$\mathbf{n1} = 5 \cdot r1 + 3 \cdot r2 + r3$$

while for the path corresponding to transitions $\{t4, t6, t7, t8, t9\}$ is

$$\mathbf{n2} = 5 \cdot r1 + 4 \cdot r2 + r3$$

For the process corresponding to the token in e there is a unique path, which requires $5 \cdot r1 + r3$. This information is sketched in the following table, where column $\mathbf{Max1}$ represents the maximal needs of the previous improvement and $\mathbf{Max2}$ the needs of the current one:

	Max1[*]	**Max2**[*]	$\mathbf{Y_R}[*]$	$\mathbf{m}_{P_R}$
c_1	$5 \cdot r1 + 4 \cdot r2 + r3$	**n1** or **n2**	$3 \cdot r2$	$r3$
c_2	$5 \cdot r1 + 4 \cdot r2 + r3$	**n1** or **n2**	$3 \cdot r2$	
e_1	$5 \cdot r1 + r3$	$5 \cdot r1 + r3$	$5 \cdot r1$	

It is easy to prove that marking $\mathbf{m_{14}}$ is not safe neither for the first improvement nor for the original algorithm. Now, let us consider that for each process represented by tokens in c either $\mathbf{n1}$ or $\mathbf{n2}$ can be chosen as the maximal needs vector.

1. If e_1 is considered, then

   $$(r3) + (5 \cdot r1) \geq 5 \cdot r1 + r3$$

   So, e_1 can finish.

2. Then, let us choose $\mathbf{n1}$ for c_1; and then:

   $$(r3) + (3 \cdot r2) + (5 \cdot r1) \geq 5 \cdot r1 + 3 \cdot r2 + r3$$

3. Finally, either $\mathbf{n1}$ or $\mathbf{n2}$ can be chosen for $c2$, since

   $$(5 \cdot r1) + (r3) + (3 \cdot r2) + (3 \cdot r2) \geq 5 \cdot r1 + 3 \cdot r2 + r3$$

So, following the proposed order and paths, $\mathbf{m}$ can be considered safe, according to the new Banker's–like approach.

The exploration of all the available paths can make this policy expensive from the computation point of view; on the other hand, the requirement of the availability of the maximal needs of resources along all the available paths proposed in the first policy seems to be excessive. In consequence, an intermediate approach will be proposed based on these two approaches: the idea is to select a single path of the set of available ones and to use it to check whether the part can finish or not. This will be examined with a proposal based on the shortest path, but more heuristics can be defined, changing the way to choose the path.

The previous ideas to improve the classical method are based on static information, computed off–line that attaches to each state place information about multi–sets of resources needed to follow a set of paths until the process termination. The on–line computations compare the set of available resources and these multi–sets. Alternatively, a (more) dynamic solution can be adopted. Let us consider a given state $\mathbf{m}$ and let us concentrate on an active process, say a, such that $\pi_{\mathbf{m}}(a) = p \in P_{S_i}$. We need to know if a can terminate, assuming that the rest of active processes do not evolve. To do that, process a can use the set of resources it is already holding, $\mathbf{Y_r}[p]$, plus those which are free, $\mathbf{m}_R$.

Therefore, in order to determine that process a can terminate, it is enough to check if there exists a path joining p and p_{0_i} involving state places of P_{S_i} and requiring no more than $\mathbf{Y_r}[p] + \mathbf{m}_R$ resources.

For example, let us recall $\#11$, represented by the marking $\mathbf{m_{11}} = e + c + 3 \cdot r2 + r3 + 5 \cdot P1\_0 + 3 \cdot P2\_0$. For the process e_1 it is easy to see that its only successor place, f, could hold a process in it with the available resources, and that is the only place in a path from e to $P2\_0$, so it can terminate. After this, there will be five available units of resource $r1$. With this, it is easy to see that the system has available resources to have a process in places d, e, and f. Then, it is straightforward to see that a path as needed exists and that processes in c (c_1 and c_2) can terminate. In consequence, the state is safe.

4.4.2 A static approach

This section presents a general formal framework for the generation of Banker's–like deadlock avoidance policies. The framework is based on the definition and parametrisation of a set of functions, the *bound functions of future needs*, that can be used to guarantee the termination of one active process. One of these functions applies each state of each type of part (a token in the Petri net) to a set of multi–sets of resources. Each one of these multi–sets must represent the future needs of the part following an available path. Different instances of these functions will generate different Banker's–like algorithms. Their values can be computed statically for each state place, and stored to be used during the normal system operation.

Let us first introduce some definitions that will be needed later

Definition 51 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle = \bigcirc_{i \in I_\mathcal{N}} \mathcal{N}_i$ be a S^*PR. Let us assume that $p \in P_{S_i}$.*

- *A* p–processing path *is a path $pt_0p_1t_1 \ldots p_{k-1}t_kp_{0_i}$ where $\{p_{0_i}\} = P_{0_i}$, $\{p, p_1 \ldots p_{k-1}\} \subseteq P_{S_i} \setminus P_{0_i}$.*
- *A* p–processing path *is a* simple p–processing path *when the path is simple.*
- *$\mathcal{T}(p)$ denotes the set of all p–processing paths,*
- *$\mathcal{T}_S(p)$ denotes the set of all simple p–processing paths.*

□

Now we will use these definitions to introduce some interesting sub–paths.

Definition 52 *Let $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$ be a S^*PR, and let us consider a p–processing path $F = (pt_0p_1t_1 \ldots p_{k-1}t_kp_0)$.*

	P–Semiflow
$\mathbf{Y_{R1}}$	$P2\_21 + 2 \cdot P1\_1 + 5 \cdot P1\_2 + 5 \cdot P1\_3 + P2\_3 + P2\_12 + R1$
$\mathbf{Y_{R2}}$	$3 \cdot P1\_4 + 4 \cdot P2\_11 + 2 \cdot P2\_12 + 3 \cdot P2\_1 + R2$

Table 4.1: Resource related minimal P–Semiflows of the net in Figure 3.5

- $SUC(p, F) = \{p, p_1, \dots, p_{k-1}\}$,
- $SUC(p) = \bigcup_{F \in \mathcal{T}(p)} SUC(p, F)$,
- $SUC_S(p) = \bigcup_{F \in \mathcal{T}_S(p)} SUC(p, F)$ □

Finally, some parameters related to the use of resources by a part considering its current state are presented.

Definition 53 *Let* $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$ *be a* S^*PR*, let* $p \in P_S$*, let* $r \in P_R$*, and let* $F \in \mathcal{T}(p)$*.*

- *The multi–set of* potential needs of resources of state p along F *is:*

$$\forall r \in P_R \,.\, \mathbf{PNR_F}(p)[r] = max_{q \in SUC(p,F)}\{\mathbf{Y_R}[q]\}$$

- *The multi–set of* potential needs of resources of state p *is:*

$$\forall r \in P_R \,.\, \mathbf{PNR}(p)[r] = max_{F \in \mathcal{T}(p)}\{\mathbf{PNR_F}(p)[r]\}$$

□

Table 4.2 shows these multi–sets for the elements in the S^*PR represented in Figure 3.5 (page 74) obtained from the minimal P–Semiflows related to resources (shown in Table 4.1.)

Definition 54 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$*,* $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$*, be a marked* S^*PR*. Let* $p \in P_S$*, let* $F_p = (q_0 t_1 q_1 \dots q_{k-1} t_k q_k) \in \mathcal{T}(p)$ $(q_0 = p, q_k \in P_0)$*, let* $f : P_0 \cup P_S \longrightarrow 2^{\mathrm{Bag}(P_R)}$ *and let* $\boldsymbol{b} \in \mathrm{Bag}(P_R)$*. The multi–set* $\boldsymbol{b}$ bounds *the path* F_p *if, and only if,* $\forall i \in \{0 \dots k\} \,.\, \exists \boldsymbol{b}_i \in f(q_i) \,.$ *such that* $\boldsymbol{b} \geq \boldsymbol{b_0} \geq \boldsymbol{b}_1 \geq \dots \geq \boldsymbol{b}_k$*.* □

That is, $\boldsymbol{b}$ bounds the path if it is possible to find a non–decreasing sequence of b–bounded multi–sets along the path.

Place	$\mathbf{PNR_F}()$			$\mathbf{PNR}()$
	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	
P1_1	$5 \cdot R1 + 3 \cdot R2$	0	0	$5 \cdot R1 + 3 \cdot R2$
P1_2	$5 \cdot R1 + 3 \cdot R2$	0	0	$5 \cdot R1 + 3 \cdot R2$
P1_3	$5 \cdot R1 + 3 \cdot R2$	0	0	$5 \cdot R1 + 3 \cdot R2$
P1_4	$3 \cdot R2$	0	0	$3 \cdot R2$
P2_1	0	$R1 + 4 \cdot R2$	$R1 + 3 \cdot R2$	$R1 + 4 \cdot R2$
P2_11	0	$R1 + 4 \cdot R2$	0	$R1 + 4 \cdot R2$
P2_12	0	$R1 + 2 \cdot R2$	0	$R1 + 2 \cdot R2$
P2_21	0	$R1$	0	$R1$
P2_3	0	$R1$	$R1$	$R1$

Table 4.2: PNR values the example.

Definition 55 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, *be a marked* S^*PR. *Let* $f : P_0 \cup P_S \longrightarrow 2^{\mathrm{Bag}(P_R)}$ *be a mapping.* f *is said to be a* bound function of future needs of resources *(*$bffnr$*) for* $\mathcal{N}$ *if, and only if,*

1. $\forall p \in P_0 \,.\, f(p) = \mathbf{0}$

2. $\forall p \in P_S \,.\, \forall \boldsymbol{b} \in f(p) \,.\, \mathbf{m_0}_{P_R} \geq \boldsymbol{b} \geq \mathbf{Y_R}[p]$

3. $\forall p \in P_S \,.\, \forall \boldsymbol{b} \in f(p)$ *there exists at least a path* $F_{p,\boldsymbol{b}} \in \mathcal{T}(p)$ *such that* $\boldsymbol{b}$ *bounds* $F_{p,\boldsymbol{b}}$. □

That is,

- f is such that it associates the multi–set **0** to places in P_0.

- For each state place p and each $\boldsymbol{b} \in f(p)$:

  1. $\boldsymbol{b}$ is bounded by the multi–set of resources available at the initial marking,

  2. $\boldsymbol{b}$ bounds the multi–set of resources used at this place.

- For each state place p, and for each $\boldsymbol{b} \in f(p)$, there exists a path bounded by $\boldsymbol{b}$.

In consequence, a $bffnr$ is a set of upper bounds of the real needs of resources from a given state place that bound at least one path until termination. As it will be shown, different instances of $bffnr$ will give different Banker's versions.

Note 56 *In the following, for a given $p \in P_S$ and $\boldsymbol{b} \in f(p)$, $\mathcal{T}_f(p, \boldsymbol{b})$ will denote the set of all the paths verifying last point of Definition 55, that is, the set of all the paths bounded by $\boldsymbol{b}$.* □

Definition 57 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^*PR. Let f be a $bffnr$ for it, let $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ and let $\mathcal{A}_\mathbf{m} = \{a_1 \dots a_{|\mathbf{m}|}\}$ be the set of active processes at $\mathbf{m}$. $\mathbf{m}$ is said to be a* f-safe *state if, and only if:*

- $\mathbf{m} = \mathbf{m_0}$, *or*
- $\mathbf{m} \neq \mathbf{m_0}$, *and:*
  1. *there exists an ordering $\alpha_\mathbf{m} : \mathcal{A}_\mathbf{m} \longrightarrow \{1 \dots |\mathbf{m}|\}$*
  2. *there exists $B_\mathbf{m} = (\boldsymbol{b}_1 \dots \boldsymbol{b}_{|\mathbf{m}|}) \in f(\pi_\mathbf{m}(a_1)) \times \cdots \times f(\pi_\mathbf{m}(a_{|\mathbf{m}|}))$ such that, for each $i \in \{1 \dots |\mathbf{m}|\}$,*
     $\mathbf{m}_{P_R} + \mathbf{Y_R}[\pi_\mathbf{m}(a_i)] + \sum_{\substack{j \in \{1 \dots |\mathbf{m}|\} \\ \alpha_\mathbf{m}(a_j) < \alpha_\mathbf{m}(a_i)}} \mathbf{Y_R}[\pi_\mathbf{m}(a_j)] \geq \boldsymbol{b}_i$ □

So, a state will be considered *f–safe* if, and only if:

- there are no active processes, or
- there exists an ordering for the active processes and there exist bounds associated to them in such a way that, for each active process, the bound is less or equal than the multi–set of resources available at marking $\mathbf{m}$, plus the multi–sets of resources held by this process and all the other active processes previous to it in the ordering.

Note 58 *In the sequel, given a S^*PR, $\langle \mathcal{N}, \mathbf{m_0} \rangle$, and a $bffnr$for it, f,*

- *The set $\mathrm{RS}_f(\mathcal{N}, \mathbf{m_0}) = \{\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) | \mathbf{m}$ is f–safe$\}$, will denote the set of reachable states that are f–safe (these states will be also called* f–reachable states*).*
- *A S^*PR system, controlled in such a way that only* f–reachable states *are allowed, is called the* f–controlled *system.*

□

Obviously, a way of avoiding deadlocks could consist in forbidding any process activation. But this has no sense from the point of view of a physical system. Next lemma proves that the $bffnr$ś based approach is less constraining.

Lemma 59 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^*PR. Let f be a $bffnr$ for it. Then:*

1. $\mathbf{m_0} \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m_0})$.
2. $|\mathrm{RS}_f(\mathcal{N}, \mathbf{m_0})| \geq 2$.

Proof

1. *From Definition 57, point 1 trivially holds.*
2. *Let $t \in {P_0}^\bullet \cap T$, and let us assume that $\mathbf{m_0} \xrightarrow{t} \mathbf{m}$ (notice that, because of the structure of a S^*PR, $t \notin {}^\bullet P_0$). Let $\{p\} = t^\bullet \cap P_S$. Let $\boldsymbol{b}_1 \in f(p)$. Condition 2 of Definition 57 becomes $\mathbf{m}_{P_R} + \mathbf{Y_R}[p] \geq \boldsymbol{b}_1$, which is equivalent to $\mathbf{m_0}_{P_R} - \mathbf{Y_R}[p] + \mathbf{Y_R}[p] \geq \boldsymbol{b}_1$ which holds by condition 2 in Definition 55.*

□

The following lemma proves that when a system is controlled using a $bffnr$ function, there is always an active process able to evolve in one of its production sequences.

Lemma 60 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, be a marked S^*PR. Let f be a $bffnr$ for it, and let $\mathbf{m} \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m_0}) \setminus \{\mathbf{m_0}\}$. Then, there exists at least one active process a_1 such that $\pi_{\mathbf{m}}(a_1) = p \in P_S$ and a transition $t_1 \in p^\bullet$ such that*

1. $\mathbf{m} \xrightarrow{t_1} \mathbf{m}_1$.
2. $\mathbf{m}_1 \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m})$.

Proof
Let $\mathbf{m} \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m_0})$, $\mathbf{m} \neq \mathbf{m_0}$. First, we are going to prove that $\mathbf{m}$ enables at least a transition, t_1. Then, we will prove that in case of firing t_1, the reached marking belongs to $\mathrm{RS}_f(\mathcal{N}, \mathbf{m_0})$.

Since $\mathbf{m} \neq \mathbf{m_0}$, let $\mathcal{A}_{\mathbf{m}} = \{a_1 \ldots a_{|\mathbf{m}|}\} (\neq \emptyset)$, and let us also consider $\alpha_{\mathbf{m}}$, $\pi_{\mathbf{m}}$ and $B_{\mathbf{m}} = (\boldsymbol{b_1} \ldots \boldsymbol{b}_{|\mathbf{m}|})$ as in Definition 57. Without lost of generality, let us assume that $\alpha_{\mathbf{m}[a_1]} = 1$ and let $\pi_{\mathbf{m}}(a_1) = p$.

Let us first prove that $\mathbf{m}$ enables t_1. Since $\mathbf{m}[p] > 0$, t_1 is process–enabled. If $t_1 \in {}^\bullet P_0$, since in S^*PR nets ${}^{\bullet\bullet}P_0 \cap P_R = \emptyset$, t_1 is also resource–enabled. Let us now assume that $t_1 \notin {}^\bullet P_0$. Since $\mathbf{m} \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m_0})$, inequality in Definition 57 for a_1 is $\mathbf{m}_{P_R} + \mathbf{Y_R}[p] \geq \boldsymbol{b_1}$. According to Definition 55, let us consider the path $F =$

$(pt_1q_1 \dots q_{k-1}t_kp_0) \in \mathcal{T}_f(p, \boldsymbol{b_1})$. *Since* $q_1 \in F$*, there exists* $\boldsymbol{b}_1 \in f(q_1)$ *such that* $\boldsymbol{b}_1 \geq \mathbf{m}_1 \geq \mathbf{Y_R}[q_1]$*, and then,* $\mathbf{m}_{P_R} \geq \mathbf{Y_R}[q_1] - \mathbf{Y_R}[p]$. *So, for each* $r \in P_R$, $\mathbf{m}_{P_R}(r) \geq \mathbf{Y_R}[q_1][r] - \mathbf{Y_R}[p][r] = \mathbf{Pre}[r, t_1]$. *Then,* t_1 *is also resource–enabled, and point 1) can be concluded.*

Let us now prove that $\mathbf{m}_1 \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m})$.

- *if* $t_1 \in {}^\bullet P_0$, $\mathcal{A}_{\mathbf{m}_1} = \mathcal{A}_\mathbf{m} \setminus \{a_1\} = \{a_2 \dots a_{|\mathbf{m}|}\}$*; let us consider the mapping* $\alpha_{\mathbf{m}_1} : \mathcal{A}_{\mathbf{m}_1} \longrightarrow \{1 \dots |\mathbf{m}| - 1\}$ *defined as* $\alpha_{\mathbf{m_1}}(a_i) = \alpha_\mathbf{m}(a_i) - 1$ *(where* $\pi_{\mathbf{m_1}}$ *is* $\pi_\mathbf{m}$ *constrained to* $\mathcal{A}_{\mathbf{m}_1}$*), and* $B_{\mathbf{m}_1} = (\boldsymbol{b_2} \dots \boldsymbol{b}_{|\mathbf{m}|})$.

  Then, inequality in Definition 57 is now

  $$\begin{aligned} \forall i \in \{2 \dots |\mathbf{m}|\}\,. \qquad & \mathbf{m}_{1_{P_R}} \\ + \quad & \mathbf{Y_R}[\pi_\mathbf{m}(a_i)] \\ + \quad & \sum_{\substack{j \in \{2 \dots |\mathbf{m}|\} \\ \alpha_\mathbf{m}(a_j) < \alpha_\mathbf{m}(a_i)}} \mathbf{Y_R}[\pi_\mathbf{m}(a_j)] \quad \geq \boldsymbol{b_i} \end{aligned}$$

  Taking into account that $\mathbf{m}_{1_{P_R}} = \mathbf{m}_{P_R} + \mathbf{Y_R}[p]$ *and that* $\mathbf{Y_R}[p] \geq 0$*, the previous set of inequalities is just a subset of the set of inequalities corresponding to* $\mathbf{m} \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m_0})$*, which are then verified.*

- *if* $t_1 \notin {}^\bullet P_0$*, then* $\mathcal{A}_\mathbf{m} = \mathcal{A}_{\mathbf{m}_1}$*; let us consider the mapping* $\alpha_{\mathbf{m_1}} = \alpha_\mathbf{m}$ *defined as* $\forall i \in \{2 \dots |\mathbf{m}|\}\,.\,\pi_{\mathbf{m}_1}(a_i) = \pi_\mathbf{m}(a_i)$*, while* $\pi_{\mathbf{m}_1}(a_1) = q_1$*, and* $B_{\mathbf{m}_1} = (\mathbf{m}_1\boldsymbol{b}_2 \dots \boldsymbol{b}_{|\mathbf{m}|})$*, where* $\mathbf{m}_1$ *is such that* $\boldsymbol{b}_1 \geq \mathbf{m}_1$.

  - *For* $i = 1$*, inequality in Definition 57 is* $\mathbf{m}_{1_{P_R}} + \mathbf{Y_R}[q_1] \geq \mathbf{m}_1$*; this is equivalent to* $\mathbf{m}_{P_R} - \mathbf{Y_R}[q_1] + \mathbf{Y_R}[p] + \mathbf{Y_R}[q_1] \geq \mathbf{m}_1$*, which is true since* $\mathbf{m}_{P_R} - \mathbf{Y_R}[p] \geq \boldsymbol{b}_1 \geq \mathbf{m}_1$ *and* $\mathbf{Y_R}[p] \geq 0, \mathbf{Y_R}[q_1] \geq 0$.
  - $$\begin{aligned} \forall i \in \{2 \dots |\mathbf{m}|\}\,. \qquad & \mathbf{m}_{1_{P_R}} \\ + \quad & \mathbf{Y_R}[\pi_{\mathbf{m}_1}(a_i)] \\ + \quad & \sum_{\substack{j \in \{1 \dots |\mathbf{m}|\} \\ \alpha_{\mathbf{m}_1}(a_j) < \alpha_{\mathbf{m}_1}(a_i)}} \mathbf{Y_R}[\pi_{\mathbf{m}_1}(a_j)] \quad \geq \boldsymbol{b_i} \end{aligned}$$

    which is equivalent to

    $$\begin{aligned} \forall i \in \{2 \dots |\mathbf{m}|\}\,. \qquad & (\mathbf{m}_{P_R} - \mathbf{Y_R}[q_1] + +\mathbf{Y_R}[p]) \\ + \quad & \mathbf{Y_R}[\pi_\mathbf{m}(a_i)] + \mathbf{Y_R}[q_1] \\ + \quad & \sum_{\substack{j \in \{2 \dots |\mathbf{m}|\} \\ \alpha_\mathbf{m}(a_j) < \alpha_\mathbf{m}(a_i)}} \mathbf{Y_R}[\pi_\mathbf{m}(a_j)] \quad \geq \boldsymbol{b}_i \end{aligned}$$

    or, in other way,

    $$\begin{aligned} \forall i \in \{2 \dots |\mathbf{m}|\}\,. \qquad & (\mathbf{m}_{P_R} + \mathbf{Y_R}[\pi_\mathbf{m}(a_i)] \\ + \quad & \sum_{\substack{j \in \{1 \dots |\mathbf{m}|\} \\ \alpha_\mathbf{m}(a_j) < \alpha_\mathbf{m}(a_i)}} \mathbf{Y_R}[\pi_\mathbf{m}(a_j)] \quad \geq \boldsymbol{b}_i \end{aligned}$$

    which is true since $\mathbf{m} \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m_0})$.

□

As a consequence we can show that the initial marking is always reachable from any f–reachable state. The algorithm simply checks the conditions of Definition 57.

Theorem 61 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, *be a marked* S^*PR. *Let* f *be a* bffnr *for it, and let* $\mathbf{m} \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m_0}) \setminus \{\mathbf{m_0}\}$. *Then* $\mathbf{m_0} \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m})$.

Proof
Let $\mathcal{A}_\mathbf{m} = \{a_1 \dots a_{|\mathbf{m}|}\}(\neq \emptyset)$; *for each* $i \in \{1 \dots |\mathbf{m}|\}$, *let us consider a path* $F_i \in \mathcal{T}_f(\pi(a_i))$, $F_i = \pi(a_i) t_i^1 p_i^1 \dots t_i^{k_i} p_{i_0}$.

Let us proceed by induction over $NT_\mathbf{m} = \sum_{i \in \{1 \dots |\mathbf{m}|\}} \left| \{t_i^1, t_i^2 \dots t_i^{k_i}\} \right|$. *Notice that since* $\mathbf{m} \neq \mathbf{m_0}$, $NT_\mathbf{m} \geq 1$.

If $NT_\mathbf{m} = 1$, *Lemma 60 ensures that* $\mathbf{m} \xrightarrow{t_1^1} \mathbf{m_0}$, *and we can conclude. If* $NT_\mathbf{m} > 1$, *let us assume that* $a_i = \alpha^{-1}(1)$. *Lemma 60 ensures that* $\mathbf{m} \xrightarrow{t_i^1} \mathbf{m}_1$, $\mathbf{m}_1 \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m_0})$. *Considering for* $\mathbf{m}_1$ *the same paths considered for* $\mathbf{m}$, $NT_{\mathbf{m}_1} = NT_\mathbf{m} - 1$ *from which, by induction hypothesis, we can conclude.* □

An immediate corollary is that *in a f–controlled system each active process can terminate and, in consequence, no deadlock problem can exist*. Let us present the last technical result, which establishes a kind of monotonic relation among bffnr's and the set of reachable states of the controlled systems.

Proposition 62 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$, *be a marked* S^*PR. *Let* f_1, f_2 *be two* bffnr *for it such that for each* $p \in P_S$, *for each* $\boldsymbol{b_1} \in f_1(p)$, *there exists* $\boldsymbol{b_2} \in f_2(p)$, $\boldsymbol{b_1} \geq \boldsymbol{b_2}$. *Then,* $\mathrm{RS}_{f_1}(\mathcal{N}, \mathbf{m_0}) \subseteq \mathrm{RS}_{f_2}(\mathcal{N}, \mathbf{m_0})$.

Proof
Let $\mathbf{m} \in \mathrm{RS}_{f_1}(\mathcal{N}, \mathbf{m_0})$; *let us consider* $\mathcal{A}_\mathbf{m} = \{a_1 \dots a_{|\mathbf{m}|}\}$, $\alpha_\mathbf{m}$, $\pi_\mathbf{m}$ *and* $B_\mathbf{m} = (\boldsymbol{b_1} \dots \boldsymbol{b_{|\mathbf{m}|}})$. *Inequality in Definition 57 is*

$$\forall i \in \{1 \dots |\mathbf{m}|\} \,.\, \mathbf{m}_{P_R} + \mathbf{Y_R}[\pi_\mathbf{m}(a_i)] + \sum_{\substack{j \in \{1 \dots |\mathbf{m}|\} \\ \alpha_\mathbf{m}(a_j) < \alpha_\mathbf{m}(a_i)}} \mathbf{Y_R}[\pi_\mathbf{m}(a_j)] \;\geq \boldsymbol{b}_i$$

According to the hypothesis, let us choose, for each $i \in \{1 \dots |\mathbf{m}|\}$, *a* $\boldsymbol{b}'_i \in f_2(\pi_\mathbf{m}(a_i))$ *such that* $\boldsymbol{b}_i \geq \boldsymbol{b}'_i$; *then, using* $\alpha_\mathbf{m}, \pi_\mathbf{m}, B'_\mathbf{m} = (\boldsymbol{b}'_1 \dots \boldsymbol{b}'_{|\mathbf{m}|})$, *we have that*

$$\forall i \in \{1 \dots |\mathbf{m}|\} \,.\, \mathbf{m}_{P_R} + \mathbf{Y_R}[\pi_\mathbf{m}(a_i)] + \sum_{\substack{j \in \{1 \dots |\mathbf{m}|\} \\ \alpha_\mathbf{m}(a_j) < \alpha_\mathbf{m}(a_i)}} \mathbf{Y_R}[\pi_\mathbf{m}(a_j)] \;\geq \boldsymbol{b}_i \geq \boldsymbol{b}'_i$$

and then, $\mathbf{m} \in \mathrm{RS}_{f_2}(\mathcal{N}, \mathbf{m_0})$. □

Note 63 *In the sequel,* $f_1 \geq f_2$ *will denote that two* bffnr*'s,* f_1 *and* f_2*, verify conditions of Proposition 62.* □

Algorithm 4.3 shows an adaptation of Algorithm 4.2 to the framework of the bffnr formalism.

Algorithm 4.3

```
 Function isFSafe(In $\langle \mathcal{N}, \mathbf{m_0} \rangle$: a marked $S^4PR$; In $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) \setminus \{\mathbf{m_0}\}$;
                         In f: a $\mathit{bffnr}$ for $\mathcal{N}$) Return (iFS:boolean)
——Pre: $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$
——Post: $iFS = (\mathbf{m} \in \mathrm{RS}_f(\mathcal{N}, \mathbf{m_0}))$
Begin
    $\mathcal{A}_\mathbf{m} := \{a_1 \ldots a_{|\mathbf{m}|}\}$
    $iFS :=$ FALSE
    options := $\prod_{i \in \{1 \ldots |\mathbf{m}|\}} f(\pi_\mathbf{m}(a_i))$
    While $\neg iFS \ \wedge \ \mathit{options} \neq \emptyset$
        choose $b = (b_1 \ldots b_{|\mathbf{m}|}) \in \mathit{options}$
        If there exists an ordering $\alpha_\mathbf{m}$ as in Definition 57
            Then $iFS :=$ TRUE
            Else $\mathit{options} := \mathit{options} \setminus \{b\}$
        End If
    End While
    Return (iFS)
End
```

Computational cost

Testing the if–guard in Algorithm 4.3 is like applying a classical Banker's algorithm being $(\boldsymbol{b}_1, ..., \boldsymbol{b}_{|\mathbf{m}|})$ the maximal needs of resources for such processes. According to [Gol78], this is $O(|P_S| \cdot \log(|P_S|) \cdot |P_R|)$. Moreover, the loop will be executed, at most $\left|\prod_{p \in P_S} f(p)\right|$ times. So, the worst–case time complexity of the previous algorithm is $O(|P_S| \cdot \log(|P_S|) \cdot |P_R| \cdot \prod_{p \in P_S} |f(p)|)$.

Once this general framework for Banker's–like solutions has been introduced, let us now concentrate on providing some specific instances of bffnr's For each case, two time costs are estimated.

1. The cost of testing if a given state is safe with respect to the considered bffnr function.

2. The cost of computing the considered bffnr itself.

This last computation has to be carried outline just once for each state place, and the result has to be stored in such a way that there is a set of multi–set of resources associated to each state place. The critical cost is the first one, since it corresponds to the on–line computing needed to decide if the firing of an enabled transition should be accepted or not.

The classical Banker's approach

Let us consider the *bffnr* f_c defined as

$$\forall i \in I_{\mathcal{N}} \, . \begin{cases} f_c(p)(r) = max_{q \in P_{S_i}} \{\mathbf{Y_R}[q]\} & \forall p \in P_{S_i}, \forall r \in P_R \\ f_c(p) = \mathbf{0}, & \forall p \in P_0 \end{cases}$$

That is, each place has associated the multi–set of maximal needs of each type of resource, along all the processing path. Notice that f_c trivially verifies conditions to be a *bffnr* for $\mathcal{N}$. Clearly, $\mathrm{RS}_{f_c}(\mathcal{N}, \mathbf{m_0})$ corresponds to the controlled system using the Banker's approach. Let us now see the costs.

1. Since $\forall p \in P_S \, . \, |f(p)| = 1$, then $\prod_{p \in P_S} |f(p)| = 1$ and the time cost of checking f_c–safeness is $O(|P_S| \cdot \log(|P_S|) \cdot |P_R|)$.
2. In order to compute the f_c function, each $p \in P_S$ has to be visited and for it, $\mathbf{Y_R}[p]$ needs to be processed. Therefore, computing the f_c function is $O(|P_R| \cdot |P_S|)$.

The global look–ahead version

This version corresponds to the first intuitive improvement shown and developed in Section 4.4.1 ([TCE00]). Let us consider the function f_{gl} defined as

$$\begin{cases} f_{gl}(p)(r) = \max_{F \in \mathcal{T}(p)} \{\mathbf{PNR_F}(p)\}, & \forall p \in P_S, \forall r \in P_R \\ f_{gl}(p) = \mathbf{0}, & \forall p \in P_0 \end{cases}$$

This function associates to each state place p the supremum of the resources needed along all the paths (composed of state places) joining p and the corresponding idle state place. Obviously f_{gl} also verifies conditions to be a *bffnr* for $\mathcal{N}$. Let us now see the computation costs.

1. As in the previous case, f_{gl}–safeness checking is $O(|P_S| \cdot \log(|P_S|) \cdot |P_R|)$.
2. The computation of f_{gl} for each state place $p \in P_{S_i}$ needs the computation of the set of paths composed of state places in $\mathcal{N}_i$, and for each one of them, $\mathbf{Y_R}[p]$ must be compared. This can be done in $O(|P_R| \cdot (\sum_{i=1}^{n} (|P_{S_i}| \cdot (|P_{S_i}| + |T_i|))))$, using any breadth–first search algorithm [CLR90].

Notice that in both cases the cost is polynomial.

A partial look–ahead version

This version corresponds to the second intuitive improvement developed in section 4.4.1, and was also presented in [TCE00]. Let us consider the function f_{pl} defined as

$$\begin{cases} f_{pl}(p) = \{\mathbf{PNR_F}(p) | F \in \mathcal{T}_s(p)\}, & \forall p \in P_S \\ f_{pl}(p) = \mathbf{0}, & \forall p \in P_0 \end{cases}$$

Then, a multi–set is associated for each simple path joining p and the corresponding idle state place, ensuring resources to follow such path, for each state place $p \in P_S$. Clearly, f_{pl} is a $bffnr$ for $\mathcal{N}$. Let us now see the costs.

1. Checking f_{pl}–safeness is $O(|P_S| \cdot \log(|P_S|) \cdot |P_R| \cdot \left|\prod_{p \in P_S} f(p)\right|)$.

   It is important to remark the fact that this time can be non–polynomial, as opposed to the two previous cases. In any case, a bound for the safeness–checking time is known a–priori and, then, it can be used to know if this time is enough to meet the real–time constraints imposed by the system.

2. In order to compute f_{pl}, the algorithm of Johnson [Joh75] can be adapted to suit our needs. This algorithm was proposed for the computation of all the elementary circuits of a directed graph. The algorithm can be adapted as follows. Let $p \in P_{S_i}$; we need to compute all the simple paths joining p and p_{0_i}. Consider the state machine containing p. Remove every transition $t \in {}^\bullet p$. Add a new transition t_p joining p_{0_i} and p (adding the arcs (p_{0_i}, t_p) and (t_p, p)). Apply the algorithm of Johnson to compute the elementary circuits in the transformed state machines and discard those not containing p_{0_i}. According to [Joh75] the cost for a given $p \in P_{S_i}$ is $O((|P_{S_i}| + |T_i|) \cdot \sum_{p \in P_{S_i}} |\mathcal{T}_s(p)|)$, which gives. $O(\sum_{i=1}^{n}((|P_{S_i}|+|T_i|) \cdot \sum_{p \in P_{S_i}} |\mathcal{T}_s(p)|))$ for the whole net.

Then, since for each $p \in P_S$, $f_c \geq f_{gl} \geq f_{pl}$ and according to Proposition 62, $\mathrm{RS}_{f_c}(\mathcal{N}, \mathbf{m_0}) \subseteq \mathrm{RS}_{f_{gl}}(\mathcal{N}, \mathbf{m_0}) \subseteq \mathrm{RS}_{f_{pl}}(\mathcal{N}, \mathbf{m_0})$. Figure 4.2 shows the reachability graph of the net in Figure 4.1. Each node corresponds to a reachable state in the uncontrolled system. Notice that some deadlocks are reachable (for instance, marking $\#9$). Some markings leading inevitably to a deadlock are also reachable (for instance, markings $\#21$ and $\#4$). Among the set of reachable markings:

- those with a “$-$” mark form the set of reachable markings when the f_c $bffnr$ is used (the original version),
- those with a “$+$” mark must be added to them in order to obtain the states reachable when f_{gl} is used,
- finally, the markings with a “$*$” mark must be added to obtain the set of reachable markings for the f_{pl} function.

Let us now present how this framework allows the introduction and study of alternative approaches.

Another polynomial versions

The general framework that has been presented allows the development of a wide family of deadlock avoidance control policies, as many as $bffnr$’s we are able to establish.

Let us consider now the function f_{sp} defined as

$$\forall i \in I_{\mathcal{N}} \,.\\ \begin{cases} f_{sp}(p) = \{\boldsymbol{PNR}_{\boldsymbol{F_{sp}}}(p) | F_{sp} \text{ the shortest path joining } (p, p_{0_i})\}, & \forall p \in P_{S_i} \\ f_{sp}(p) = 0, & \forall p \in P_0 \end{cases}$$

Function f_{sp} associates to each place the supremum of the multi–set of resources needed if a part follows the shortest path joining p and the corresponding idle state place. Let us now see the costs.

1. Since for each state place p, $|f_{sp}(p)| = 1$, f_{sp}–safeness checking complexity is the same as for f_c and f_{gl}.
2. The computation of f_{sp}, needs the shortest path between p and the corresponding idle state place for each $p \in P_S$. This can be done applying the single–source shortest path algorithm [CLR90]. So, computation of the f_{sp} function is $O(\sum_{i=1}^{n}(|P_{S_i}| \cdot (|P_{S_i}| + |T_i|))$.

In a similar way, we could consider the function mapping to each state place the longest path, or the path corresponding to the shortest processing time, or the path corresponding to the use of the cheapest resources, any of them, etc. All of them would require a polynomial time for safeness checking. Moreover, the computation of the $bffnr$ will also be of polynomial complexity (they correspond to graph algorithms of polynomial complexity).

Finally, notice that $f_c \geq f_{gl} \geq f_{sp} \geq f_{pl}$ and in consequence, we can state that $\mathrm{RS}_{f_c}(\mathcal{N}, \mathbf{m_0}) \subseteq \mathrm{RS}_{f_{gl}}(\mathcal{N}, \mathbf{m_0}) \subseteq \mathrm{RS}_{f_{sp}}(\mathcal{N}, \mathbf{m_0}) \subseteq \mathrm{RS}_{f_{pl}}(\mathcal{N}, \mathbf{m_0})$. Another interesting remark is that adopting the concept of Banker's–like algorithm presented in this chapter, no $bffnr$ can be more permissive than f_{pl}.

4.4.3 A dynamic approach

A $bffnr$ function is a way to associate information to each place about the needs of resources ensuring that a set of paths can be followed until sequential termination using the set of free resources.

This set of paths is chosen in a static way, independently of any reachable marking. However, there is an alternative way which consists in choosing the paths in a dynamic way, depending on the current marking of the set of resources. For this, we are going to present an alternative way of testing if at a reachable marking a process can terminate, based on a graph algorithm. The algorithm determines at a given reachable marking if there exists a path that can be followed with the current free resources. As it will be shown, this new method will be as permissive as the partial look–ahead solution, but with a polynomial cost.

Let us first introduce some more terminology.

Definition 64 *Let* $\langle\mathcal{N}, \mathbf{m_0}\rangle$, $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C}\rangle$, *be a marked* S^*PR. *Let* $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$. *An active process* $a \in \mathcal{A}_\mathbf{m}$ *is* $\mathbf{m}_R$–Terminable *if, and only if, there exists a path* $F_a = (q_0 t_1 q_1 \ldots q_{k-1} t_k q_k) \in \mathcal{T}_S(p)$, $(q_0 = \pi_\mathbf{m}(a), q_k \in P_0)$, *such that* $\mathbf{m} \xrightarrow{t_1 t_2 \ldots t_k} \mathbf{m}^a$. □

Note 65 *The reached marking,* $\mathbf{m}^a$, *will be as follows:*

- $\mathbf{m}^a[q_0] = \mathbf{m}[q_0] - 1$,
- $\mathbf{m}^a[q_k] = \mathbf{m}[q_k] + 1$, $k > 0$,
- $\mathbf{m}^a[p] = \mathbf{m}[p], \forall p \in P_S \setminus \{q\}$,
- $\mathbf{m}^a_R = \mathbf{m}_R + \mathbf{Y_r}[q_0]$. □

A path like F_a is said to be $\mathbf{m}_R$–*Executable* for the process a. Notice that a process is $\mathbf{m}_R$–terminable (at marking $\mathbf{m}$) if there exists a path joining the state place where the process stay with the corresponding idle state, and this path can be followed by the part using the available free resources plus those that at present are allocated to the process itself.

In the partial look–ahead approach the selection of the route that the part needed to follow in order to finish its processing was done considering all the different available paths from a given place to the end of the processing; with the information about all of these paths, one of them can be chosen as in Definition 64. Let us now show the conditions under which a path is executable.

Proposition 66 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle, \mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, \mathbf{C} \rangle$*, be a marked* S^*PR*. Let* $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$*, let* $a \in \mathcal{A}_\mathbf{m}$ *and let* $F_a = (q_0 t_1 q_1 \ldots q_{k-1} t_k q_k) \in \mathcal{T}_S(p)$*,* $(q_0 = \pi_\mathbf{m}(a), q_k \in P_0)$*.* F_a *is* $\mathbf{m}_R$-Executable *for* a *if, and only if,* $\forall \alpha \in \{1 \ldots k\}, \mathbf{m}_R + \mathbf{Y_R}[q_0] - \mathbf{Y_R}[q_\alpha] \geq 0$*.*

Proof

$\Longrightarrow$*) Let us assume that* $\mathbf{m} \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{t_2} \mathbf{m}_2 \ldots \xrightarrow{t_{i-1}} \mathbf{m}_{i-1} \xrightarrow{t_i} \mathbf{m}_i \ldots$*.*

By contradiction, let i *be the minimal index such that* $\mathbf{m}_R + \mathbf{Y_R}[q_0] - \mathbf{Y_R}[q_i] \ngeq 0$*. Firing* $t_1 t_2 \ldots t_{i-1}$*,* $\mathbf{m}_{i-1}$ *is reached, and* $\mathbf{m}_{(i-1)_R} = \mathbf{m}_R + \mathbf{Y_R}[q_0] - \mathbf{Y_R}[q_{i-1}]$*. Notice that* $\mathbf{m}_{i-1}$ *enables* t_i*,* $\mathbf{m}_{(i-1)_R} + \mathbf{Y_R}[q_{i-1}] - \mathbf{Y_R}[q_i] \geq 0$*, which is equivalent to say that* $\mathbf{m}_R + \mathbf{Y_R}[q_0] - \mathbf{Y_R}[q_{i-1}] + \mathbf{Y_R}[q_{i-1}] - \mathbf{Y_R}[q_i] \geq 0$*. This is clearly a contradiction.*

$\Longleftarrow$*) Let us prove by induction over the prefix of* $t_1 \ldots t_k$ *which is fireable from* $\mathbf{m}$ *that* F_a *is* $\mathbf{m}_R$-Executable.

1. *Since* $\mathbf{m}_R + \mathbf{Y_R}[q_0] - \mathbf{Y_R}[q_1] \geq 0$*,* t_1 *is both process– and resource–enabled. Therefore,* t_1 *can be fired.*

2. *Let us now assume that* $\mathbf{m} \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{t_2} \mathbf{m}_2 \ldots \xrightarrow{t_{i-1}} \mathbf{m}_{i-1}$*. First, since* $t_{i-1} \in {}^\bullet q_i$*,* $\mathbf{m}_{i-1}[q_i] > 0$ *(*t_i *is process–enabled). Moreover,* $\mathbf{m}_{R_{i-1}} = \mathbf{m}_R + \mathbf{Y_R}[q_0] - \mathbf{Y_R}[q_{i-1}] \geq 0$*. To prove that* t_i *is also resource–enabled, we must ensure that* $\mathbf{m}_{R_{i-1}} + \mathbf{Y_R}[q_{i-1}] - \mathbf{Y_R}[q_i] \geq 0$*. This is equivalent to* $\mathbf{m}_R + \mathbf{Y_R}[q_0] - \mathbf{Y_R}[q_{i-1}] + \mathbf{Y_R}[q_{i-1}] - \mathbf{Y_R}[q_i] \geq 0$*, which is true by hypothesis.*

□

The Algorithm 4.4 checks if an active process corresponding to a reachable marking $\mathbf{m}$ is $\mathbf{m}_R$-*Executable*. Or, in other words, if the free resources are enough for the process to terminate when the rest of active processes do not move from their current states.

Complexity of Algorithm 4.4: The cost of the *for* loop is $O(|P_{S_i}| \, |P_R|)$, while the cost of looking for a simple path joining two nodes in the graph of marked nodes is $O(|P_{S_i}| + |T_i|)$ [CLR90]. Then, the cost of the function is $O(max\{(|P_{S_i}| + |T_i|\,, |P_R| \, |P_{S_i}|\})$. Since $|P_{S_i}| \leq |T_i|$ and $|P_R| \geq 1$, the cost is $O(|P_R| \, |T_i|)$.

Let $T_{max} = \max_{i \in \{1..n\}} |T_i|$. Therefore, taking into account the cost computed in Section 4.3.1, checking for safety of a given marking is $O(|P_S|^2 \cdot |P_R| \cdot T_{max})$

Algorithm 4.4

```
Function isTerminable(In $\langle\mathcal{N},\mathbf{m_0}\rangle$: a marked $S^*PR$;
                      In $\mathbf{m} \in \mathrm{RS}(\mathcal{N},\mathbf{m_0}) \setminus \{\mathbf{m_0}\}$;
                      In a: an active process) Return (iT:boolean)
——Pre: $a \in \mathcal{A}_{\mathbf{m}}, \pi_{\mathbf{m}}(a) \in P_{S_i}$
——Post: $iT$= Is process $a$ $\mathbf{m}_R$–Terminable?
Begin
    For Each $p \in P_{S_i}$
        If $\mathbf{m}_R + \mathbf{Y_R}(\pi_{\mathbf{m}}(a)) - \mathbf{Y_R}(p) \geq 0$ Then
            mark $p$
        End If
    End For
    $iT$ := a simple path of marked nodes exists
            joining $\pi_{\mathbf{m}}(a)$ and $p_{0_i}$
    Return (iT)
End
```

(clearly, since $1 \leq T_{max} \leq |T|$, it is also $O(|P_S|^2 \cdot |P_R| \cdot |T|)$ Let us recall that the Algorithm 4.2 embeds a call to the Algorithm 4.4 in order to check if a reachable marking is safe.

It is very important to note that the safeness solution presented is the most permissive one when the Banker's approach is adopted: any solution will require that at least one path could be followed until termination; if such path exists, the function *isSafe* will return *TRUE*.

Let us consider the two more permissive approaches (the one just presented and also the one based on f_{pl}.) Considering only the asymptotic complexity, it is clear that the last solution has the best behavior: it gives the most permissive control (when Banker's like approaches are considered) with a polynomial cost. However, the partial look–ahead approach can also be taken into consideration when the number of paths joining state places and their corresponding idle places is not very big; the cost of the on–line computations can be in some cases lower than the cost needed for the graph based most permissive approach.

4.5 Some numerical results

Let us present some numerical results in order to highlight how "good" are the solutions proposed in this chapter. Table 4.3 shows the results obtained when the

			f_c	f_{gl}	f_{sp}	f_{pl}
	NM	NB	NMC (%)	NMC (%)	NMC (%)	NMC (%)
dif–1	66	64	23.44	70.31	78.12	100
dif–2	1570	1568	53.51	94.58	94.90	100
dif–3	14561	14559	68.40	98.04	98.08	100
dif–4	91238	91236	78.24	99.17	99.17	100
fil–4	81	79	54.43	100.00	100.00	100
fil–5	243	241	50.62	100.00	100.00	100
fil–6	729	727	50.89	100.00	100.00	100

Table 4.3: Some empirical results

considered Banker's like deadlock avoidance policies have been applied to some particular cases. There:

NM corresponds to the number of states of the uncontrolled system (when some deadlocks can appear);

NB corresponds to the most permissive controlled system (with no deadlock);

$f_{(*)}$ correspond to the percentage of the markings allowed with respect to NB of each one of the considered control policies. Notice that the column represented by f_{pl} will be also valid for the control policy presented in Section 4.4.3 because it can be considered as the same one but computed in different ways.

Each row corresponds to a different S^*PR net.

- The rows labelled as *dif-i*, $i \in \{1..4\}$ correspond to the net in Figure 4.4, where different initial markings are considered; the initial marking of row i-th is i times the initial m marking shown in the figure.

- The rows labelled *fil-i*, $i \in \{4..6\}$ correspond to the dining philosophers problem (the model corresponding to one philosopher is shown in Figure 4.3).

In general, and not only for these examples, it can be said that any of the proposed $bffnr$ improves in a very clear way the original approach. It is also important to note that, even if in the shown cases f_{pl} gives the more permissive solution, this is not always the case.

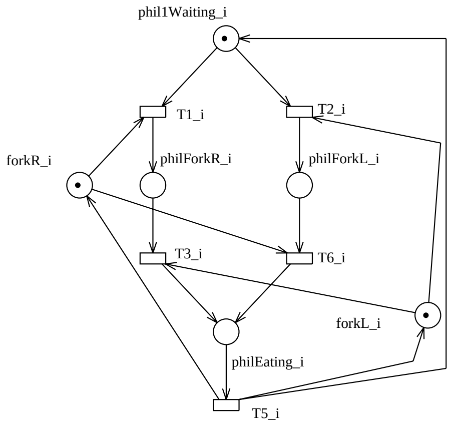


Figure 4.3: Petri net model of a philosopher with decisions

4.6 Conclusions

In this chapter the class of S^*PR nets has been introduced. This class is adequate for the modeling and control of MT–NO–RAS, extending the S^4PR nets so that the internal cyclic behavior in the structure of the involved processes can be modeled. Doing so, S^*PR nets can deal with the most general class of S-RAS.

To deal with deadlock problems in this class of systems an avoidance point of view has been adopted. It is based on the adaptation of the underlying ideas of the well–known Banker's algorithm to the specific characteristics of these systems, taking advantage of the knowledge of the complete life cycle of each one of the involved processes that S^*PR systems provide.

First, a general framework for different Banker's–like solutions has been developed, based on the concept of bound functions of the future needs of resources that a process can demand for. Some interesting properties of these bound functions of the future needs of resources have been proven. In particular, a monotonicity property on the number of allowed states and provides a way to compare two given functions and to select the most permissive one. The property can be used to reach an adequate trade–off between permissiveness of the controlled system and computational cost of the (on–line) application of the avoidance method, that the designer must solve according to real time requirements.

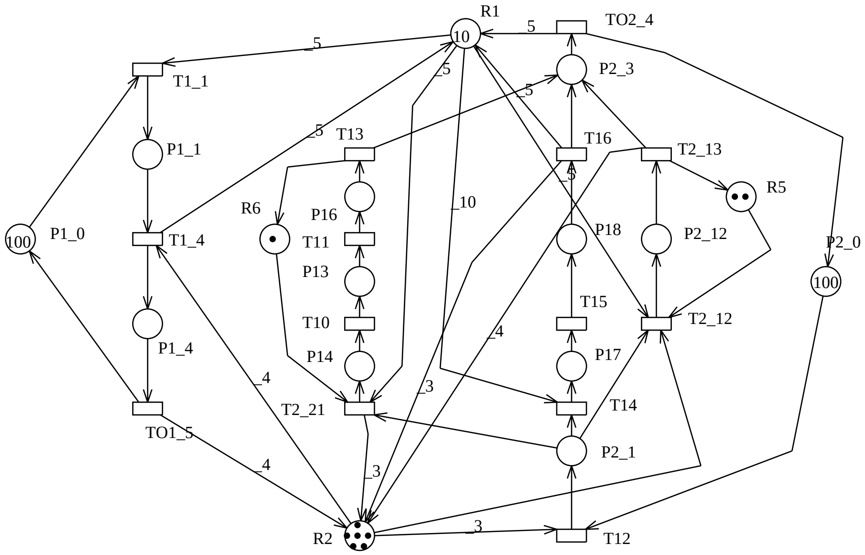


Figure 4.4: S^*PR used for the numerical experiments

Since the most permissive of the presented bound functions can be of non-polynomial cost, the chapter later concentrates on how to obtain an efficient algorithm for this case. In this sense, a polynomial cost solution has been developed based on some graph algorithms. This last approach can be considered as the most permissive when an Banker's approach is adopted. It provides solutions of polynomial cost for some cases in which the approach based on bound functions will need an exponential time. However, the price we can have to pay in other cases is more time spent in the on–line computations. This means that, in many cases, the approach based on bound functions of future needs of resources will be more adequate, specially in those systems with a polynomial number of paths joining a state place and its corresponding idle place and where real time processing requires a fast answer.

Chapter 5

Computing minimal siphons in S^4PR nets: a parallel solution

Abstract
Siphons are related to liveness properties of Petri net models. This relation is strong in the case of resource allocation systems. Siphons can be used in these systems in order to both, characterize and prevent/avoid deadlock situations. However, the computation of these structural components can be very time consuming or, even, impossible. Moreover, if, in general, the complete enumeration of the set of minimal siphons must be avoided (there can exist an exponential number of such components), some deadlock prevention methods rely on its (complete or partial) computation and enumeration. The special syntactical constraints of the classes of resource allocation systems (remember that S^4PR is a class of RAS) can help in developing specific algorithms to compute siphons in a more efficient way.

In this chapter we are going to present an adaptation of the algorithm proposed in [BM94] to the class of S^4PR. The proposed solution has a parallel nature. Some empirical results will be presented which try to show the interest of the proposed method.

5.1 Introduction

Siphons and traps have interesting properties related to the token distribution in a Petri net. In particular, siphons are closely related to liveness properties, as we have seen in previous chapters. It is easy to prove a classical property: if there exists a total deadlock in an ordinary Petri net, there exists an empty siphon ([Com72]). In non–ordinary Petri nets this idea can be generalized: if there exits a total deadlock, then there exists a siphon with a "deficient marking" ([Bra83]). Moreover, for some special classes of nets, siphons can be used to characterize liveness (free choice nets [ES92], extended free choice nets [BM92], asymmetric choice nets [BPP96], Petri nets with resources (PNR) [JXH00], system of simple sequential processes with general resource requirements (S^3PGR^2) [PR01], and S^4PR nets as shown in previous chapters). Finally, these structural components have been used to implement deadlock control policies in [ECM95, BA95, TGVCE00, TCE99, JXH00, IMA02]. Since the techniques introduced in these works rely on minimal siphons, some (efficient) methods to compute them are needed.

It is well known that, in general, the computation of minimal siphons for a Petri net is a NP–complete problem (see, for example [KB92, YW99b]). However, we wanted to know how time consuming would be computing siphons for S^4PR nets, and how parallelization could improve this computation. Two facts have been considered for this computation in S^4PR nets.

- The special structure of this class of nets.
- Only siphons containing resource places are interesting from the point of view of deadlock prevention policies.

Among the wide set of methods for the computation of minimal siphons we have chosen the one proposed in [BM94] for the following reasons:

- As it will be shown, the method can be easily implemented in a parallel way. This is specially interesting, since the computation of families of siphons can be very time consuming.
- It is oriented to the computation of the set of siphons containing a given place (set of places). This makes it interesting for those control policies that rely on the computation of families of siphons ([ECM95, BCZ97, TCE99, IMA02].)

It is clear that, when possible, the computation of big families of siphons must be avoided. In the case of S^4PR nets, the deadlock prevention methods proposed in Chapter 3 followed this approach computing one siphon at each step by means of an integer programming problem. However, in other cases this cannot be avoided, and this makes interesting to obtain efficient implementations.

5.2 Some methods for the computation of siphons

Researchers have considered and studied different methods for finding siphons and traps. Among them let us present the main ones, that we will classify based on the underlying techniques used for their computation:

Algebraic methods

In this group we include those methods that compute families of siphons (traps) by means of the solution of a set of linear equations or inequalities. The system is based either in the direct use of the net incidence–matrix or in a transformation of it. The following methods can be included in this group:

- The method presented in [Tou81, AT85] is based on the resolution of a system of linear inequalities. They are obtained via a transformation of the net by means of the 'unfolding' of some transitions. The support of the minimal P–Semiflows of this unfolded net corresponds to a generating family of siphons of the original one: a family of siphons is a generating family if any siphon can be constructed by means of the union of siphons of the family.

- In [Lau87, ES92] the proposed method transforms the net via the 'expansion' of 'shared places' (places with $|^\bullet p| > 1$ or $|p^\bullet| > 1$) and the addition of some new restrictions that can be seen as new transitions. Siphons and traps are obtained as the support of some P–Semiflows of this modified net. The obtained solutions are not a generating family of all the siphons (traps) of the net, but a generating family of the strongly connected ones.

- In [EC91, ECS93] the proposed method is based on solving a set of inequalities, whose associated matrix is a slight transformation of the net incidence matrix. The transformation just changes the weights of some arcs. The supports of the solutions of such inequalities form a generating family of the set of siphons (traps).

- The method proposed in [BM94] uses an algebraic characterization, based on the direct application of the definition of siphon (trap) to an special version of the incidence–matrix, the *sign incidence matrix*. This matrix is like the incidence matrix where the actual values of the arc weights have been substituted by signs that represent whether a place is an entry, output, or both for each transition. With this, they compute a basis of minimal siphons (traps). This method is further improved in [YW99a].

Methods based on graph theory

This class includes those methods that directly use the graph representation of the Petri net to compute siphons:

- In [BL89] a simple characterization of minimal siphons in terms of path properties is presented. The authors presented a polynomial algorithm to determine if a given set of places is a siphon based on *alternating circuits*. This work was later improved in [BM92] for the class of bounded *extended free–choice nets* and *non self–controlling nets*. They are not used to compute siphons, but to prove liveness and other related properties.

- In [WYT98] some methods based on a branch–and–bound algorithm and graph theory for the computation, if any, of minimal siphons that contain a given subset of places were presented. With this method minimal siphons containing a given set of places can be computed.

- In [JPH99] another approach based in an algorithm that uses recursive depth–first search in the underlying graph structure of the Petri net is proposed. The method can compute the set of minimal siphons (traps).

Methods based on logic formulae

- In [Sil85] a method based on characterizing siphons using logic formulae is presented. For this some logical equations whose variables represent whether a place belongs to a siphon or not are constructed. This method was adapted to be used in a more efficient way in Section 3.3.2, in Chapter 3.

- In [MB90] the presented methods are based on the satisfactibility of some Horn clauses related to these structural components: the idea is to represent the siphon condition via logic formulas, and then to transform them into Horn clauses. They can compute minimal siphons containing a specified subset of places.

- Finally, in [Val99] the use of *ordered binary decision diagrams* is proposed to manage and solve the logic equations representing the siphon (trap) condition. The computation of minimal siphons and traps, and also the way to verify Commoner's property, is shown as an example of the proposed techniques.

	T1	T2	T3	T4	T5	T6	T7
P1_1	+	−	0	0	0	0	0
P1_2	0	+	−	0	0	0	0
P1_3	0	0	+	−	0	0	0
P2_1	0	0	0	0	+	−	0
P2_2	0	0	0	0	0	+	−
P1_0	−	0	0	+	0	0	0
P2_0	0	0	0	0	−	0	+
R1	−	+	0	0	0	−	+
R2	0	−	−	+	−	+	0

Table 5.1: Sign incidence matrix of net of Figure 2.6

The selected method

From these methods, we have selected the one presented in [BM94], mainly due to the following two facts: each row in the *sign incidence matrix* corresponds to a place, and the method operates based on selecting one of these rows. In consequence we can select which place–siphons compute, avoiding the non–interesting ones; moreover, this computation can be done in an independent way for each place, introducing an easy way to add parallelism to the computation.

The method uses the sign incidence matrix, which is the representation of the underlying graph.

Definition 67 ([BM94]) *Let $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$, be a Petri net with $|T| = n$ and $|P| = m$. The* sign incidence matrix $\mathbf{A} = [a_{ij}]$ *is an $m \times n$ matrix, whose elements are:*

- $a_{ij} = +$ *if place i is an output place of transition j.*
- $a_{ij} = -$ *if place i is an input place of transition j.*
- $a_{ij} = \pm$ *if place i is both input and output place of transition j.*
- $a_{ij} = 0$ *otherwise.* □

For example, for the net depicted in Figure 2.6 whose incidence matrix is shown in the Table 3.2, the *sign incidence matrix* can be seen in Table 5.1.

Since each row represents the set of input/output transitions of a place, the adequate arithmetic operations with these values need to be defined in order to be usable for siphon computation.

Definition 68 ([BM94]) *The addition, denoted by* $\oplus$*, is a commutative binary operation on the set* $E = \{+, -, \pm, 0\}$*, defined as follows:*

- $+ \oplus - = \pm$
- $a \oplus a = a, \forall a \in E$
- $\pm \oplus a = \pm, \forall a \in E$
- $0 \oplus a = a, \forall a \in E$ □

Basically this arithmetic allows to cancel any positive entry with a negative or a '$\pm$' valued one.

For example, if we use the sign incidence matrix of Table 5.1, we can sum the rows corresponding to places $P1\_1$ and $R1$, obtaining:

	T1	T2	T3	T4	T5	T6	T7
$P1\_1$	$+$	$-$	0	0	0	0	0
$R1$	$-$	$+$	0	0	0	$-$	$+$
$P1\_1 \oplus R1$	$\pm$	$\pm$	0	0	0	$-$	$+$

The '$\pm$' symbols in the first and the second columns represent that, in the net, ${}^\bullet\{P1\_1 \cup R1\} \cap \{P1\_1 \cup R1\}^\bullet = \{T1, T2\}$. Moreover, the '$-$' symbol in the sixth column represents the fact that $T6 \in \{P1\_1 \cup R1\}^\bullet$, but it is not an input transition of this set; finally, the '$+$' symbol in the seventh column represents that $T7$ is an input transition of this set of places, but it is not an output transition.

Then, if a positive entry represents an input transition for the considered set of places, a negative one represents an output transition, and a $\pm$–signed one represents a transition that is both input and output of the set, we can try to cancel positive signs with the addition of places that can produce '$\pm$' and negative ones in order to have a siphon.

The siphons can be obtained with the result presented in the following theorem. It is based on the selection of a set of rows (places) of the *sign incidence matrix*.

Theorem 69 ([BM94]) *Let $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ be a Petri net with $|T| = n$ and $|P| = m$. A subset of places, $S = \{p_1, p_2, ..., p_k\} \subseteq P$, is a siphon if and only if the addition of the k row vectors of the sing incidence matrix of $\mathcal{N}$, $A_1 \oplus A_2 \oplus ... \oplus A_k$, contains no '$+$' entries, where A_j denotes the row vector corresponding to place $p_j, j = 1, 2, ..., k$.* □

Given the previously defined arithmetic, if the sum of these rows does not contain '$+$' entries, this means that for each '$+$' in a row (the place is the output of the corresponding transition), there is another row with a '$-$', or a '$\pm$' (there exists a place of the net that is the input of this transition).

For example, if we use the sign incidence matrix of Table 5.1, we can sum the rows corresponding to places $P1\_1$, $P2\_2$ and $R1$, obtaining:

	T1	T2	T3	T4	T5	T6	T7
$P1\_1$	$+$	$-$	0	0	0	0	0
$P2\_2$	0	0	0	0	0	$+$	$-$
$R1$	$-$	$+$	0	0	0	$-$	$+$
$P1\_1 \oplus P2\_2 \oplus R1$	$\pm$	$\pm$	0	0	0	$\pm$	$\pm$

So they are a siphon, since now ${}^\bullet\{P1\_1, P2\_2, R1\} = \{P1\_1, P2\_2, R1\}^\bullet$ (they are also a trap).

The method proposed in [BM94] is based on the iterative application of the previous 'cancellation' of '$+$' signs by means of the addition of matrix rows. Any grouping of places (addition of a set of rows) with no '$+$' signs corresponds to a siphon.

Since the approach is based on selecting a matrix row, the corresponding place will be always present in the obtained siphon (if any). In consequence, some of the obtained siphons can be non–minimal. To clarify this, let us show the definition of *place–minimal siphons* which are the structural component that this method is able to compute.

Definition 70 *A* place–minimal siphon *with respect to place p is a siphon containing place p and a minimal set of places.* □

Now it is clear that the algorithm can be adapted to our problem: the method can select just the rows corresponding to the places that are of interest for the control policy (resources).

Since the method computes place–minimal siphons (let us remark that they can be non–minimal), some filtering will be needed in order to obtain the minimal ones.

The specific syntactical constraints of S^4PR nets pointed towards the use of the method presented in [BM94] to compute siphons in this class of nets. The following facts supported that decision:

1. In resource allocation system all the 'interesting' siphons must contain resource places (Theorem 26 in Chapter 2). We can expect a reduction on the computation costs.

2. Finally, let us recall that the method starts selecting a row of the sign matrix in order to compute place–minimal siphons related to it; in consequence, it is straightforward to parallelize the method: the computation of the siphons related to a subset of places can be easily distributed among different processors.

In order to confirm these ideas, a first experiment was done comparing this method with Lautenbach's method [Lau87] when applied to a set of S^4PR nets. Table 5.3 (page 158) shows the results of the experiment (the meaning of each row is explained in Section 5.4). From the results, it seems that the Boer-Murata's method is well adapted to the problem. In the table we can see that when the size of the problem increases, the adopted method tends to be better than the Lautenbach's one.

An alternative method to compute siphons containing a set of places was presented in [WYT98]. It was discarded because of its poor performance in our preliminary tests. Finally, the improvements to the Boer-Murata's method introduced in [YW99a] were also considered. However, these improvements are not of interest in S^4PR nets, since when applied to nets of this class no better results were obtained.

5.3 The implementation

Since the original algorithm was able to obtain the place–minimal siphons for any place in the net it is obvious that the computation of the siphons that involve resources can be easily parallelized, computing place–minimal siphons for different resources in different processors. In this way, as many sets of place–minimal siphons as available processors can be computed in parallel. The approach used to

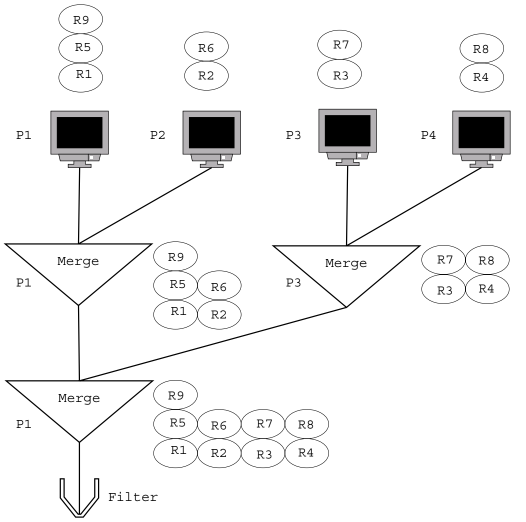


Figure 5.1: Schematic representation of the computation with four processors and nine resources

organize the parallel computation can be seen in the literature with the name of *divide and conquer* since it follows the well known strategy for sequential programs (see for example [Akl97][1].)

Let us sketch the basic steps of the parallel implementation. Algorithm 5.5 shows the skeleton of the program implemented in each one of the processors: it is based on the application of three steps.

- **Step 1**

  First of all, the set of resources is distributed among the available processors, in such a way that each processor has to compute siphons for a similar number of resources. For this, an ordered list of active processors is created an maintained.

  Let us consider, for instance, the nine philosophers problem. There will be nine resources (in this case the resources are the forks, and there is one fork for each philosopher). Moreover, let us suppose that we have four available processors. The first step would be to distribute the places modelling the forks among the processors. Figure 5.1 shows a possible way of distribution

[1] There is a pattern–based approach [MMS01], where the authors identify, among others, the *DivideAndConquer* pattern in the context of parallel application programs design.

Algorithm 5.5

```
Function computeSiphons(In $\mathcal{N}$: a $S^4PR$) Return : (S: set)
——Pre: TRUE
——Post: $S$ is the set of minimal siphons of $\mathcal{N}$ that contain resource places
Begin
    $process\_Id := get\_identifier$
    $num\_Processes := get\_number\_of\_processes$
    If (process_Id = 0) Then
        —— Process 0 is the last one to terminate
        Start_Clock
        Send Signal to the other processes to start
    Else
        —— The other processors wait for the Process 0
        Wait Signal from process 0
    End If
    Read the Petri Net
    $S_{process\_Id}$ := compute place–minimal siphons for the resources assigned to process_Id
    While $num\_Processes > 1$
        If $process\_Id\ mod\ 2 = 0$ Then
            —— Processes in charge of merging
            Wait for the process with identifier $process\_Id + 1$
            Obtain the Siphons from $process\_Id + 1$
            $S_{process\_Id} = S_{process\_Id} \cup S_{process\_Id+1}$
        Else
            —— Processes that send their siphons and terminate
            Contact with process $process\_Id - 1$
            Send the Siphons to $process\_Id - 1$
            Terminate
        End If
    End While
    —— Only process 0 reaches this point
    $S$ = minimal siphons from $S_0$
    Stop_Clock
    Return (S)
End
```

of resources among four processors and sketches the execution of the algorithm. There, two resources have been assigned to processors two, three, and four, and three resources to the first processor.

Each processor will be in charge of computing the set of place–minimal siphons related to the resources assigned to it. In the example, processor $P1$ will compute the place–minimal siphons related to forks $R1$, $R5$ and $R9$; processor $P2$ the ones corresponding to $R2$ and $R6$; processor $P3$ the ones corresponding to $R3$ and $R7$; finally, processor $P4$ will compute the ones related to $R4$ and $R8$.

- **Step 2**

  When each processor terminates its computation, it tries to contact with one of the remaining active processors. To organize this a simple strategy is proposed: the processors that are in an odd-numbered position in the list are in charge of merging, while the others just have to communicate with the corresponding ones in charge of merging, send the data, and terminate (they are then eliminated from the list of active processors); each even–numbered processor will contact with its predessor in the list[2].

  The merging operation is very simple: the union of both sets of siphons. The goal is to obtain all the siphons, so this merging will continue until just one processor remains active.

  In the example of Figure 5.1 processor $P1$ merges the siphons it has computed with the ones computed by processor $P2$ (obtaining the place–minimal siphons corresponding to resources $\{R1, R5, R9, R2, R6\}$); when this merging operation terminates, processor $P2$ is eliminated from the list of active processors. Processor $P3$ merges the siphons it computed with the ones obtained by processor $P4$ (in this way, the set of place-minimal siphons corresponding to resources $\{R3, R4, R7, R8\}$ is constructed); processor $P4$ is also eliminated from the list when this operation terminates.

  When one of the processors in charge of merging terminates, and while there exist more active ones, it tries to contact with other processor (again depending on its position in the list of active processes, it will have to execute either a merging operation or just sending its computed data and terminate). This step will be repeated until just one processor remains active, which is the one that will have all the place–minimal siphons.

[2] We will see later that it is a naive approach but useful for the purposes of our experiment.

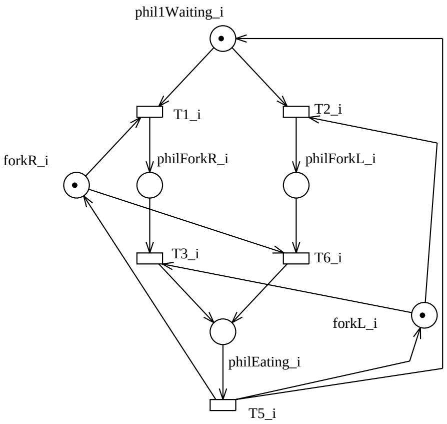


Figure 5.2: Petri net model of the i–th philosopher

In the considered example, a second round is needed, where processor $P1$ merges its place–minimal siphons with the ones obtained by processor $P3$.

- **Step 3**

  When the previous stages terminate there is a unique remaining active process, which contains all the place–minimal siphons (related to resource places.) Since some of them could be non–minimal, the last stage consists of a filtering operation to eliminate the non–minimal siphons.

The merging and filtering steps require the efficient storage and manipulation of sets of siphons (sets of sets of places). To do that, *Ordered Binary Decision Diagrams* (OBDDs) [Bry92] have been used, transforming these merging and filtering operations into the manipulation of symbolic boolean functions.

An immediate way of decreasing the number of sets (siphons) that have to be manipulated consists in doing the filtering operations before executing a new merging. However, our preliminary experiments with this approach showed that it was quite expensive, so we decided to do the filtering only at the last step.

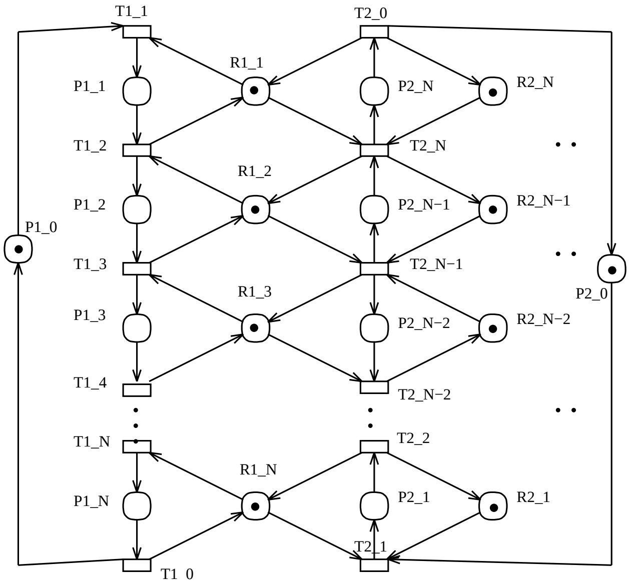


Figure 5.3: Petri net model of two sequential processes using resources

5.4 The experiments

In order to study the behaviour of the proposed parallel version some experiments have been done. Since no benchmarks have been proposed in the literature to measure the run–time cost, three parametrised families of nets have been chosen:

- The first one corresponds to an implementation of the well known dining philosophers problem. The parameter corresponds to the number of philosophers. Figure 5.2 shows the model of the ith philosopher. Places $forkR\_i$ and $forkL\_i$ model, respectively, the state (free/engaged) of its right and left forks. The fork $forkR\_i$ will be shared with the philosopher on his right, and the $forkL\_i$ will be shared with the philosopher on his left.

  The results obtained for this family of nets are entitled *Phil* in Tables 5.3 (page 158) and 5.5 (page 164).

- The second and third classes of systems are obtained by means of the composition of a set of sequential processes: each process, at each processing step, has attached a single (and different) resource. An skeleton of the Petri nets representing two of such sequential processes can be seen in Figure 5.3.

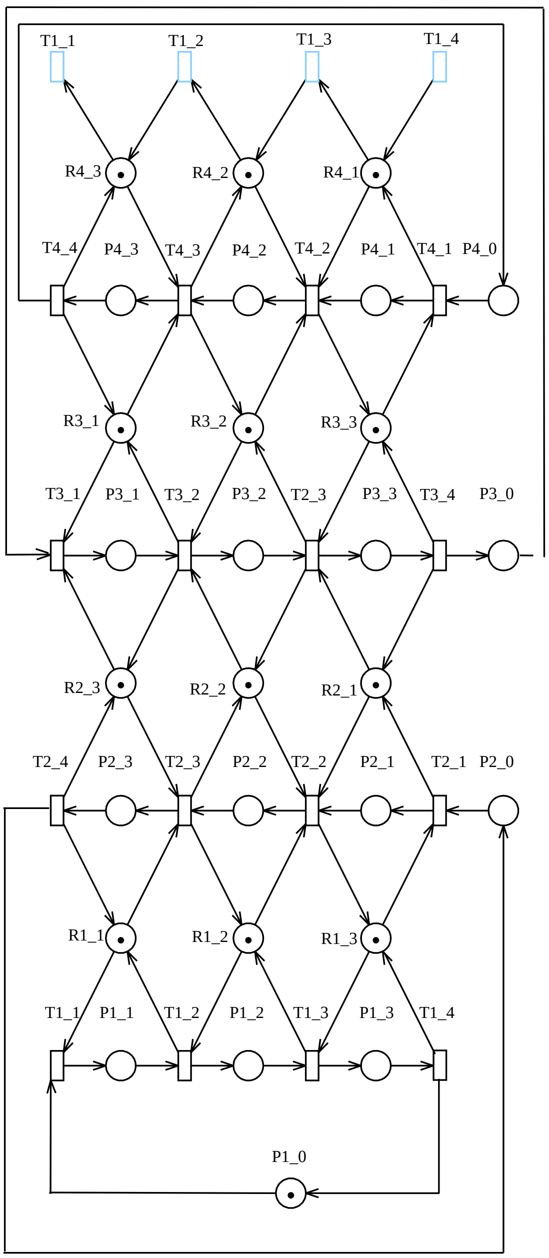


Figure 5.4: Petri net model of four sequential processes of length three

There are two ways to study size variations in these families of systems: one of them is changing the length of the process; that is, the number of processing steps (N in the Figure). The second one is changing the number of processes to be composed (in the figure two processes are shown). For the experiment, the sequential processes are composed with other processes according to the following rules:

- The first process shares its resources with the second one in reverse order: the resource used at the first step in the first process is used at the last step of the second process; the resource used at the second step of the first process is used by the one that is previous to the last step in the second process, and so on. That is, if resources are numbered (according to process one) $1, 2, ..., N$ (where N is the length), they will be used by the first process in this order and by the second one in reverse order ($N, N-1, ..., 1$).
- The second process is composed with the third one in a similar way, and the same is also true for the composition of the third with the fourth and so on, until we reach the total number of composed processes.
- Finally, the last process is composed with the first one, following the same pattern.

Using the previous skeletons, two different families of S^4PR nets have been generated, labelled as $FMSLD$ and $FMSAD$ in Tables 5.3 (page 158) and 5.5 (page 164).

- *FMSLD* nets are obtained by means of the composition of two sequential processes as the ones depicted in Figure 5.3, where the parameter is the length of each process (N in the figure). In the experiments, N is varying from 1 to 8.
- *FMSAD* nets are obtained by means of the composition of a variable number of sequential processes as the ones depicted in Figure 5.3, whose length is 3 and the parameter is the number of processes. In the experiments, the number of composed processes is varying from 1 to 11. (Figure 5.4 shows the Petri net model of the FMSAD of size 4).

These last two families have been chosen with the objective of exploring the two obvious (and different) growing directions; the first one shows a system whose resources are involved in siphons that can be of a big size, while in the second one all the siphons are rather small.

	Size	[BM94]	[Lau87]
Phil.	3 (15, 3, 46, 15)	0.35	0.02
Phil.	4 (20,4,64,20)	0.52	0.25
Phil.	5 (25, 5, 80, 25)	0.76	5.49
Phil.	6 (30,6,96,30)	1.09	194.03
Phil.	7 (35, 7, 102, 35)	1.44	4932.85
FMSLD	3 (15,6,20,12)	0.4	0.01
FMSLD	4 (20,8,26,16)	0.54	0.02
FMSLD	5 (25,10,32,20)	0.85	0.9
FMSLD	6 (30,12,38,24)	1.69	26.49
FMSLD	7 (35,14,44,28)	4.17	1012.93
FMSAD	3 (21,9,60,12)	0.93	0.02
FMSAD	4 (28,12,80,16)	2.32	0.43
FMSAD	5 (35, 15,100,20)	9.32	8.51
FMSAD	6 (42,18,120,24)	66.05	258.4
FMSAD	7 (49,21,140,28)	710.7	6337.44

Table 5.3: Comparing Lautenbach's method with Boer and Murata's one with processor

5.5 Numerical results

Two kinds of experiments have been carried out. The first class of experiments tried to see if the approach introduced in [BM94] was adequate for S^4PR nets. To do that, we have compared the time required to compute the set of minimal siphons with that method and the one proposed in [Lau87]. Moreover, since the authors of [JPH99] provided data about the temporal cost of their algorithm, we decided to include these results in the comparison for the net used as example there. The second class of experiments measured the speed–up when the parallel implementation presented above is used.

5.5.1 Comparison of the proposals in [Lau87, BM94, JPH99]

The first set of results has been obtained in a SGI Challenge L ($6\times$ R4400 200 MHz) with 512 Mb of RAM. The operating system is IRIX 6.2. The measures are in seconds.

The results can be seen in Table 5.3. This table has four columns.

1. The first column shows the problem identifier, with the following meanings for the tags:

    - Phil: the philosopher's problem.
    - FMSLD: the composition of two processes of variable length.
    - FMSAD: the composition of several processes of length 3.

2. The second column shows the parameters defining the problem size as explained before. This value is followed by a tuple that corresponds to the *total number of places, the number of resources, the number of arcs, and the number of transitions*.

3. The third column shows the run–time costs when the method in [BM94] is used.

4. The fourth column shows the run–time costs when the method in [Lau87] is used.

In the following comments we are going to say that a method is more or less appropriate for S^4PR nets. However, it must be clearly stated that we only can provide some conclusions based on the results of our experiments and that they are not general enough to obtain conclusions about all the S^4PR nets.

The experiment results show that the Lautenbach's method is better for small S^4PR nets. However, when the size of the net grows, the approach in [BM94] obtains a better performance. These results are coherent with the election done in [YW99b], where the method proposed in [BM94] was selected to improve their graph–based approach for general nets.

It must be noticed that the approach presented in [BM94] has been adapted to compute the set of place–minimal siphons related to resource places, while the method proposed in [Lau87] computes all the minimal siphons. In this sense, no conclusion about which method is better in general can be offered, since we were just trying to measure which one was more adequate for our purposes.

In a second set of experiments both methods have been applied to the net used as example in [JPH99]. This net modeled a flexible manufacturing system (the net can be seen in Figure 5.5). In their article, the authors used a reduction of that net 'in order to save the algorithm run time', and the siphons and traps were obtained for this reduced version. The results should be the same for the reduced net and for the net in the figure (provided the adequate translation from the places of the original net to the resulting reduced system). They used a PC with a Pentium II (233 Mhz) CPU processor and 32 Mb of RAM for the test, and the cost[3] of the computation of all the minimal siphons was of 931 minutes and 45 seconds (55905 seconds) and 584 minutes and 17 seconds (35057 seconds) for all the minimal traps.

Our tests were done with the whole original net, in a Pentium III (1GHz) computer with 256Mb of RAM. The operating system is Linux (kernel version 2.4.9).

[3] They do not state if this time is the total time, or the processor time.

Our implementation of the method proposed in [Lau87] for the computation of all the minimal siphons spent 2063.24 seconds of user time, 0.16 seconds of system time, and 34:23.61 of total time (2063.61 seconds). The time for the computations of all the minimal traps was 2894.71 seconds of user time, plus 1.19 seconds of system time which gives a total elapsed time of 48:20.46 (2900.46 seconds).

Since the net in Figure 5.5 is not a S^4PR, we have chosen a set of places as being (behaving like) resources, and computed the set of place–siphons (traps) for them (the name of such places starts with 'R' in the Figure 5.5). The election of such places can be difficult in the general case, and then, a second experiment has been done computing the minimal place–siphons for every place. In this way, we tried to reproduce the general case, in which there is no information available to select the adequate places to obtain all the minimal siphons from their minimal place–siphons.

In the first case the obtained results were: 8.78 seconds of user time, 0.02 seconds of system time, and a total elapsed time of 8.832 seconds for the computation of the set of minimal siphons. For the computation of all the minimal traps this approach spent 343.45 seconds of user time, 0.24 seconds of system time and 5:44.46 (344.46 seconds) total time[4].

In the second case, the computation of the set of minimal siphons took 57.17 seconds of user time, plus 0.06 seconds of system time, and a total elapsed time of 57.276 seconds. The computation of the set of minimal traps took 7893.46 seconds of user time, 5.52 seconds system time and a total elapsed time of 2:11:59.43 (7919.43 seconds). Notice that the computation of traps is more expensive in this case than with the [Lau87] method.

In order to get a more fair comparison, we also did the tests in another computer, with a PII computer with a 350 MHz processor. The results in this case for the minimal siphons computation with [Lau87] were: 1h20m11.07 of user time (4811.07 seconds), 1.22 seconds of system time, and 1h20m13.37 of total time (4813.37 seconds). For the minimal traps computation: 1h52m22.22s of user time, (6742 seconds), 1.36s of system time, and 1h52m24.18s of total time (6744.18 seconds).

Minimal siphons computation with [BM94] (selecting a sub–set of places as resources): 29.40 seconds of user time, 0.06 seconds of system time, and 29.51s of total time. Minimal traps computation: 14m38.30s of user time (878.30 seconds), 0.32s of system time, and 14m43.85s of total time (883.85 seconds).

[4] In fact, both the previous approach and this one found 179 siphons, instead of the 74 listed in [JPH99]. Checking by hand the siphons shown in the article we found several errors, so it is possible that there exists a bad transcription of the obtained results.

Experiment	Time		
Pentium II-233MHz (32Mb)			
Siphons	55905		
Traps	35057		
Pentium III-1GHz (256Mb)			
	User	System	Total
Siphons ([Lau87])	2063.24	0.16	2063.61
Traps ([Lau87])	2894.71	1.19	2900.46
Siphons (all)	57.17	0.06	57.276
Traps (all)	7893.46	5.52	7919.43
Siphons (resources)	8.78	0.02	8.832
Traps (resources)	343.45	0.24	344.46
Pentium II-350MHz (256Mb)			
	User	System	Total
Siphons ([Lau87])	4811.07	1.22	4813.37
Traps ([Lau87])	6742	1.36	6744.18
Siphons (all)	158.32	0.11	229.63
Traps (all)	14085.71	4.14	14096.41
Siphons (resources)	29.40	0.06	29.51
Traps (resources)	878.30	0.32	883.85

Table 5.4: Sketch of the times obtained for the net in Figure 5.5 with different methods and computers

Minimal siphons computation with the original approach in [BM94] (computing the place–siphons for all the places of the net): 2m38.32s of user time (158.32 seconds), 0.11s of system time, and 3m49.63s of total time (229.63 seconds). Minimal traps computation: 3h54m45.71s of user time (14085.71 seconds), 4.14s of system time, and 3h54m56.41s of total time (14096.41 seconds).

We did a last experiment limiting the amount of memory allowed to be used by the program to 32Mb, obtaining similar results.

All of these results are sketched in Table 5.4.

5.5.2 Measuring the proposed parallel implementation

The parallel version has been implemented using 12 Pentium III (1GHz) computers with 256Mb of RAM each. They are in a swithched network connected via Ethernet. The operating system is Linux (kernel version 2.4.9). To implement the parallel version we used `mpich 1.2.4` ([GL96]), a freely available implementation of the *Message Passing Interface* (MPI, [For94]). MPI provides a standard set of definitions which allow parallel programs to be implemented under the distributed memory paradigm, together with a mechanism to transfer data between processors; this mechanism is based on the use of message passing; each processor is given the ability to send and receive copies of data to and from other processors. The run–time cost has been aproximated by the run–time cost of the first processor which is the first to start the computations and the last one to terminate. To do that, a clock is reset when it starts working (sentence *Start_Clock* in Algorithm 5.5) and

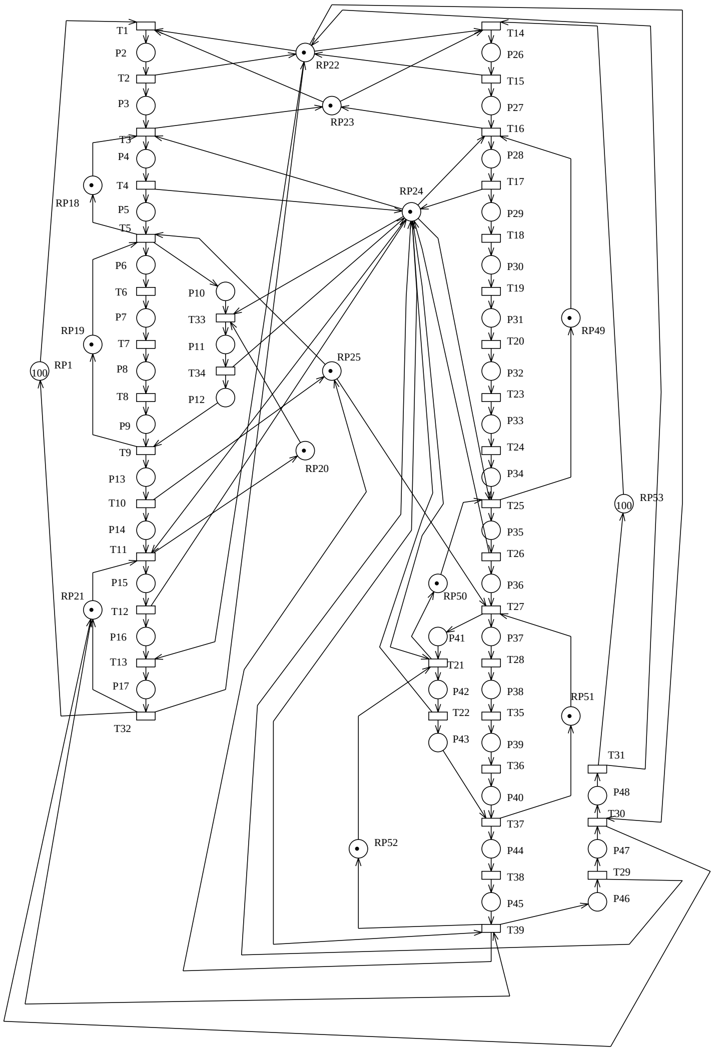


Figure 5.5: Petri net model of a flexible manufacturing system in [JPH99]

stopped when it is terminated (sentence *Stop_Clock* in Algorithm 5.5). No more processors than resources are used in any case, since we have not investigated other ways to further divide the needed computations.

We were interested in measuring the relative speed–up as a way to know the effect of adding paralelism to the application. The relative speed–up is a standard measure for parallel solutions defined as the quotient of the time spent in solving the problem with several processors divided by the time spent in solving the same problem using the same method in just one processor (see [Kuc78], for example).

The numerical results obtained in the experiments are shown in Table 5.5. The structure of such table is as follows:

1. The first column shows the problem identifier (as in the previous experiment).

2. The second column shows the parameters defining the size of the problem as explained before. This value is followed by a tuple that corresponds to the *number of places of the net, the number of resources, the number of arcs, and the number of transitions*.

3. The third and following columns show the time of our parallel implementation of Boer and Murata's method solved with 1, 2, 4, ..., 12 processors. In parentheses the relative speed–up is shown.

Empty cells represent cases where the number of processors is bigger than the number of resources. The proposed approach is not able to take advantage of the processors that exceed the number of resources.

Following the results obtained in this experiment we can conclude that parallellization clearly improves the computations: important reductions are obtained when the number of processors grows. Anyway, it is important to notice that no clear conclusion about the optimal number of processors can be established, since the growing is different for the proposed families.

The maximum size for each problem has been decided when not enough processors were available to show growing in the speed–up or when the base case (with only one processor) was very expensive.

Notice that the results are significative when the net is big enough. For small–sized nets no significative results can be obtained. They are graphically depicted in Figures 5.6, 5.7, and 5.8.

Figure 5.6 shows the program behaviour when applied to the philosophers problem. The high homogeneity of the models help to reach a good speed–up.

	Size		1	2	4	6	8	10	12
Phil	2	(10,2,32,10)	0.07 (1.00)	0.09 (0.74)					
Phil	4	(20,4,64,20)	0.27 (1.00)	0.20 (1.35)	0.14 (1.91)				
Phil	6	(30,6,96,30)	0.64 (1.00)	0.44 (1.45)	0.31 (2.09)	0.25 (2.61)			
Phil	8	(40,8,128,40)	1.20 (1.00)	0.68 (1.78)	0.42 (2.85)	0.43 (2.78)	0.50 (2.39)		
Phil	10	(50,10,160,50)	2.17 (1.00)	1.29 (1.69)	0.73 (2.96)	0.65 (3.33)	0.65 (3.33)	1.15 (1.89)	
Phil	12	(60,12,192,60)	3.83 (1.00)	1.99 (1.93)	1.18 (3.25)	0.83 (4.60)	0.77 (4.95)	0.77 (4.98)	0.66 (5.79)
Phil	14	(70,14,224,70)	6.23 (1.00)	3.33 (1.87)	2.16 (2.89)	1.65 (3.77)	1.42 (4.38)	1.48 (4.21)	1.57 (3.98)
Phil	16	(80,16,256,80)	9.88 (1.00)	5.27 (1.87)	2.79 (3.54)	2.33 (4.24)	2.08 (4.75)	2.10 (4.69)	2.02 (4.90)
Phil	18	(90,18,288,90)	15.67 (1.00)	7.94 (1.97)	4.44 (3.53)	3.18 (4.92)	2.76 (5.68)	2.24 (7.00)	2.23 (7.02)
Phil	20	(100,20,320,100)	24.82 (1.00)	12.53 (1.98)	6.62 (3.75)	4.98 (4.99)	4.55 (5.46)	4.23 (5.86)	2.74 (9.07)
Phil	22	(110,22,352,110)	38.64 (1.00)	19.37 (2.00)	10.73 (3.60)	7.10 (5.44)	5.68 (6.80)	5.67 (6.82)	4.20 (9.20)
Phil	24	(120,24,384,120)	58.66 (1.00)	29.56 (1.98)	15.22 (3.85)	9.98 (5.88)	8.17 (7.18)	7.62 (7.70)	6.44 (9.11)
FMSLD	1	(5,2,8,4)	0.07 (1.00)	0.08 (0.90)					
FMSLD	2	(10,4,14,8)	0.11 (1.00)	0.10 (1.04)	0.13 (0.84)				
FMSLD	3	(15,6,20,12)	0.19 (1.00)	0.15 (1.26)	0.15 (1.20)	1.96 (0.09)			
FMSLD	4	(20,8,26,16)	0.29 (1.00)	0.21 (1.43)	0.18 (1.65)	0.26 (1.12)	0.82 (0.36)		
FMSLD	5	(25,10,32,20)	0.48 (1.00)	0.31 (1.58)	0.24 (2.03)	0.23 (2.07)	0.59 (0.81)	1.40 (0.34)	
FMSLD	6	(30,12,38,24)	0.97 (1.00)	0.55 (1.78)	0.35 (2.77)	0.32 (3.02)	0.32 (3.09)	0.31 (3.17)	0.37 (2.64)
FMSLD	7	(35,14,44,28)	2.35 (1.00)	1.25 (1.88)	0.79 (2.98)	0.55 (4.29)	0.53 (4.41)	0.56 (4.19)	0.61 (3.87)
FMSLD	8	(40,16,50,32)	6.98 (1.00)	3.57 (1.95)	1.97 (3.54)	1.50 (4.64)	1.35 (5.16)	1.27 (5.51)	1.20 (5.82)
FMSLD	9	(45,18,56,36)	25.25 (1.00)	12.80 (1.97)	6.68 (3.78)	4.63 (5.45)	3.91 (6.46)	3.80 (6.64)	3.62 (6.97)
FMSLD	10	(50,20,62,40)	108.53 (1.00)	54.08 (2.01)	26.98 (4.02)	19.79 (5.48)	14.22 (7.63)	14.41 (7.53)	13.83 (7.85)
FMSLD	11	(55,22,68,44)	480.84 (1.00)	237.83 (2.02)	125.13 (3.84)	88.03 (5.46)	62.33 (7.71)	59.06 (8.14)	57.88 (8.31)
FMSAD	1	(7,3,20,4)	0.08 (1.00)	0.14 (0.61)					
FMSAD	2	(14,6,40,8)	0.20 (1.00)	0.16 (1.22)	0.52 (0.37)	0.50 (0.39)			
FMSAD	3	(21,9,60,12)	0.56 (1.00)	0.38 (1.48)	0.31 (1.81)	0.27 (2.07)	1.51 (0.37)		
FMSAD	4	(28,12,80,16)	1.48 (1.00)	0.87 (1.69)	0.63 (2.35)	0.58 (2.53)	0.52 (2.84)	1.29 (1.14)	1.18 (1.25)
FMSAD	5	(35,15,100,20)	5.13 (1.00)	2.73 (1.88)	1.94 (2.64)	2.03 (2.52)	1.21 (4.24)	1.37 (3.74)	1.78 (2.88)
FMSAD	6	(42,18,120,24)	26.58 (1.00)	13.79 (1.93)	9.29 (2.86)	9.50 (2.80)	5.83 (4.56)	5.15 (5.16)	7.08 (3.75)
FMSAD	7	(49,21,140,28)	319.73 (1.00)	159.15 (2.01)	83.72 (3.82)	125.45 (2.55)	43.42 (7.36)	40.78 (7.84)	65.62 (4.87)
FMSAD	8	(56,24,160,32)	7815.53 (1.00)	3873.56 (2.02)	1791.60 (4.36)	3130.70 (2.50)	955.19 (8.18)	901.96 (8.67)	1535.25 (5.09)

Table 5.5: Execution of the parallel implementation for the considered families of S^4PR nets

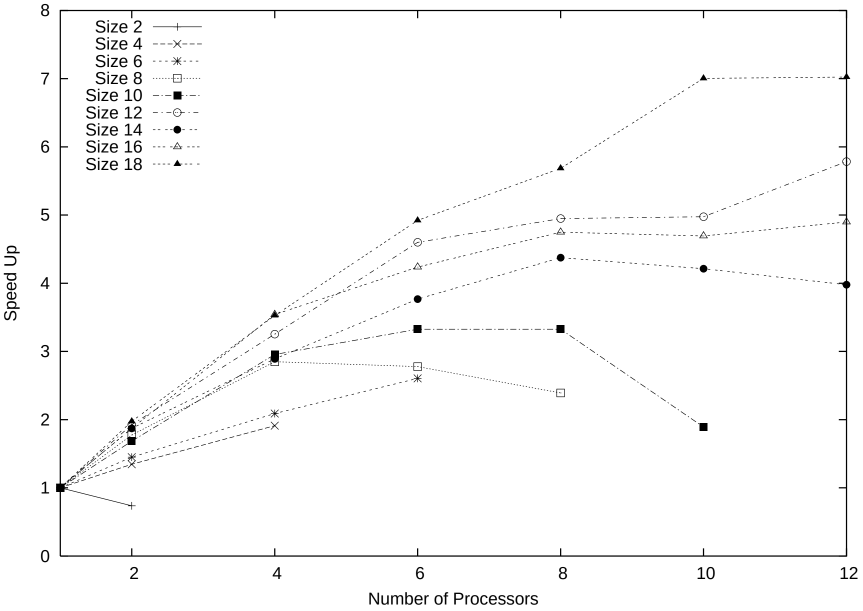


Figure 5.6: Speed–up for the Philosophers family

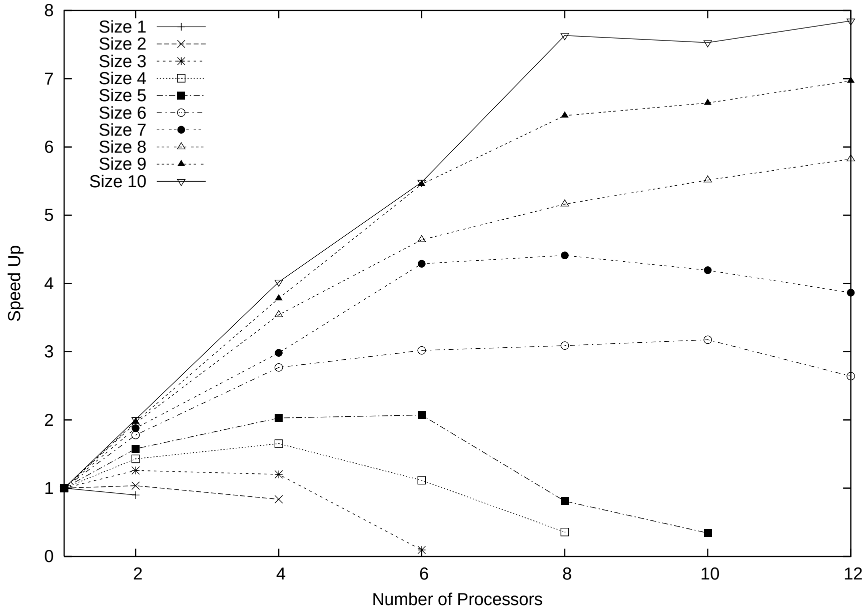


Figure 5.7: Speed–up for the FMSLD family

Figure 5.7 shows the growing of the relative speed–up for the FMSLD family. Finally, Figure 5.8 shows the behaviour of the method for the FMSAD family.

Let us concentrate on some irregularities appearing in the behaviour of the program, and let us explain our interpretation.

For example, in the case of the Philosophers family of size 18 there is only small improvement when 8 processors are used in comparison with the case of 6 processors (see Figure 5.6).

Table 5.6 shows how resources would be distributed among 6 processors, and Table 5.7 shows the distribution of resources among 8 processors. It is clear that when 8 processors are used processors $P1$ and $P2$ have to do some more work than the rest (aproximately as many as in the case of 6 processors). Therefore, little improvement can be obtained. Due to the regular form of the philosophers model, in this case the reason is clear, and the behaviour is due to a problem of local–balance.

A second type of irregular behaviour can be seen in the chart in Figure 5.8, corresponding to the solutions for the $FMSAD$ family. If we concentrate on the speed–ups for the problem sizes 5, 6, 7, and 8 we can notice that: first, it seems that

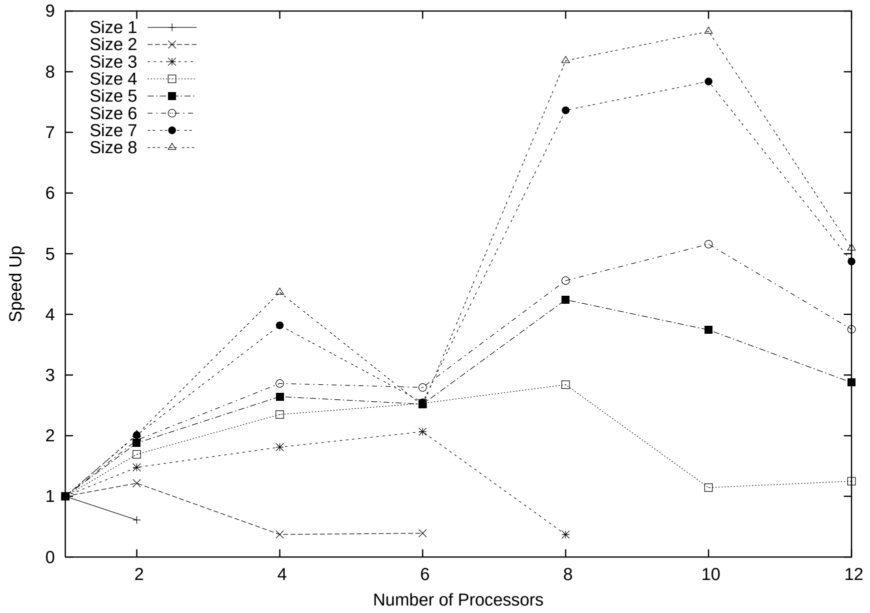


Figure 5.8: Speed–up for the FMSAD family

P1	P2	P3	P4	P5	P6
R1 R7 R13	R2 R8 R14	R3 R9 R15	R4 R10 R16	R5 R11 R17	R6 R12 R18

Table 5.6: Distribution of resources among six processors for the Phil–18 problem

P1	P2	P3	P4	P5	P6	P7	P8
R1 R9 R17	R2 R10 R18	R3 R11	R4 R12	R5 R13	R6 R14	R7 R15	R8 R16

Table 5.7: Distribution of resources among eigth processors for the Phil–18 problem

P1	P2	P3	P4
R1_1(e)	**R1_2**(i)	R1_3(e)	R2_1(e)
R2_2(i)	R2_3(e)	R3_1(e)	**R3_2**(i)
R3_3(e)	R4_1(e)	**R4_2**(i)	R4_3(e)
R5_1(e)	**R5_2**(i)	R5_3(e)	R6_1(e)
R6_2(i)	R6_3(e)	R7_1(e)	**R7_2**(i)
R7_3(e)			

Table 5.8: Distribution of resources among four processors for the FMSAD–7 problem

P1	P2	P3	P4	P5	P6
R1_1(e)	**R1_2**(i)	R1_3(e)	R2_1(e)	**R2_2**(i)	R2_3(e)
R3_1(e)	**R3_2**(i)	R3_3(e)	R4_1(e)	**R4_2**(i)	R4_3(e)
R5_1(e)	**R5_2**(i)	R5_3(e)	R6_1(e)	**R6_2**(i)	R6_3(e)
R7_1(e)	**R7_2**(i)	R7_3(e)			

Table 5.9: Distribution of resources among six processors for the FMSAD–7 problem

P1	P2	P3	P4
R1_1(e)	R2_3(e)	**R4_2**(i)	R6_1(e)
R1_2(i)	R3_1(e)	R4_3(e)	**R6_2**(i)
R1_3(e)	**R3_2**(i)	R5_1(e)	R6_3(e)
R2_1(e)	R3_3(e)	**R5_2**(i)	R7_1(e)
R2_2(i)	R4_1(e)	R5_3(e)	**R7_2**(i)
			R7_3(e)

Table 5.10: A different distribution of resources among four processors for the FMSAD–8 problem

P1	P2	P3	P4	P5	P6
R1_1(e)	R2_1(e)	**R3_2**(i)	**R4_2**(i)	R5_3(e)	R6_3(e)
R1_2(i)	**R2_2**(i)	R3_3(e)	R4_3(e)	R6_1(e)	R7_1(e)
R1_3(e)	R2_3(e)	R4_1(e)	R5_1(e)	**R6_2**(i)	**R7_2**(i)
	R3_1(e)		**R5_2**(i)		R7_3(e)

Table 5.11: A different distribution of resources among six processors for the FMSAD–8 problem

the speed–up grows until 10 processors are used; second, there is a very irregular behaviour when 6 (in fact, with any number of processors multiple of 3) processors are used: the speed–up decreases with respect to the use of 4 processors, and then it grows whith 8 processors. It clearly shows that this is a spurious behaviour.

Let us try to give an explanation of this behavior. In this family of problems two 'types' of resources can be identified (as we can easily see in Figure 5.3): the resources used at the beginning (or at the end) of the processing ($R1\_1$, $R1\_N$, $R2\_1$, and $R2\_N$), that can be named 'external resources', and the rest, that can be named 'internal resources'. The computation of place–siphons corresponding to internal resources seems to be more expensive than in the case of external resources. Tables 5.8 and 5.9 show the distribution of internal and external resources for the FMSAD–8 problem among processors when using 4 and 6 processors, respectively (boldface items correspond to internal resources). There we can see that in the case of using 6 processors, all the internal resources have to be processed by two processors (P2 and P5), which is a clear situation where the load balance is not adequate. Notice that in the case of 4 processors the distribution of hard resources among processors is balanced in a more adequate way.

To confirm the hypothesis that the reason for the observed anomalies is the bad load balance, some new experiments have been done, with the resource distributions shown in Tables 5.10 and 5.11.

Figure 5.9 compares the speed–up results for the FMSAD–7 problem obtained with the old and the new distributions. Now, the speed–up is also increased with 6 processors (and, for the same reason, with 12 processors). The same can be said for the FMSAD–8 problems, as shown in Figure 5.10

We can also see the comparison in Table 5.12. There we can see the results for the original experiment and the new one. The rows are grouped in pairs: the first row shows the results for the original experiment (as can be found in Table 5.5), and the second one the results for the new distribution of resources.

5.6 Conclusions

Some deadlock prevention control policies need the set of minimal siphons to be computed. It is well known that this is a very hard task because the number of such components can be very big (even exponential in some cases). However, the special syntactic characteristics of the Petri net models corresponding to RAS and, specifically, to S^4PR systems, made us to wonder whether specific adaptations of methods for siphon computation could be possible. This chapter is devoted to the

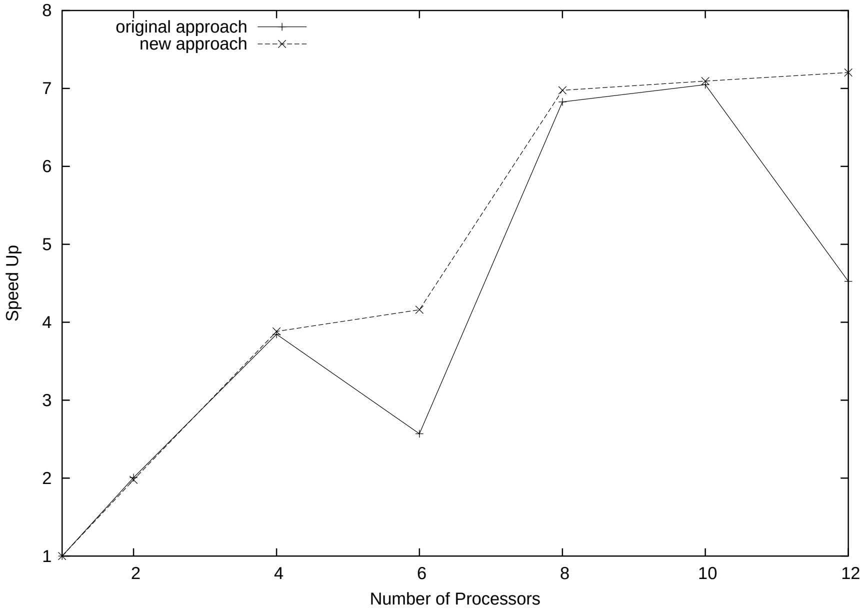


Figure 5.9: Speed–up comparison for the FMSAD of size 6

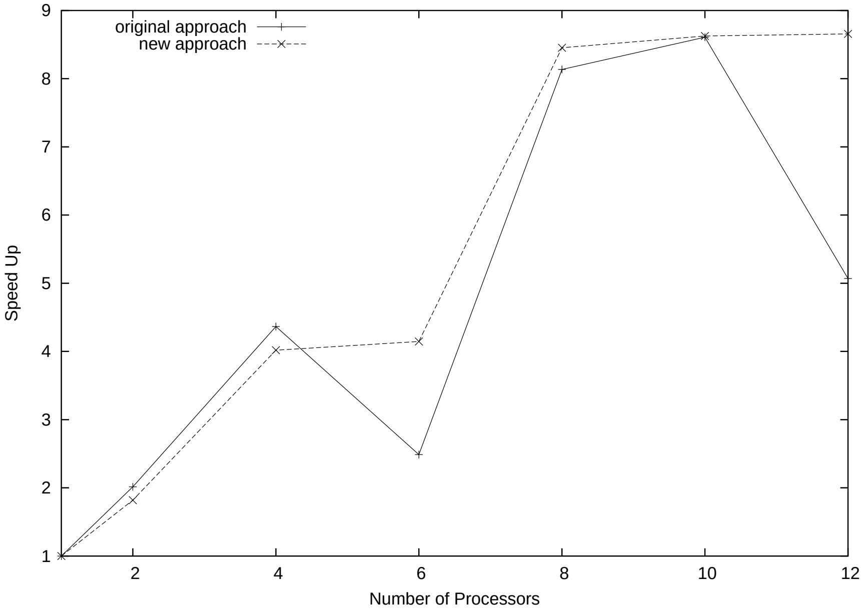


Figure 5.10: Speed–up comparison for the FMSAD of size 8

	Size		1	2	4	6	8	10	12
FMSAD	7	(49,21,140,28)	320.75 (1.00)	159.81 (2.01)	83.41 (3.85)	**124.84 (2.57)**	46.98 (6.83)	**45.51 (7.05)**	70.89 (4.52)
FMSAD	7	(49,21,140,28)	319.73 (1.00)	161.62 (1.98)	82.35 (3.88)	**76.84 (4.16)**	45.83 (6.98)	45.07 (7.09)	44.38 (7.20)
FMSAD	8	(56,24,160,32)	7813.57 (1.00)	3877.28 (2.02)	1789.67 (4.37)	**3141.11 (2.49)**	960.35 (8.14)	**907.68 (8.61)**	1541.26 (5.07)
FMSAD	8	(56,24,160,32)	7847.67 (1.00)	4311.81 (1.82)	1952.95 (4.02)	**1893.02 (4.15)**	928.31 (8.45)	909.82 (8.63)	906.46 (8.66)

Table 5.12: A comparison with a different distribution of resources for the first family

study of such question.

For that, a first analysis of the published methods for the computation of siphons is presented. The aim of this analysis was to determine which methods could be adapted to concentrate on the minimal siphons used for the control of S^4PR systems, trying to take advantage of his syntactical structure, and, in particular, of the fact that the siphons related to deadlock problems must contain at least one resource place. As a conclusion of this first analysis and also of some empirical experiments a the method proposed in [BM94] has been selected.

As a second step, and taking advantage of both the structure of S^4PR nets and the way the siphons are computed in the selected method, it has been improved implementing a parallel solution.

In order to measure the improvements some experiments have been carried out, comparing the time required for the computations of siphons in families of Petri nets.

As a result of such experiments we can conclude that the proposed parallel algorithm is of great interest since the speed–up charts show that the use of a set of processors (even in the case of them being connected using a network of computers instead of a multi-processor) gives very good results.

Another conclusion is that, as it was predictable, the Petri net structure is very important when an adequate load balance is needed. However, we can provide little insight of how resources should be distributed among the available processors in order to obtain an adequate distribution of computing time among them.

Chapter 6

Conclusions

In this work we have addressed the analysis, prevention and evitation of deadlock problems in sequential resource allocation systems, with the domain of flexible manufacturing systems as an application case.

In the first part we have proposed a new classes of nets, adequate to model a wide variety of flexible manufacturing systems. For this class, some structural properties have been established providing a way to show that the class is adequate for dealing with the kind of systems considered, and that there exist adequate ways to identify deadlock problems in it. In particular, a characterization of deadlock problems has been presented, together with some equivalent results more convenient to deal with the deadlock prevention problem. Moreover, it has also been shown that the characterization is not suitable for more general classes of systems.

In order to apply these results, the characterization has been adapted to the framework of the linear algebraic view inherent to the proposed Petri net model. For this, the main following results have been needed: adaptation of existing linear characterizations of the siphon property to our problem, and an adaptation of the proposed characterization of deadlocked states to a linear set of inequalities adequate to solve the problem. These two results provide a new way to compute the problematic states, by means of the solution of an integer linear programming problem. The method proposed to control the system is based on the addition of new preconditions implemented as virtual resources. This has two main advantages: the resulting model can be analyzed with the same tools as the original one (technical tools and background provided by the study of properties previously presented); we can apply a more detailed control, trying to prevent only bad states.

From the avoidance point of view, the proposed methods allow to deal with a wider class of systems. The main properties have also been studied, showing similar results to the previous one, plus the advantage obtained by the elimination of some restrictions. No characterization of the deadlock problem has been provided, and a framework based on the specialization of the Banker's algorithm taking advantage of the "a–priori" knowledge of the whole process structure. This framework is useful to better understand the approach. In this sense, several solutions have been proposed based on it. One of them has been further reformulated in order to obtain an efficient on–line solution.

Finally, some results about the computation of siphons have been presented. Since the proposed methods in the previous chapters include an adequate way of computing these structural components, these results have been not used there. However, we feel that these results serve as a suitable approximation that could be easily extended to more general classes of systems, obtaining further improvements. The main contributions are the selection of a method suitable for the kind of problems proposed, and its parallelization, together with the reduction of the computing time provided by the ability of the method to start searching the siphons at adequate places.

Bibliography

[AE98] I. B. Abdallah and H. A. ElMarghy. Deadlock prevention and avoidance in FMS: A Petri net based approach. *International Journal of Advanced Manufacturing Techology*, (14):704–715, 1998.

[Akl97] S.G. Akl. *Parallel Computation: Models and Methods*. Prentice-Hall, 1997.

[AT85] H. Alaiwan and J.M. Toudic. Recherche des semi-flots, des verroux et des trappes dans les rseaux de Petri. *Technique et Science Informatiques*, pages 103–112, 1985. In French.

[BA95] K. Barkaoui and I. Ben Abdallah. A deadlock prevention method for a class of FMS. In *Proc. 1995 IEEE Int. Conf. on Systems, Man and Cybernetics.*, pages 4119–4124, Vancouver, Canada, October 1995. IEEE Systems, Man and Cybernetics Society.

[BA96] K. Barkaoui and I. B. Abdallah. Analysis of a resource allocation problem in FMS using structure theory of Petri nets. In M. Silva, R. Valette, and K. Takahashi, editors, *Proc. of the workshop: Manufacuring and Petri Nets*, pages 62–76, Osaka, Japan, June 1996.

[BCG98] Francesco Basile, Pasquale Chiacchio, and Allessandro Giua. Supervisory control of Petri nets based on suboptimal monitor places. In *Proceedings of the IEE International Workshop on Discrete Event Systems*, pages 85–87, Cagliari, Italy, aug 1998. IEE.

[BCZ97] K. Barkaoui, A. Chaoui, and B. Zouari. Supervisory control of discrete event systems based on structure of Petri nets. In *SMC'97 Conf. Proc.*, pages 3750–3755, Orlando, Florida, USA, October 1997. IEEE.

[BDR$^+$84] J. Browne, D. Dubois, K. Rahtmill, P. Sethi, and K.E. Stecke. Classification of flexible manufacturing systems. *The FMS Magazine*, pages 114–117, apr 1984.

[BK90] Z.A. Banaszak and B.H. Krogh. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Trans. on Robotics and Automation*, 6(6):724–734, December 1990.

[BL89] K. Barkaoui and B. Lemaire. An effective characterization of minimal deadlocks and traps in Petri nets based on graph theory. In *Proceedings of the 10th. International Conference on Theory and Applications of Petri nets*, pages 1–22, Bonn, 1989.

[BLP96] G.C. Barroso, A.M.N. Lima, and A. Perkusich. Supervision of discrete event system using Petri nets and supervisory control theory. In M. Silva, R. Valette, and K. Takahashi, editors, *Proc. of the workshop: Manufacuring and Petri Nets*, pages 77–96, Osaka, Japan, June 1996.

[BM92] K. Barkaoui and M. Minoux. A polynomial-time graph algorithm to decide liveness of some basic classes of bounded Petri nets. In K. Jensen, editor, *Advances in Petri Nets 1992*, volume 616 of *Lecture Notes in Computer Science*, pages 62–75. Springer Verlag, Berlin, 1992.

[BM94] E. R. Boer and Tadao Murata. Generating basis siphons and traps of petri nets using the sign incidence matrix. *IEEE Trans. on Circuits and Systems, I – Fundamental Theory and Applications*, 41(4):266–271, 1994.

[BPP96] K. Barkaoui and J.F. Pradat-Peyre. On liveness and controlled siphons in Petri nets. In *Proc. of the 1996 Int. Conf. on Aplications and Theory of Petri Nets*. Springer Verlag, June 1996.

[Bra83] G. W. Brams. *Rèseaux de Petri. Theorie et Pratique (2 tomes)*. Masson, Paris, 1983.

[Bry92] R.E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[Can98] Marco Cantamessa. The manufacturing system as a complex artifac. *Robotics and Computer–Integrated Manufacturing*, (14):403–141, 1998.

[CES71] E.G. Coffman, M.J. Elphick, and A. Shoshani. System deadlocks. *ACM Computer Surveys*, 3(2):67–78, 1971.

[CG96] Darren D. Coffer and Vijay K. Garg. Supervisory control of real–time discrete–event systems using lattice theory. *IEEE Transactions on Automatic Control*, 41(2):199–209, Feb 1996.

[CGL94] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16:1512–1542, 1994.

[Che00] Haoxun Chen. Control synthesis of Petri nets based on s–decreases. *Discrete Event Dynamic Systems: Theory and Applications*, 10(3):233–249, jul 2000.

[CKW95] Hyuenbo Cho, T.K. Kumaran, and Richard A. Wysk. Graph-theoretic deadlock detection and resolution for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation*, 11(3):413–421, 1995.

[CLR90] T.H. Cormen, C.E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The M.I.T. Press, 1990.

[Com72] F. Commoner. Deadlocks in Petri nets. *Applied Data Research*, 1972.

[CS89] J.M. Colom and M. Silva. Improving the linearly based characterization of P/T nets. In *Proceedings of the 10th International Conference on Application and Theory of Petri Nets*, pages 52–73, Bonn, Germany, June 1989.

[CS91] J.M. Colom and M. Silva. Improving the linearly based characterization of P/T nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 113–145. Springer Verlag, Berlin, 1991.

[CX97] Feng Chu and Xiaolan Xie. Deadlock analysis of petri nets using siphons and mathematical programming. *IEEE Transactions on*

Robotics and Automation, 13(6):793–804, dec 1997. Descubrimiento de los cerrojos para el análisis de S3PR 'extendidos'. Trabajo en la línea de Teruel,Silva,Colom. Se caracterizan los cerrojos usando un problema de programación mixta. Cita a [ECM95].

[Dij65] E.W. Dijkstra. *Co-operating Sequential Processes*. Programming Languages. Academic Press, f. genyus edition, 1965.

[EC91] J. Ezpeleta and J.M. Couvreur. A new technique for finding a generating family of siphons, traps and st-components. application to colored Petri nets. In *Proceedings of the 12th. International Conference on Application and Theory of Petri Nets*, pages 145–164, Aarhus (Denmark), June 1991.

[ECM95] J. Ezpeleta, J.M. Colom, and J. Martínez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation*, 11(2):173–184, April 1995.

[ECS93] J. Ezpeleta, J.M. Couvreur, and M. Silva. A new technique for finding a generating family of siphons, traps and st-components. application to colored Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets 1993*, volume 674 of *Lecture Notes on Computer Science*, pages 126–147. Springer-Verlag, Aarhus (Denmark), 1993.

[EGVC98a] J. Ezpeleta, F. García-Vallés, and J.M. Colom. A class of well structured petri nets for flexible manufacturing systems. pages 64–83, jun 1998.

[EGVC98b] 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, *Application and Theory of Petri Nets 1998*, volume 1420 of *Lecture Notes on Computer Science*, pages 64–83. Springer-Verlag, 1998.

[EH93] J. Ezpeleta and S. Haddad. A distributed algorithm for resource management. In *Proceedings of the Int. Conference on Decentralized and Distributed Systems (ICDDS'93)*, Palma de Mallorca (Spain), September 1993.

[ES92] J. Esparza and M. Silva. A polynomial-time algorithm to decide liveness of bounded free choice nets. *Theoretical Computer Sciences*, 102:185–205, 1992.

[FMMT97] M.P. Fanti, B. Maione, S. Mascolo, and B. Turchiano. Event–based feedback control for deadlock avoidance in flexible production systems. *IEEE Trans. on Robotics and Automation*, 13(3):347–363, June 1997.

[FMT00] M.P. Fanti, B. Maione, and B. Turchiano. Comparing digraph and Petri net approaches to deadlock avoidance in FMS. *IEEE Transactions on Systems, Man, and Cybernetics–Part B: Cybernetics*, 30(5):783–798, oct 2000.

[FNTS94] L. Ferrarini, M. Narduzzi, and M. Tassan-Solet. A new approach to modular liveness analysis conceived for large logic controllers' design. *IEEE Trans. on Robotics and Automation*, 10(2):169–184, April 1994.

[For94] Message Passing Interface Forum. MPI: A message–passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4), 1994.

[FTM99] M.P. Fanti, B. Turchiano, and G. Maione. Design and implementation of supervisory avoiding deadlocks i flexible assembly sytems. pages 667–672, 1999.

[GDS92] A. Giua, F. DiCesare, and M. Silva. Generalized mutual exclusion constraints for nets with uncontrollable transitions. In *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*, pages 974–799, Chicago,USA, Oct 1992.

[Ge03] C. Girault and R. Valk (eds.). *Petri Nets for Systems Engineering. A Guide to Modeling, Verification, and Applications*. Springer Verlag, 2003.

[Giu96] Alessandro Giua. Petri net techniques for supervisory control of discrete event systems. In M. Silva, R. Valette, and K. Takahashi, editors, *Proc. of the workshop: Manufacuring and Petri Nets*, pages 1–30, Osaka, Japan, June 1996.

[GK90] G. Georgakopoulos and D. Kavadias. The banker's problem with precedences. *Information and Computation*, 84:1–12, 1990.

[GL96] William D. Gropp and Ewing Lusk. *User's Guide for* `mpich`*, a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.

[GL01] N.Z. Gebraeel and M.A. Lawley. Deadlock detection, prevention, and avoidance for automated tool sharing systems. *IEEE Transactions on Robotics and Automation*, 17(3):342–356, 2001.

[Gol78] E. Mark Gold. Deadlock prediction: Easy and difficult cases. *SIAM J. Comput.*, 7(3):320–336, Aug 1978.

[GS02] Alessandro Giua and Carla Seatzu. Liveness enforcing supervisors for railway networks using ES^2PR Petri nets. In *Proceedings of the 6th International Workshop on Discrete Event Systems*, pages 55–66, Zaragoza, Spain, oct 2002. IEEE.

[GVTCE00] F. García-Vallés, F. Tricas, J.M. Colom, and J. Ezpeleta. Structuraly safe net systems. In R. Boel and G. Stremersch, editors, *Discrete Event Systems: Analysis and Control. Proc. of the Workshop On Discrete Event Systems 2000*, pages 441–448, Ghent, Belgium, Aug 2000. Kluwer Academic Publishers.

[Hab69] A. N. Habermann. Prevention of system deadlocks. *Communications of the ACM*, 12(7):373–377, 385, 1969.

[Hab75] A.N. Habermann. A new approach to avoidance of system deadlocks. *R.A.I.R.O. Informatique Theorique*, B-3:19–28, September 1975.

[HC92] F. Hsieh and S. Chang. Deadlock avoidance controller synthesis for flexible manufacturing systems. In *Proc. of Rensselaer's 3rd Int. Confernce on Computer Integrated Manufacturing*, pages 252–261, Troy (New York), May 20-22 1992. Rensselaer Polytechnic Institute.

[HC94] F. Hsieh and S. Chang. Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation*, 10(2):196–209, April 1994.

[HJW02] YiShen Huang, MuDer Jeng, and YuanLin Wen. Analysis of a siphon–based deadlock prevention policy for flexible manufacturing systems. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pages 2327–2332, Washington DC, May 2002.

[HKG97] L.E. Holloway, B.H. Krogh, and A. Giua. A survey of petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems: Theory and Applications*, (7):151–190, 1997.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HSBM96] K. Hasegawa, M. Sugisawa, Z. A. Banaszak, and L. Qun Ma. Graphical analysis and synthesis of deadlock avoidance in flexible manufacturing systems. In M. Silva, R. Valette, and K. Takahashi, editors, *Proc. of the workshop: Manufacturing and Petri Nets*, pages 161–176, Osaka, Japan, June 1996.

[IM80] Sreekaanth S. Isloor and T. Anthony Marsland. The deadlock problem: An overview. *Computer*, 13(9):58–78, sep 1980.

[IMA02] Marian V. Iordache, John O. Moody, and Panos Antsaklis. Synthesis of deadlock prevention supervisors using Petri nets. *IEEE Transactions on Robotics and Automation*, 18(1):59–68, feb 2002.

[IV88] K. Inan and P. Varaiya. Finitely recursive process models for discrete event systems. *IEEE Trans. Automatic Control*, 33(7):626–639, 1988.

[JD95] M.D. Jeng and F. DiCesare. Synthesis using resource control nets for modeling shared-resource systems. *IEEE Trans. on Robotics and Automation*, 11(3):317–327, June 1995.

[Jen96] M.D. Jeng. A Petri net synthesis theory for modeling flexible manufacturing systems. *IEEE Trans. on Systems, Man and Cybernetics-Part B: Cybernetics*, 27(2):169–183, April 1996.

[Joh75] D. B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM J. Comput.*, 4(1):77–84, 1975.

[JPH99] Muder Jeng, Maoyu Peng, and Yisheng Huang. An algorithm for calculating minimal siphons and traps of petri nets. *Int. Journal of Intelligent Control and Systems*, 3(3):263–275, 1999.

[JXH00] MuDer Jeng, Xiaolan Xie, and YiSheng Huang. Manufacturing modeling using process nets with resources. In *Proc. of the 2000 IEEE*

International Conference on Robotics and Automation, pages 2185–2190, San Francisco, USA, apr 2000.

[KB92] Peter Kemper and Falko Bause. An efficient polynomial-time algorithm to decide liveness and boundedness of free choice nets. In Jensen, K., editor, *Lecture Notes in Computer Science; 13th International Conference on Application and Theory of Petri Nets 1992, Sheffield, UK*, volume 616, pages 263–278. Springer-Verlag, June 1992.

[KTJK97] C.W. Kim, Tanchoco, J.M.A., and P.H. Koo. Deadlock prevention in manufacturing systems with AGV systems: Banker's algorithm approach. *Journal of Manufacturing Science and Engineering*, (119):849–854, 1997.

[Kuc78] David J. Kuck. *The structure of computers and computations*, volume 1. iJahn Wiley and Sons, 1978.

[Lan99] Sheau-Dong Lang. An extended banker's algorithm for deadlock avoidance. *IEEE Transactions on Software Engineering*, 25(3):428–432, May/June 1999.

[Lau87] Kurt Lautenbach. Linear algebraic calculation of deadlocks and traps. In Voss, Genrich, and Rozemberg, editors, *Concurrency and Nets*, pages 315–336. Springer Verlag, 1987.

[Law99] Mark A. Lawley. Deadlock avoidance for production systems with flexible routing. *IEEE Transactions on Robotics and Automation*, 15(3):1–13, jun 1999.

[Law00] M. Lawley. Integrating flexible routing and algebraic deadlock avoidance policies in automated manufacturing systems. *International Journal of Production Research*, 38(13):2931–2950, 2000.

[LGB$^+$98] F.L. Lewis, A. Gurel, S Bogdan, A Doğanalp, and OC Pastravanu. Analysis of deadlock and circular waits using a matrix model for flexible manufacturing systems. *Automatica*, 34(9):1083–1100, 1998.

[LMB97] S.C. Lauzon, J.K. Mills, and B. Benhabib. An implementation methodology for the supervisory control of flexible manufacturing workcells. *Journal of Manufacturing Systems*, 16(2):91–101, 1997.

[LR96] Kurt Lautenbach and Hanno Ridder. The linear algebra of deadlock avoidance – a PN approach. Technical report, Institute for Computer Science. U. Koblenz, 1996.

[LRF97] M. Lawley, S. Reveliotis, and P. Ferreira. FMS structural control and the neghborhood policy: Part 1 correctness and scalability. *IIE Transactions*, (29):877–887, 1997.

[LRF98a] M. Lawley, S. Reveliotis, and P. Ferreira. The application and evaluation of Banker's algorithm for deadlock–free buffer allocation in flexible manufacturing systems. *Int. Journal of Flexible Manufacturing Systems*, (10):73–100, 1998.

[LRF98b] M. Lawley, S. Reveliotis, and P. Ferreira. A correct and scalable deadlock avoidance policy for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation*, 14(5):796–809, October 1998.

[LT79] K. Lautenbach and P.S. Thiagarajan. Analysis of a resource allocation problem using Petri nets. In J.C. Syre, editor, *1st. European Conf. on Parallel and Distributed Systems*, pages 1–17, 1979.

[MB90] M. Minoux and K. Barkaoui. Deadlocks and traps in petri nets as horn-satisfiability solutions and some related polynomially solvable problems. *Discrete Mathematics*, 29:195–210, 1990. NewsletterInfo: 39.

[MBSD99] Mbi Makungu, Michel Barbeau, and Richard ST-Denis. Synthesis of controllers of processes modeled as colored petri nets. *Discrete Event Systems: Theory and Applications*, 9(2):147–169, may 1999.

[MMS01] Berna L. Massingill, Timothy G. Mattson, and Beverly A. Sanders. More patterns for parallel application programs. In *Proceedings of the Eighth Pattern Languages of Programs Workshop*, 2001.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[MV00] F. Magnino and P. Valigi. A petri net approach to deadlock analysis for classes of kanban systems. In *IEEE Int. Conf. on Robotics and Automation*, pages 2877–2882, San Francisco, USA, Apr 2000. IEEE Computer Society.

[NV86] Y. Narahari and N. Viswanadham. On the invariants of colored Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets 1986*, volume 222 of *Lecture Notes in Computer Science*, pages 330–345. Springer-Verlag, Berlin, 1986.

[OH00] Yeong-Chang Ou and Jwu-Sheng Hu. A modified method for supervisor specification and synthesis of a class of discrete event systems. *Asian Journal of Control*, 2(4):263–273, dec 2000.

[Ost89] J.S. Ostroff. *Temporal Logic for Real-Time Systems*. Advanced Software Development Series. Research Studies. Press Limited (Jon Wiley and Sons), England, 1989.

[Pet81] J. L. Peterson. *Petri Net Theory and The Modeling of Systems*. Prentice-Hall, 1981.

[PR00a] J. Park and S.A. Reveliotis. Algebraic synthesis of efficient deadlock avoidance policies for sequential resource allocation systems. *IEEE Transactions on Robotics and Automation*, 16(2):190–195, apr 2000.

[PR00b] Jonghun Park and Spyros A. Reveliotis. Enhancing the flexibility of algebraic deadlock avoidance policies through petri net structural analysis. In *Proc. of the 2000 IEEE Int. Conf. on Robotics and Automation*, pages 3371–3376, San Francisco, USA, April 2000.

[PR01] Jonghun Park and Spyros Reveliotis. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. *IEEE Transactions on Automatic Control*, 46(10):1572–1583, oct 2001.

[PS85] J. L. Peterson and A. Silberschatz. *Operating System Concepts, 2nd ed*. Addison-Wesley (Reading MA), 1985.

[QJ99] Robin G. Qiu and Sanjay B. Joshi. A structured adaptive supervisory control methodology for modeling the control of a discrete event manufacturing system. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS PART A: SYSTEMS AND HUMANS*, 29(6):573–586, nov 1999.

[Rev98] Spiridon A. Reveliotis. Variations on Banker's algorithm for resource allocation systems. In *1998 Japan-USA Symposium on Flexible Automation*, pages 1185–1192, Otsu, Japan, Jul 1998.

[Rev99] Spyros A. Reveliotis. Acomodating FMS operational contingencies through routing flexibility. *IEEE Trans. on Robotics and Automation*, 15(1):3–19, Feb 1999.

[Rev00] Spyros A. Reveliotis. Conflict resolution in AGV systems. *IEE Transactions*, 32(7):647–659, 2000.

[RF96] S. Reveliotis and P.M. Ferreira. Deadlock avoidance policies for automated manufacturing cells. *IEEE Transactions on Robotics and Automation.*, 12(6):846–857, dec 1996.

[RJ96] Sanjay E. Ramaswamy and Sanjay B. Joshi. Deadlock–free schedules for automated manufacturing workstations. *IEEE Transactions on Robotics and Automation*, 12(3):391–400, jun 1996.

[RR92a] E. Roszkowska and R.Wojcik. Problems of process flow feasibility in FAS. In K. Leiviska, editor, *IFAC CIM in Process and manufacturing Industries*, pages 115–120, Espoo, Finland, 1992. Oxford: pergamon press.

[RR92b] Z.Banaszak R.Wojcik and E. Roszkowska. Automation of self–recovery resource allocation procedures synthesis in FMS. In K. Leiviska, editor, *IFAC CIM in Process and manufacturing Industries*, pages 127–132, Espoo, Finland, 1992. Oxford: pergamon press.

[RW87] P.J.G. Ramadge and W.M.Wonham. Supervisory control of a class of discrete event systems. *SIAM Journal Control Optim.*, 25(1):206–230, jan 1987.

[RW89] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings IEEE*, 77(1):81–98, 1989.

[RW92] K. Rudie and W. M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, nov 1992.

[RYJ91] R.A.Wysk, N.S. Yang, and S. Joshi. Detection of deadlocks in flexible manufacturing cells. *IEEE Trans. on Robotics and Automation*, 7(6):853–859, December 1991.

[Sil85] M. Silva. *Las Redes de Petri en la Automática y la Informática*. Editorial AC, Madrid, 1985.

[Sin89a] M. Singhal. Deadlock detection in distributed systems. 22(11):37–48, November 1989.

[Sin89b] M. Singhal. Deadlock detection in distributed systems. *IEEE Computer*, pages 37–48, November 1989.

[SPG91] A. Silberschatz, J. Peterson, and P. Galvin. *Operating System Concepts.* Addison-Wesley, 1991.

[SR95] Kathryn E. Stecke and Narayan Raman. FMS planning decisions, operating flexibilities, and system performance. *IEEE Transactions on Enginering Management*, 42(1):82–90, feb 1995.

[Sre00] Ramavarapu S. Sreenivas. On partially controlled free choice Petri nets. In R. Boela and G. Stremersch, editors, *Discrete Events Systems: Analysis and Control*, pages 159–168, Ghent, Belgium, Aug 2000. Kluwer Academic Publishers.

[ST96] M. Silva and E. Teruel. Petri nets for the design and operation of manufacturing systems. In *Proceedings of the 5th. Conference on Computer Integrated Manufacturing and Automation Control (CIMAT 96)*, pages 330–343. IEEE Computer Society Press, 1996.

[ST97] M. Silva and E. Teruel. Petri nets for the design and operation of manufacturing systems. *European Journal of Control*, (3):182–199, 1997.

[SV89] M. Silva and R. Valette. Petri nets and flexible manufacturing. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 424 of *Lecture Notes on Computer Science*, pages 374–417. Springer-Verlag, Berlin, 1989.

[Tan87] A.S. Tanenbaum. *Operating Systems: Desing and Implementation*. Prentice-Hall International Editions, 1987.

[TCE99] F. Tricas, J.M. Colom, and J. Ezpeleta. A solution to the problem of deadlocks in concurrent systems using Petri nets and integer linear programming. In G. Horton, D. Moller, and U. Rude, editors, *Proc. of* *the 11th European Simulation Symposium*, pages 542–546, Erlangen, Germany, October 1999. The society for Computer Simulation Int.

[TCE00] F. Tricas, J.M. Colom, and J. Ezpeleta. Some improvements to the Banker's algorithm based on the process structure. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 2853–2858, San Francisco, USA, April 2000. IEEE.

[TE99] F. Tricas and J. Ezpeleta. A Petri net solution to the problem of deadlocks in systems of processes with resources. In *Proc. of the 7th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, pages 1047–1056, Barcelona, October 1999.

[TGVCE98] F. Tricas, F. García-Vallés, J.M. Colom, and J. Ezpeleta. A structural approach to the problem of deadlock prevention in processes with resources. In *Proc. of the Int. Workshop on Discrete Event Systems(WODES'98)*, pages 273–278. IEE Control, IEE, August 1998.

[TGVCE00] F. Tricas, F. García-Vallés, J.M. Colom, and J. Ezpeleta. An iterative method for deadlock prevention in FMS. In R. Boel and G. Stremersch, editors, *Disscrete Event Systems: Analysis and Control. Proc. of the Workshop On Discrete Event Systems 2000*, pages 139–148, Ghent, Belgium, Aug 2000. Kluwer Academic Publishers.

[TM95] F. Tricas and J. Martínez. An extension of the liveness theory for concurrent sequential processes competing for shared resources. In *Proc. of the 1995 IEEE Int. Conf. on Systems, Man and Cybernetics.*, pages 4119–4124, Vancouver, Canada, October 1995.

[Tou81] J.M. Toudic. *Algorithmes d'analyse structurelle de réseaux de Petri*. PhD thesis, Université Pierre et Marie Curie (Paris VI), 1981.

[Val99] Fernando García Vallés. *Contributions to the Structural and Symbolic Analysis of Place/Transition Nets with Applications to Flexible Manufacturing Systems and Asynchronous Circuits*. PhD thesis, Zaragoza. España, Departamento de Ingeniería Eléctrica e Informática, Universidad de Zaragoza, 1999.

[VN92] N. Viswanadham and Y. Narahari. *Performance Modeling of Automated Manfacturig Systems*. Prentice-Hall, 1992.

[VNJ90] N. Viswanadham, Y. Narahari, and T.L. Johnson. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models. *IEEE Trans. on Robotics and Automation*, 6(6):713–723, December 1990.

[WYT98] Toshimasa Watanabe, Masahiro Yamauchi, and Shinji Tanimoto. Extracting siphons containing specified set of places in petri net. In *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'98), 11-14 October 1998, San Diego, USA*, pages 142–147, October 1998.

[XHC96] K. Xing, B. Hu, and H. Chen. Deadlock avoidance policy for Petri–net modeling of flexible manufacturing systems with shared resources. *IEEE Trans. on Automatic Control*, 41(2):289–295, February 1996.

[XHD99] Chen Zhou X. Hua Du. Message–oriented decomposition for supervisory control in manufacturing systems. *Robotics and Computer Integrated Manufacturing*, 15(6):441–452, dec 1999.

[XJ99] Xiaolan Xie and MuDer Jeng. ERCN–merged nets and their analysis using siphons. *IEEE Transactions on Robotics and Automation*, 15(4):692–703, aug 1999.

[YB00] Ali Yalcin and Thomas O. Boucher. Deadlock avoindance in flexible manufacturing systems using finite automata. *IEEE Transactions on Robotics and Automation*, 16(4):424–429, aug 2000.

[YW99a] M. Yamauchi and T. Watanabe. Algorithms for extracting minimal siphons containing specified places in a general Petri net. *IEICE Trans. Fundamentals*, E82-A(11):2566–2575, November 1999.

[YW99b] M. Yamauchi and T. Watanabe. Time complexity analysis of the minimal siphon extraction problem of Petri nets. *IEICE Trans. Fundamentals*, E82-A(11):2558–2565, November 1999.

[ZD93a] M. Zhou and F. DiCesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Academic Publishers, 1993.

[ZD93b] M. Zhou and F. DiCesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Academic Publishers, 1993.

Appendix A

Appendix: Petri nets

In this appendix we recall the basic definitions and notation about Petri nets. Interested readers can look for more information on the topic in [Pet81, Bra83, Sil85, Mur89].

A.1 Basic concepts on Petri nets

Petri nets: A Petri net (or Place/Transition net) is a 3-tuple $\mathcal{N} = \langle P, T, W \rangle$ where P and T are two non-empty disjoint sets whose elements are called *places* and *transitions*, respectively. In a generic way, elements belonging to $P \cup T$ are called *nodes*. $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ defines the *weighted flow relation*: if $W(x, y) > 0$, then we say that there is an arc from x to y, with weight or multiplicity $W(x, y)$. Ordinary nets are those where $W : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$.

A Petri net can be seen as a bipartite weighted directed graph in which the two kinds of nodes are places and transitions. There is a graphical representation for Petri nets where places are depicted as circles while transitions are depicted as bars or boxes.

Given a net $\mathcal{N} = \langle P, T, W \rangle$ and a node $x \in P \cup T$, ${}^\bullet x = \{y \in P \cup T \mid W(y, x) > 0\}$ is the *pre-set* of x, while $x^\bullet = \{y \in P \cup T \mid W(x, y) > 0\}$ is the *post-set* of x. This notation is extended to a set of nodes as follows: given $X \subseteq P \cup T$, ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$, $X^\bullet = \bigcup_{x \in X} x^\bullet$.

A Petri net is *self–loop free* when $W(x, y) \neq 0$ implies that $W(y, x) = 0$. The Pre–incidence matrix $\mathbf{Pre} : P \times T \rightarrow \mathbb{N}$ of $\mathcal{N}$ is $\mathbf{Pre}[p, t] = W(p, t)$. The Post–incidence matrix $\mathbf{Post} : P \times T \rightarrow \mathbb{N}$ of $\mathcal{N}$ is $\mathbf{Post}[p, t] = W(t, p)$. A self–loop Petri net $\mathcal{N} = \langle P, T, W \rangle$ can be alternatively represented as $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$

where $\mathbf{C}$ is the incidence matrix: a $P \times T$ indexed matrix such that $\mathbf{C}[p,t] = W(t,p) - W(p,t) = \mathbf{Post}[p,t] - \mathbf{Pre}[p,t]$. Being $\Pi \subseteq P$ and $\Theta \subseteq T$, $\mathbf{C}[\Pi, \Theta]$ denotes the submatrix of $\mathbf{C}$ corresponding to rows of places in Π, and to columns of transitions in Θ. This also will be used for matrices $\mathbf{Pre}$ and $\mathbf{Post}$.

A net $\mathcal{N}' = \langle P', T', W' \rangle$ is a subnet of $\mathcal{N} = \langle P, T, W \rangle$ if, and only if, $P' \subseteq P, T' \subseteq T$, and W' is the restriction of W to P' and T'. On the other hand, given a net, $\mathcal{N}$, and $P' \subseteq P, T' \subseteq T$, we can define the *subnet generated by P' and T'* (and denote it with $\mathcal{N}_{|(P',T')}$) as the subnet of $\mathcal{N}$ whose places are the ones in P', the transitions are the ones in T' and W' is the restriction of W to P' and T'. One of the subsets can be empty, and then the subnet will be defined, if the non empty subset of nodes is V, as the subnet generated by V and ${}^\bullet V \cup V^\bullet$.

A net is a *State Machine* (SM) if and only if is ordinary and for all $t \in T$, $|{}^\bullet t| = |t^\bullet| = 1$.

Multi–sets

Let $A \neq \emptyset$ be a set. A *multi–set* $\mathbf{m}$ over a A is a mapping $\mathbf{m} : A \to \mathbb{N}$, which associates to each element $a \in A$ a non-negative integer coefficient (or multiplicity) $\mathbf{m}(a)$. $\mathrm{Bag}(A)$ will denote the set of multi-sets over A. A multi–set will be denoted as the symbolic addition of its components: $\mathbf{m} = \sum_{a \in A} \mathbf{m}(a) \cdot a$. The addition of multi–set is defined as $\mathbf{m} + \mathbf{m}' = \sum_{a \in A} (\mathbf{m}(a) + \mathbf{m}'(a)) \cdot a$. On the other hand, $\mathbf{m} \geq \mathbf{m}'$ when for each $a \in A$, $\mathbf{m}(a) \geq \mathbf{m}'(a)$. For short, $\mathbf{0}$ will be used to denote the *empty multi–set*. Let $S \subseteq \mathrm{Bag}(A)$; $\mathbf{SUP}(S)$ denotes the multi–set $\mathbf{m} \in \mathrm{Bag}(A)$ such that for every $s \in S$, $\mathbf{m} \geq s$ and if $\mathbf{m}' \in \mathrm{Bag}(A)$ verifies the same property, then $\mathbf{m}' \geq \mathbf{m}$. $|\mathbf{m}|$ will denote $|\mathbf{m}| = \sum_{a \in A} \mathbf{m}(a)$.

Petri net behavior

A *marking* is a mapping $\mathbf{m} : P \to \mathbb{N}$; in general, markings are represented in vector form. Some times, multi–sets notation will be more convenient for markings: $\mathbf{m} = \sum_{p \in P} \mathbf{m}[p] \cdot p$. When talking about a set of places $S \subseteq P$, $\mathbf{m}[S] = \sum_{p \in S} \mathbf{m}[p]$. The pair $\langle \mathcal{N}, \mathbf{m_0} \rangle$, where $\mathcal{N}$ is a Petri net and $\mathbf{m_0}$ is an (initial) marking, is called a *marked Petri net* (or Place/Transition system). A transition $t \in T$ is *enabled* for a marking $\mathbf{m}$ if and only if $\forall p \in {}^\bullet t \, . \, \mathbf{m}[p] \geq W(p,t)$; this fact will be denoted as $\mathbf{m}\xrightarrow{t}$ (or $\mathbf{m}[t\rangle$). If t is enabled at $\mathbf{m}$, it can *occur*; when it occurs, this gives a new marking $\mathbf{m}' = \mathbf{m} + \mathbf{C}[P,t]$; this will be denoted as $\mathbf{m}\xrightarrow{t}\mathbf{m}'$ (or $\mathbf{m}[t\rangle\mathbf{m}'$), and we say that $\mathbf{m}$' is reached from $\mathbf{m}$ by the occurrence of t. A *firing sequence* (also *occurrence sequence*) from $\mathbf{m}$ is a sequence $\sigma = t_1 t_2 ... t_n$

so that $\mathbf{m}\xrightarrow{t_1}\mathbf{m}_1\xrightarrow{t_2}\mathbf{m}_2...\mathbf{m}_{n-1}\xrightarrow{t_n}\mathbf{m}'$ ($\mathbf{m}[t_1{>}\mathbf{m}_1[t_2{>}\mathbf{m}_2[\ldots{>}\mathbf{m}_{n-1}[t_n{>}\mathbf{m}'$). This is denoted as $\mathbf{m}\xrightarrow{\sigma}\mathbf{m}'$ (or $\mathbf{m}[\sigma{>}\mathbf{m}'$). The *firing count* vector of a sequence σ is $\boldsymbol{\sigma}[t] = \#(t,\sigma)$, where $\#(t,\sigma)$ denotes the number of occurrences of t in σ. Therefore, if $\mathbf{m}\xrightarrow{\sigma}\mathbf{m}'$, then $\mathbf{m}' = \mathbf{m}+\mathbf{C}\cdot\boldsymbol{\sigma}$. The set of sequences that are fireable from $\mathbf{m_0}$ is a language: $\mathrm{L}(\mathcal{N},\mathbf{m_0}) = \{\sigma \mid \mathbf{m_0}\xrightarrow{\sigma}\mathbf{m}\}$.

A marking $\mathbf{m}'$ is *reachable* from another marking $\mathbf{m}$ if and only if there exists a firing sequence σ so that $\mathbf{m}\xrightarrow{\sigma}\mathbf{m}'$. Given a Place/Transition system, $\langle\mathcal{N},\mathbf{m_0}\rangle$, the set of markings *reachable* from $\mathbf{m_0}$ in $\mathcal{N}$ is denoted as $\mathrm{RS}(\mathcal{N},\mathbf{m_0})$ and is called the *reachability set*. If we consider any reachable marking, $\mathbf{m} \in \mathrm{RS}(\mathcal{N},\mathbf{m_0})$, $\mathrm{RS}(\mathcal{N},\mathbf{m})$ denotes the set of markings reachable from it. The *reachability graph* is a labelled directed graph whose nodes are the reachable markings, and the arcs are given by the occurrence relation for transitions, that is, there is an arc from node $\mathbf{m}$ to node $\mathbf{m}$' if and only if $\exists t \in T \,.\, \mathbf{m}\xrightarrow{t}\mathbf{m}'$.

The *state equation* of a marked net is an algebraic equation that gives a necessary condition for the reachability of a marking from the initial marking: a markings $\mathbf{m} \in \mathbb{N}^{|P|}$ such that $\exists\boldsymbol{\sigma} \in \mathbb{N}^{|T|} \,.\, \mathbf{m} = \mathbf{m_0} + \mathbf{C}\cdot\boldsymbol{\sigma}$ is said to be *potentially reachable*. The *potentially reachability set* of a net is the set of solutions for the *state equation*. This set will be denoted as $\mathrm{PRS}(\mathcal{N}$,$\mathbf{m_0})$. The *potentially reachability graph* is the extension of the reachability graph whose nodes are the marking solutions for the net state equation.

A place $p \in P$ is *k–bounded* if, and only if, $\forall\mathbf{m} \in \mathrm{RS}(\mathcal{N},\mathbf{m_0}) \,.\, \mathbf{m}[p] \leq k$. A net system is *k–bounded* if, and only if, each place is k–bounded. A net system is *bounded* if, and only if, there exists some k for which it is k–bounded. A net system is *safe* (or *binary*) if, and only if, each place is 1–bounded. A net $\mathcal{N}$ is *structurally bounded* if, and only if, it is bounded no matter which $\mathbf{m_0}$ is the initial marking. Related with this, the *structural bound* of a place is: $\mathbf{sb}[p] = max\{\mathbf{m}[p] \mid \mathbf{m} \in \mathrm{PRS}(\mathcal{N},\mathbf{m_0})\}$.

A transition is *live* if it can be fired from every reachable marking; a transition t is dead for a reachable marking $\mathbf{m}$ if and only if $\forall\mathbf{m}' \in \mathrm{RS}(\mathcal{N},\mathbf{m}) \,.\, \neg(\mathbf{m}\xrightarrow{t})$. It is *dead* if it is dead for the initial marking. A marked Petri net is *live* when every transition is live, and it is *deadlock–free* when every reachable marking enables at least a transition. A net, $\mathcal{N}$, is *structurally live* if, and only if, there exists an initial marking, $\mathbf{m_0}$, such that $\langle\mathcal{N},\mathbf{m_0}\rangle$ is live.

Given a net system, $\langle\mathcal{N},\mathbf{m_0}\rangle$, and a marking $\mathbf{m} \in \mathrm{RS}(\mathcal{N},\mathbf{m_0})$, $\mathbf{m}$ is a *home state* of the system if, and only if, $\forall\mathbf{m}' \in \mathrm{RS}(\mathcal{N},\mathbf{m_0}) \,.\, \mathbf{m} \in \mathrm{RS}(\mathcal{N},\mathbf{m}')$. The *home space* of the net system is the set of its home states.

A *path is a sequence of nodes* $x_1x_2\ldots x_n$ belonging to $P \cup T$ such that $x_{i+1} \in {x_i}^{\bullet}$, for all $i \in \{1,\ldots,n-1\}$. A path is *simple* if all the nodes are different. A *circuit* is a path such that $x_1 = x_n$. A *simple circuit* is a circuit such that all the nodes, except the first one and the last one, are different.

A.2 Some structural objects

Flows (*Semiflows*) are integer (natural) annuller of matrix $\mathbf{C}$. Right and left annullers are called *T–(Semi)flows* and *P–(Semi)flows*, respectively. The *support* of T–(Semi)flows is given by: $\|x\| = \{t \in T \mid \mathbf{x}[t] > 0\}$ and the *support* of P–(Semi)flows is given by: $\|y\| = \{p \in P \mid \mathbf{y}[p] > 0\}$. A (Semi)flow is called *minimal* when its support is not a strict superset of the support of any other, and the greatest common divisor of its elements is one.

A T–Semiflow, $\mathbf{x}$ defines the following invariant property:

$$\exists \mathbf{m_0} \,.\, \exists \sigma \in \mathrm{L}(\mathcal{N}, \mathbf{m_0}) \,.\, \mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}' = \mathbf{m_0}, \text{ and } \mathbf{x} = \boldsymbol{\sigma}$$

(token conservation law.)

A P–Semiflow, $\mathbf{y}$ defines the following invariant property:

$$\forall \mathbf{m_0} \,.\, \forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) \,.\, \mathbf{y} \cdot \mathbf{m} = \mathbf{y} \cdot \mathbf{m_0}$$

(cyclic behavior law.)

Several structural properties are defined in terms of the existence of certain annullers, or similar vectors:

- $\mathcal{N}$ is consistent (structurally repetitive) if, and only if $\exists \mathbf{x} > 0\,.$ such that $\mathbf{C} \cdot \mathbf{x} = (\geq)0$.
- $\mathcal{N}$ is conservative (structurally bounded) if, and only if $\exists \mathbf{y} > 0\,.$ such that $\mathbf{y} \cdot \mathbf{C} = (\leq)0$.

The first two properties are related to the existence of potential sequences containing all the transitions that could be fired repeatedly (repetitive sequences or vectors), because either they do not change or they increase the marking. The second two properties are related to the existence of either token conservation or token non–generation properties for the set of places.

Given $\mathcal{N}' = \langle P', T', \mathbf{C}' \rangle$ a subnet of the net, $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$, if $\mathbf{y}'$ is a P–Semiflow of $\mathcal{N}$’, we can extend this vector to the whole net in the following way: $\mathbf{y}'_{P'|P}$ is a vector such that: $\forall p \in P' \,.\, \mathbf{y}'_{P'|P}[p] = \mathbf{y}'[p]$ and $\forall p \in P \setminus P' \,.\, \mathbf{y}'_{P'|P}[p] = 0$.

Given $\mathcal{N}$ an ordinary Petri net, a subset of places $D \subseteq P$ is a *siphon* of the net $\mathcal{N}$ if, and only if, ${}^\bullet D \subseteq D^\bullet$. A subset of places $E \subseteq P$ is a *trap* of the net $\mathcal{N}$ if, and only if, $E^\bullet \subseteq {}^\bullet E$. A siphon (trap) is minimal if, and only if, it does not properly contain another siphon (trap). Siphons have the important property that, if at a given marking the siphon is unmarked, it never will be marked. Traps have the reverse property: once they get marked, they will remain marked. For non–ordinary nets, we define *siphons* (*traps*) as the siphons (traps) that can be obtained in a net that is a copy of the original one, substituting all the arcs' weights for one.