xx (108184.1)

Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional

U n i d a d G u a d a l a j a r a

"Scheduling de Sistemas de Eventos Discretos Modelados por Máquinas de Estados Sincronizadas"

Tesis que presenta
José Luis Córdova Barba

Para obtener el grado de
Maestro en Ciencias

En la especialidad de
Ingeniería Eléctrica

Guadalajara, Jal., Enero de 2003

"Scheduling de Sistemas de Eventos Discretos Modelados por Máquinas de Estados Sincronizadas"

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:

José Luis Córdova Barba

Ingeniero en Comunicaciones y Electrónica
Universidad de Guadalajara
CUCEI 1994-1998

Becario del CONACyT, expediente no. 143885

Directores de Tesis
Dr. Antonio Ramírez Treviño
Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Enero de 2003

Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

U n i d a d   G u a d a l a j a r a

"Scheduling of Discrete Event Systems Modelled by Synchronised State Machines"

Thesis submitted by
José Luis Córdova Barba

For the degree of
Master of Sciences

In the speciality of
Electrical Engineering

Guadalajara, Jal., January 2003

“Scheduling of Discrete Event Systems Modelled by Synchronised State Machines”

Thesis of Master of Sciences
Electrical Engineering

By

José Luis Córdova Barba

Communications and Electronics Engineer
Universidad de Guadalajara
CUCEI 1994-1998

Scholarship CONACyT, no. 143885

Thesis Advisors
Dr. Antonio Ramírez Treviño
Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, January 2003

Resumen

Esta tesis aborda el problema de scheduling de Sistemas de Evento Discretos (SED) modelados con Redes de Petri Temporizadas (RPT). Se presenta un método para calcular un schedule óptimo para un subclase de RPT llamada Máquinas de Estados Sincronizadas (MES).

Se muestra que las MES tienen la capacidad de describir una gran variedad de SED; y además, las MES presentan la propiedad de optimalidad local, la cual es utilizada en el método propuesto.

La técnica propuesta reduce el proceso de análisis a través de la búsqueda de una base de T-semiflujos para cada máquina de estados, para la cual se da un vector del ratio de visitas.

El algoritmo incluye una heurística que esta basada en la idea de que un tiempo de ciclo mínimo en el sistema es equivalente a la máxima utilización de los recursos; de forma tal que el algoritmo encuentra un schedule óptimo en tiempo polinomial.

Se incluyen dos ejemplos detallados que ilustran la aplicación del algoritmo propuesto.

Abstract

This thesis addresses the problem of scheduling discrete event dynamic systems (DEDS) modelled with timed Petri nets (TPN). A method to compute an optimal schedule for a TPN subclass called Synchronised State Machines (SSM) is presented.
It is shown that the expressiveness of SSM copes with a wide variety of DES; furthermore SSM exhibit the local optimality property that is exploited in the proposed method.
The technique herein proposed reduces the analysis process by searching a T-semiflows basis for every state machine, which the visit ratio vector is given. The algorithm includes a heuristic that is based in the idea that a minimum time schedule is equivalent to the maximum transition utilisation schedule; so the algorithm obtains an optimal schedule in polynomial time.
It is included two detailed examples that illustrate the application of the proposed method.

Contents

Chapter 1

INTRODUCTION

The necessity to improve the benefits of a discrete system, leads to the scheduling theory. Efficient scheduling of resources is critical to the proper functioning of systems in today's competitive environment. Scheduling focuses on theoretical as well as applied aspects of the resources optimisation. This optimisation seeks the best performance of a system with the minimum number of resources or invested time, fulfilling the characteristic system's restrictions.

Due to the discrete nature of man made systems, the scheduling problem is a very complex problem, either for the combinatory quantity of states or for the restriction to entire values of the system parameters.

As consequence, different methods have been developed to solve specific characterizations of the problem. Unfortunately, many of them require a large amount of time and in some cases they obtain a solution near to the optimal one.

1.1 Introduction

Systems optimisation has been a need since humans require improving systems performance due to the limitation of resources. Optimisation is not a single problem by itself, it includes some stages, two of the most important stages are *planning* and *scheduling*. Frequently these terms are confused; an intuitive definition will be introduced in order to state the differences between these two stages.

Planning is the stage where it is stated what is going to be done. In this stage, time and resources are assigned to tasks, and also the relationship between tasks is defined. The result of this stage is a plan. This stage presents its own optimisation problems.

Scheduling deals with the allocation of scarce resources to tasks over time. The aim of this stage is to improve systems performance by defining the tasks starting time depending on the tasks relationship. Scheduling can be formally represented as a decision-making process with the goal of optimising one or more objectives.

Resources and tasks in a organization can take many forms. The resources may be machines in a workshop, runway at an airport, crews at a construction site, processing units in a computing environment, and so on. Each task may have certain priority level, an earliest possible starting time, and a due date. The objectives can also take many forms. One objective may be the minimization of the completion time of the last task, the minimization of the number of tasks completed after their respective due dates, or the minimization of the cost of a project.

Since one of the main difficulties that optimisation problem presents is the intractability due to the combinatory quantity of states or the restriction to entire values of the system parameters. Many approximations have been studied, which have been classified on different methods: operation research, artificial intelligence, etc.

Researches in scheduling theory have evolved over the last fifty years and have been the subject of much significant literature. Not surprisingly, approaches have been formulated from a diverse spectrum of researchers ranging from management scientists to production workers. However with the advent of new methodologies, such as neural networks and evolutionary computation, researchers from fields such as biology, genetics and neurophysiology have also become regular contributors to scheduling theory emphasising the multidisciplinary nature of this field.

1.2 Problem Phrasing

In scheduling terminology, a distinction is often made among a *sequence*, a *schedule*, and a *scheduling policy*. A sequence usually corresponds to a permutation of tasks. A schedule usually refers to an allocation of tasks within a more complicated setting of tasks, allowing possible for preemptions of tasks by other tasks that are leased at later points in time. The scheduling policy prescribes an appropriate action for each state the system may be in.

There exist optimisation functions allowing to formulate the schedule objectives mathematically, and to evaluate the obtained solution with regard to a desired objective.

Depending on the system's structure, different classifications to the scheduling problem are proposed, some of they are:

- Job-Shops scheduling problem: It consists on determining the order and the starting time of the processing of a piece in each machine.
- Assembly lines balance: It consists on determining the minimal number of machines to be assigned to a task, or to minimize the production cycle time.
- Project Scheduling: It consist on determining the starting time of a task depending on the precedence relationship between tasks in order to minimize the cost and the invested time in the project..

The scheduling problem is an NP problem, in other words, the required time to find out an optimal schedule cannot be represented by a polynomial in the size of the input data. To deal with NP complete problems, the divide and concord technique is used to improve results in the computation of the optimal solution. In the scheduling problem, the divide and concord technique is translated to achieve local optimisations.

Local optimisation technique consists in three phases:

1. To divide a system into subsystem,
2. to optimise the subsystems, and
3. to integrate the optimal solutions of the subsystems.

This technique simplifies the computation of the optimal schedule substantially, but not all the systems exhibit the local optimality property. Therefore, in this work will be shown that system which previous results prove that is not possible to achieve local optimisations in, can be locally optimised through a statistic perspective.

In the solution of the scheduling problem, different methods are combined. Some of the methods that will be used to deal with and give solution to scheduling problem are mentioned below.

1.2.1 Operations Research Methods

Operation research seeks the determination of the best (optimum) course of action of a decision problem under limited resources. The term *operations research* quite often is associated almost exclusively to the use of *mathematical techniques* to model and analyse decision problems. Although mathematics and mathematical problems represent a cornerstone of operation research, there are more factors involved in problem solving than the construction and solution of mathematical models. Specifically, decision problems (as scheduling) usually include important intangible factors that cannot be translated directly in terms of the mathematical model. Foremost among these factors is the presence of the human element in almost every decision environment. Indeed, in decision situations where the human behaviour influence the decision problem, the solution obtained from the mathematical model is deemed impractical.

British scientists were the pioneers on operations research during World War II. Although their works were concerned primarily with the optimum allocation of the limited resources of war materiel, the team included scientist from such fields as sociology, psychology, etc.

Linear Programming

The success of an Operation Research technique is ultimately measured by its acceptance as a decision-making tool. Since its introduction in the late 1940's, linear programming has proven to be one of the most effective Operations Research tool. Its success resides on its flexibility in describing multitudes of real-life situations. Additionally, the availability of very efficient computer codes for solving very large linear programming problems is an important factor in the widespread use of the technique.

Linear programming is a deterministic tool, meaning that all the model parameters are assumed to be known with certainty. In real life, however, it is rare to encounter a problem in which true certainly prevails. Linear Programming technique compensates for this 'deficiency' by providing systematic postoptimal and parametric analyses that allow the decision maker to test the sensitivity of the "static" optimum solution to discrete or continuous changes in the parameters of the model.

One and the most used methods in Linear Programming is the Simplex Method.

1.2.2 Artificial Intelligence Methods

The field of Artificial Intelligence attempts to understand intelligent entities. Thus, one reason to study it is to learn more about humankind. But unlike philosophy and psychology, which are also concerned with intelligence, Artificial Intelligence strives to build *intelligent* entities as well as understand them. Although no one can predict the future in detail, it is clear that computers with human-level intelligence (or better) would have a huge impact on our everyday lives and on the future course of civilization.

Artificial Intelligence techniques comprise methods for searching 'problems spaces', planning sequences of actions to solve decision problems, methods of reasoning and deduction, and techniques for representing knowledge. Any particular application will typically call on numerous AI methods and techniques. Historically a proportion of AI research has been application driven whilst other work has looked at extending the available methods and techniques.

AI is often characterized as the attempt to build **computational models** of intelligent processes. The art of AI modelling consist of the ability to specify the problem to be solved. Specification is the art of computer modelling.

Search Strategies

The majority of work in the area of Search has gone into finding out the right **Search strategy** for a problem. In the study of the decision making problems (as scheduling) the strategies will be evaluated in terms of four criteria:

- **Completeness:** Guaranties to find out a solution when there is one.
- **Time complexity:** Determines how much time takes to find out a solution.
- **Space complexity:** Determines how much memory it is needed to perform the search.
- **Optimality:** Determines the strategy used to find out the highest-quality solution.

There exist different search strategies, but they can be classified into two classes:

- uninformed search, and
- informed search.

This classification is done depending if the strategy has information or not about the cost of the goal.

Uninformed search strategies have no information about the number of steps of the path cost from the current state to the goal state; all they can do is distinguish the goal state from a nongoal state. They can find solutions to problems by systematically generating new states and testing them against the previous generated. Unfortunately, these strategies are incredibly inefficient in most cases.

Informed search strategies can find solutions more efficiently. They explore more states by combining different heuristics and exploring not only the best-first state. Therefore, informed search strategies are able to distinguish a local optimum from the optimal one. Some informed search strategies are:

One and the most used search strategies is the A* search strategy.

1.3 Discrete Event Dynamic Systems

Modern technology has increasingly created dynamic systems which are not easily described by ordinary or partial differential equations. The *state* of such dynamic systems changes only at certain time instants instead of continuously. Such are called *discrete event dynamic systems* (DEDS's) as opposed to the more familiar continuous variable dynamic systems in the physical world that are described by differential equations.

Almost all DEDS are man made:

- Computer Systems (operating, communication, and processing systems),
- Discrete Production Systems (manufacturing systems),
- Transport Systems (traffic systems, airports, and train stations),
- Military Commands,
- Engineering (Control Systems),

- Mathematics,
- Chemistry, and
- Judicial Systems.

Although there exist many different types of DEDS's, they share some common characteristics, which include:

Event-driven. A DEDS may be viewed as a sequence of events. The completion of an activity may initiate one or more new activities. Moreover, the order of occurrence of events is not necessarily unique.

Concurrent. Many activities take place simultaneously.

Asynchronous. The evolution of system events is aperiodic. This may be due to variable processing routines and processing times.

Deadlock. When a DEDS reaches a particular state, where the systems cannot evolve any more. A well-designed system ought to be able to detect and resolute deadlock states.

Conflict. This may happen when two or more processes require a common resource at the same time.

1.3.1 DEDS Performance

Performance evaluation plays a vital role in the operation of a DEDS. In the life cycle of a DEDS, decision making is involved at various stages of planning, design and operation of the system. Therefore, a good scheduling policy for a DEDS will lead to an optimum performance.

Performance models for DEDS's can be partitioned into two fields: one is *models simulation*, and the other is *models analysis*.

In the case of models simulation, models are "run" rather than "solved"; that is, an artificial history of the system is generated based on the model assumptions, and observations are collected to be analysed and to estimate the true system performance measures. A model simulation can be built as accurate as one desires, limited only by cost and time. However, models simulation validation is quite difficult.

Models analysis uses the mathematical deductive reasoning to solve the performance analysis. One of the most used analytical modelling tools for DEDS is the Petri nets. Petri nets are a formal tool to model the flow of information and control in systems, especially those which exhibit asynchronous and concurrent properties.

In the first instance it must to be asked whether the model is *consistent*. A model is consistent if it does not generate contradictions, if its various components fit together so as to form a coherent whole. It can also be asked if the model is *complete*. Does the specification cover the range of desired cases to model, does it have the need scope? A further feature which is hard to define absolutely is *elegance*. Does the model represent the original system in an elegant fashion, are the general principles captured without too many special exceptions, or does the model has to be resorted to little bits and pieces which make it work but which have no realization in the original system? It can also be seen if the model is *superficial*. To what extent have simplifying assumptions been made which simplify away crucial problems? Further, it can be asked if the model is *predictive*. Does it generate novel behaviour which nevertheless seems reasonable? This often arises out of computational models which have a degree of complexity and generality. Indeed the ability of computational models to reveal the consequences of complex changing processes is a feature which no pen and paper model can easily conveny. Finally, it ought to be considered whether the model is capable of *refutation*; can it be lain against the original and said to be adequate or inadequate, right or wrong?

1.4 Outline of the work

This work focuses on both theory and application of scheduling. The theoretical side deals with the detailed sequencing and scheduling of tasks. Given a set of tasks to be done, the problem is to sequence the tasks, subject to precedence relations, in such a way that one or more performance criteria are optimized.

Thousands of scheduling problems and models have been studied and analysed in the past. Obviously, only a reduced number are considered in this work.

Although the applications driving the models in this work come mainly from manufacturing and production environments, it is clear that scheduling plays a role in a wide variety of situations. The models and concepts dealt with in this work are also applicable to other applications or environments.

This work is divided in five chapters. Chapter 1 defines what scheduling is, and states the problem. Chapter 2 introduces Petri nets as a formal tool to model and analyse DEDS, and presents previous results for scheduling in free choice Petri nets. Chapter 3 introduces a subclass of free choice Petri nets which previous results prove that is not possible to achieve local optimisations in, and exhibits two DEDS modelled by this subclass. Chapter 4 presents an heuristic (using local optimization by isolating the optimization of parallel branches) to find out an optimal schedule for systems modelled by Synchronised State Machines Petri Nets. Chapter 5 contains the conclusions, advantages and extensions of the work done here in.

1.5 Conclusions

The scheduling problem is an optimisation problem for discrete event systems. To solve this problem, there exist different methods such as operations research and artificial intelligence. The main advantage of these methods is their flexibility to combine them; providing new and better heuristics for specific applications like the scheduling problem.

Operations research methods can solve some specific scheduling problems by the system representation as a set of linear equations. In the practice is difficult to apply these methods on discrete event systems, either for the combinatory quantity of states or for the restriction to integer values of the system parameters. Operations research must be viewed as both a science and an art. The science aspect lies in providing mathematical techniques and algorithms for solving appropriate decision problems. Operations Research is an art because success in all the phases that precede and succeed the solution of a mathematical model depends largely on the creativity and personal abilities of the decision making analysts. Thus gathering of the data for a model construction, validation of the model, and implementation of the obtained solution will depend on the ability of the Operations Research team to establish good lines of communication with the sources information as well as with the individuals responsible for implementing recommended solutions.

Artificial intelligence based methods represent systems describing its behaviour and the inner relationships. These methods find out a good solution in a reasonable amount of time, considering that the solution fulfils the stated constraints. In most of the cases it is not possible to prove if the solution is the optimal one.

The difficulties to solve a scheduling problem depend on the accuracy of the model used for the analysis and on the reliability of the input data required. Therefore, Petri nets are chosen a as modelling tool since they capture DEDS features through a strong mathematical support allowing to analyse properties and characterize systems from a formal model.

Different methods can be combined and applied to systems modelled by Petri nets, taking advantage of having a unique model, simplifying the scheduling problem abstraction, and exposing systems properties simplifying the computation of the optimal schedule by heuristics.

Chapter 2

SCHEDULING OF DEDS MODELLED BY PETRI NETS

There exist different formal tools to model DEDS. Each one of these different tools captures some DEDS properties. Petri nets are a graphical, mathematical, and formal modelling tool applicable to many systems. This formalism is used in this work to describe DEDS, since they are suitable to capture and analyse most of the characteristics of the DEDS, for example concurrence, causal relationship, mutual exclusion, etc. Petri nets are also useful to analyse qualitative and quantitative properties of DEDS.

To describe the time in DEDS, this work uses a class of Petri nets called timed Petri nets. This class of Petri nets are defined introducing the concept of time into the Petri net structure, providing it the ability to represent actions' delays or duration.

This chapter summarizes concepts and notation of Petri nets and timed Petri nets, and presents previous results in scheduling theory using timed Petri nets.

2.1 Introduction

Petri nets were first introduced in the years 1960-1962 by Carl Adam Petri, when he wrote his PhD on a general purpose mathematical tool for describing relations existing between conditions and events, capable to model those aspects of systems behaviour which can be expressed in terms of causality and choice. Since then, Petri nets become a useful tool in considerable research areas due to they allow modelling some system characteristics such as process synchronization, asynchronous events, sequential operations, concurrent operations, and conflicts or resource sharing.

Complex requirement specifications can be easily represented graphically using Petri nets instead of using ambiguous textual descriptions or mathematical notation which is difficult to understand by the customer. Evolution rules of Petri nets are easy to understand, even for people who are not familiar with the details of Petri nets.

Petri nets combine a graphical environment with a powerful underlying mathematical formalism. As a mathematical tool, Petri net models can be described by a set of linear algebraic equations, reflecting the behaviour of the system. This allows performing a formal check of properties related to the behaviour of the underlying system.

In real world every event is time related, therefore DEDS's models should include the time notion in their structure. Ordinary Petri nets do not capture the time, thereby considering this necessity, several extensions to Petri nets including the "time" have been proposed. When a Petri net contains a time variable, it becomes a *Timed Petri Net.*

Typical examples of application of DEDS areas are manufacturing systems, communication networks, traffic control systems, military command and control systems, etc. Petri nets also find applications in a number of different disciplines including engineering, manufacturing, business, chemistry, mathematics, and even within the judicial system.

2.2 Petri Nets Basics

2.2.1 Primitives

Since the beginning, there have been variations on the initial structure of Petri nets, but most of these variations are simply additions to the basic definition of a Petri net.

Definition 2.1 *A Petri Net Structure (PN) is the 4-tuple* $N = (P, T, Pre, Post)$, *where*

$P = \{p_1, p_2, ..., p_n\}$ *is a finite set of places*

$T = \{t_1, t_2, ..., t_m\}$ *is a finite set of transitions*

Pre: $P \times T \rightarrow \mathbb{N}$ *is the previous incidence function representing directed arcs from places to transitions,* *where $\mathbb{N}$ is a set of nonnegative integers.*

Post: $T \times P \longrightarrow \mathbb{N}$ is the subsequent incidence function representing directed arcs from transitions to places, where $\mathbb{N}$ is a set of nonnegative integers.

Definition 2.2 *Ordinary Petri Nets (OPN) are Petri Nets whose Pre and Post incidence functions take values in $\{0,1\}$. The incidence function of a given arc in a nonordinary Petri net is called weight or multiplicity.*

A Petri net is a bipartite graph whose two different kind of nodes are named places (P) and transitions (T), where $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$. Graphically, transitions are represented by bars or boxes, see Figure 2.1 a); and places by circles, see Figure 2.1 b). The *Pre* function denotes a flow relation of the net, and graphically is represented by arcs (arrows) leading from places to transitions, see Figure 2.1 c). The *Post* function denotes another flow relation of the net, different from the *Pre* function, and graphically is represented by arcs leading from transitions to places, see Figure 2.1 d). An *input place* p_i of a transition t_j, denoted by $Pre(p_i, t_j) = 1$, defines a directed arc from p_i to t_j. Similarly, an *output place* p_i of a transition t_j, denoted by $Post(t_j, p_i) = 1$, defines a directed arc from t_j to p_i. If $Pre(p_i, t_j) = k$ ($Post(t_j, p_i) = k$), then there exist k directed (parallel) arcs connecting place p_i to transition t_j (connecting transition t_j to place p_i). Usually, in the graphical representation, parallel arcs connecting a place (transition) to a transition (place) are represented by a single directed arc labelled with its multiplicity, or weight k, see Figure 2.1 d).

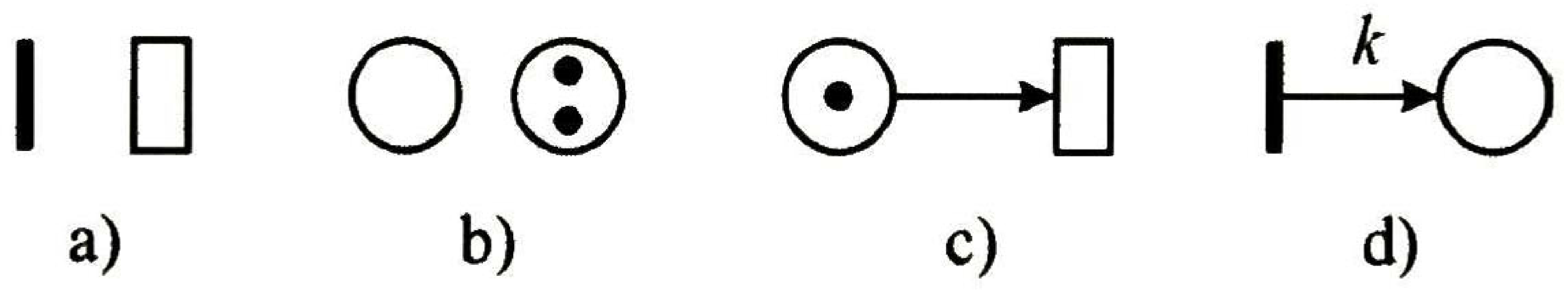

Figure 2.1: Nodes and arcs of a Petri net

Given a node $x | x \in P \cup T$, the set $\bullet x = \{y | Pre(x,y) > 0\}$ is defined as the pre-set of x and the set $x\bullet = \{y | Post(x,y) > 0\}$ is defined as the post-set of x. In other words, the elements in the pre-set (post-set) of a place are its input (output) transitions. Similarly, the elements in the pre-set (post-set) of a transition are its input (output) places.

Definition 2.3 *Paths, circuits*

A path of a net $N = (P, T, Pre, Post)$ is a nonempty sequence of nodes $x_1, ..., x_k$ which satisfies (x_1, x_2), (x_2, x_3), $\ldots$, $(x_{k-1}, x_k) \in Pre \cup Post$, and no element occurs more than once in it. A path is said to lead from x_1 to x_k.

A path leading from a node x to a node y is a circuit if $x = y$. Observe that a sequence containing one element is a path but no a circuit, because for every node x, $(x, x) \notin Pre \cup Post$.

A net $N = (P, T, Pre, Post)$ is called weakly connected (or just connected) if for every two nodes x and y, *there exist a path leading from x to y.*

A net $N = (P, T, Pre, Post)$ is called strongly connected if for every two nodes x and y, there is a path leading from x to y, and another leading from y to x.

Note that every strongly connected net is also weakly connected.

Definition 2.4 *Incidence Matrix*

Let $N = (P, T, Pre, Post)$ be a net where $P = \{p_1, p_2, ..., p_n\}$ and $T = \{t_1, t_2, ..., t_m\}$. The incidence matrix $\mathbf{N} = [n_{ij}]$ of the Petri net N, where $i = 1, ..., n$, and $j = 1, ..., m$, is defined by $n_{ij} = Post(t_j, p_i) - Pre(p_i, t_j)$. Similarly the pre- and post-incidence matrices are defined as $\mathbf{N}^- = [a_{ij}]$ and $\mathbf{N}^+ = [b_{ij}]$, where $a_{ij} = Pre(p_i, t_j)$ and $b_{ij} = Post(t_j, p_i)$.

Notice that different nets can have the same incidence matrix, due to self loops (for some transition t and some place $p : p \in \bullet t \cap t\bullet$). A net containing no self loops is called *pure*. It is easy to see that the mapping γ between the set of pure nets and the sets of matrices on $\{-1, 0, 1\}$ given by $\gamma(N) = \mathbf{N}$ is a bijection.

Example 2.1 *The incidence matrix of the pure Petri net shown in Figure 2.2 is:*

$$\mathbf{N} = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

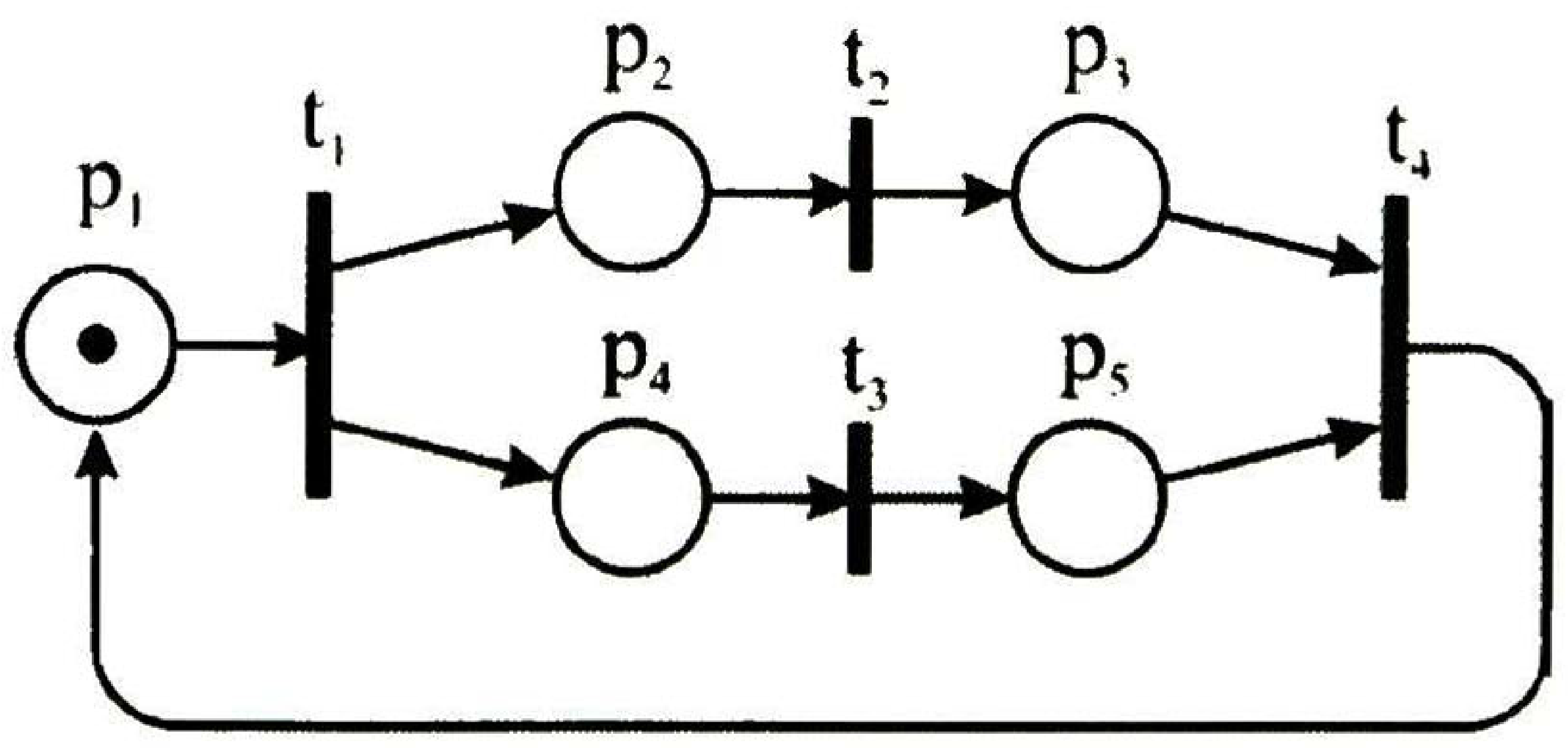

Figure 2.2: A pure Petri net

Example 2.2 *The incidence matrix of the non-pure Petri net shown in Figure 2.3 is:*

$$\begin{bmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Definition 2.5 *Marking*

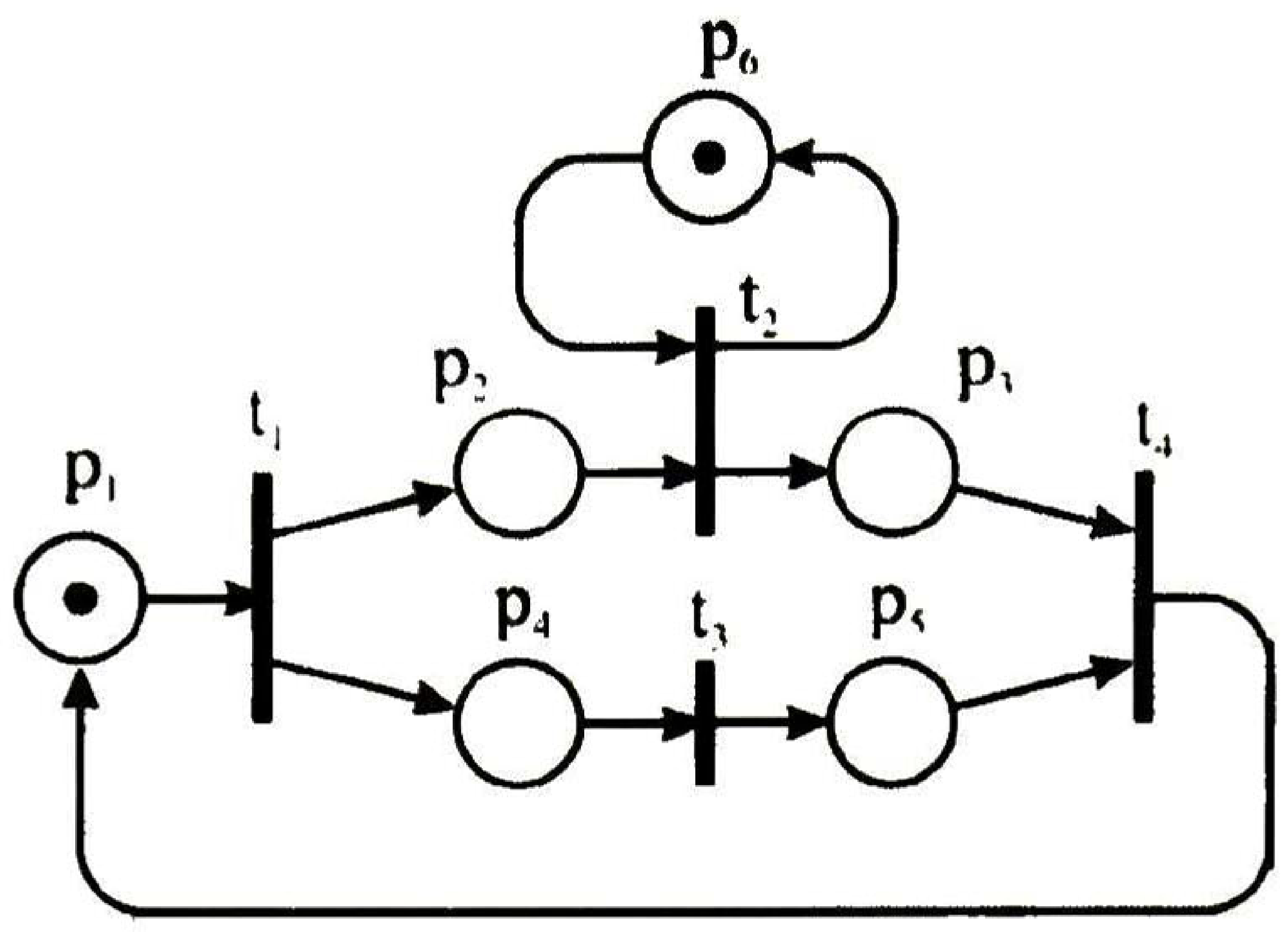

Figure 2.3: A nonpure Petri net

A function $M : P \to \mathbb{N}$. (usually represented in vector form) is called marking. A marked Petri Net (N, M_0) is a Petri Net N with an initial marking M_0.

A place is marked in a marking M if $M(p) > 0$. Graphically, a marking $M(p)$ is represented by tokens (black dots) into the place p. A token is a primitive concept for Petri nets (like transitions and places). Tokens reside into places and control the execution of the transitions of the net.

Definition 2.6 *Cluster*

Let x be a node of a Petri net. The cluster of x, denoted by $Cl(x)$,is the minimal set of nodes such that

- $x \in Cl(x)$
- If a place p belongs to $Cl(x)$ then $p\bullet$ is included in $Cl(x)$, see figure 2.4 a), and
- If a transition t belong to $Cl(x)$ then $\bullet t$ is included in $Cl(x)$, see figure 2.4 b).

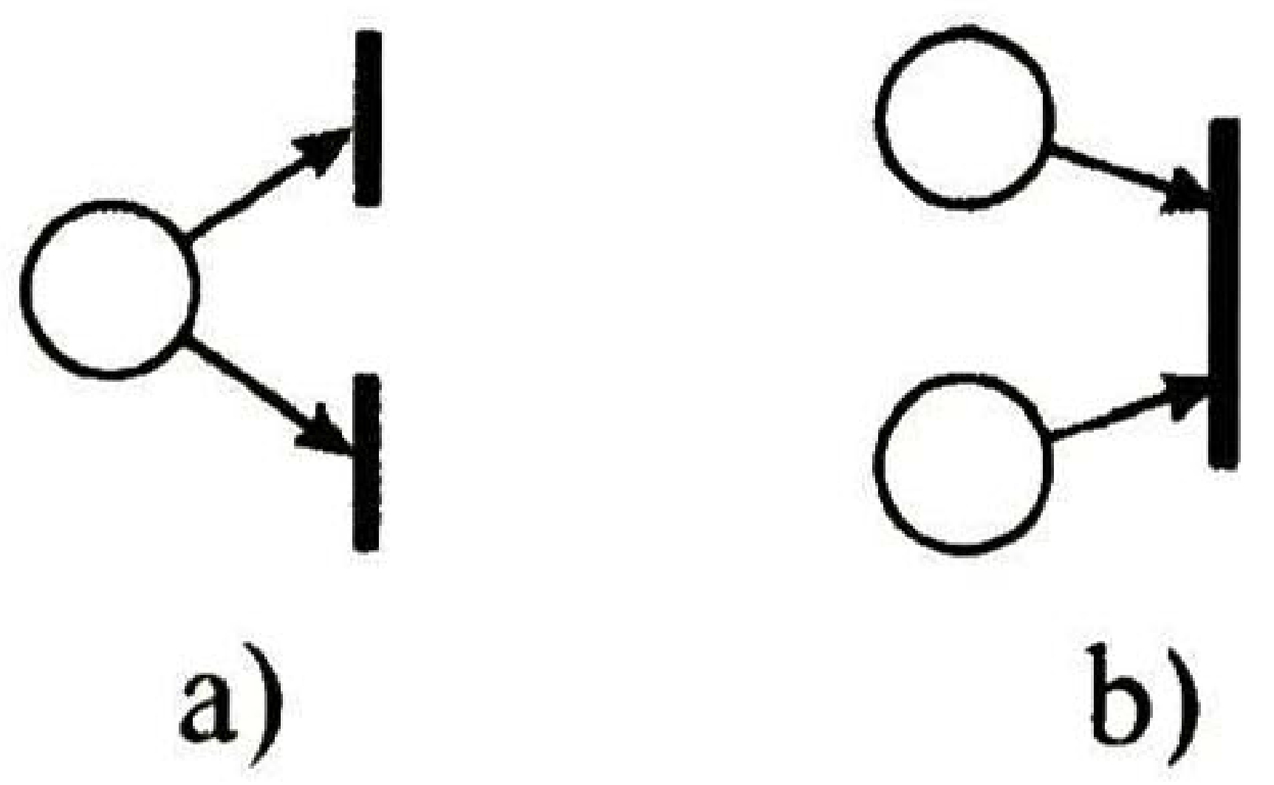

Figure 2.4: a) A cluster with one place and two transitions, b) a cluster with two places and one transition.

Figure 2.5 shows a Petri net together with the partition of its nodes into clusters.

Definition 2.7 *A system (Petri net) is a pair (N, M_0) where*

- N is a connected net having at least one place and one transition, and
- M_0 is a marking of N called initial marking.

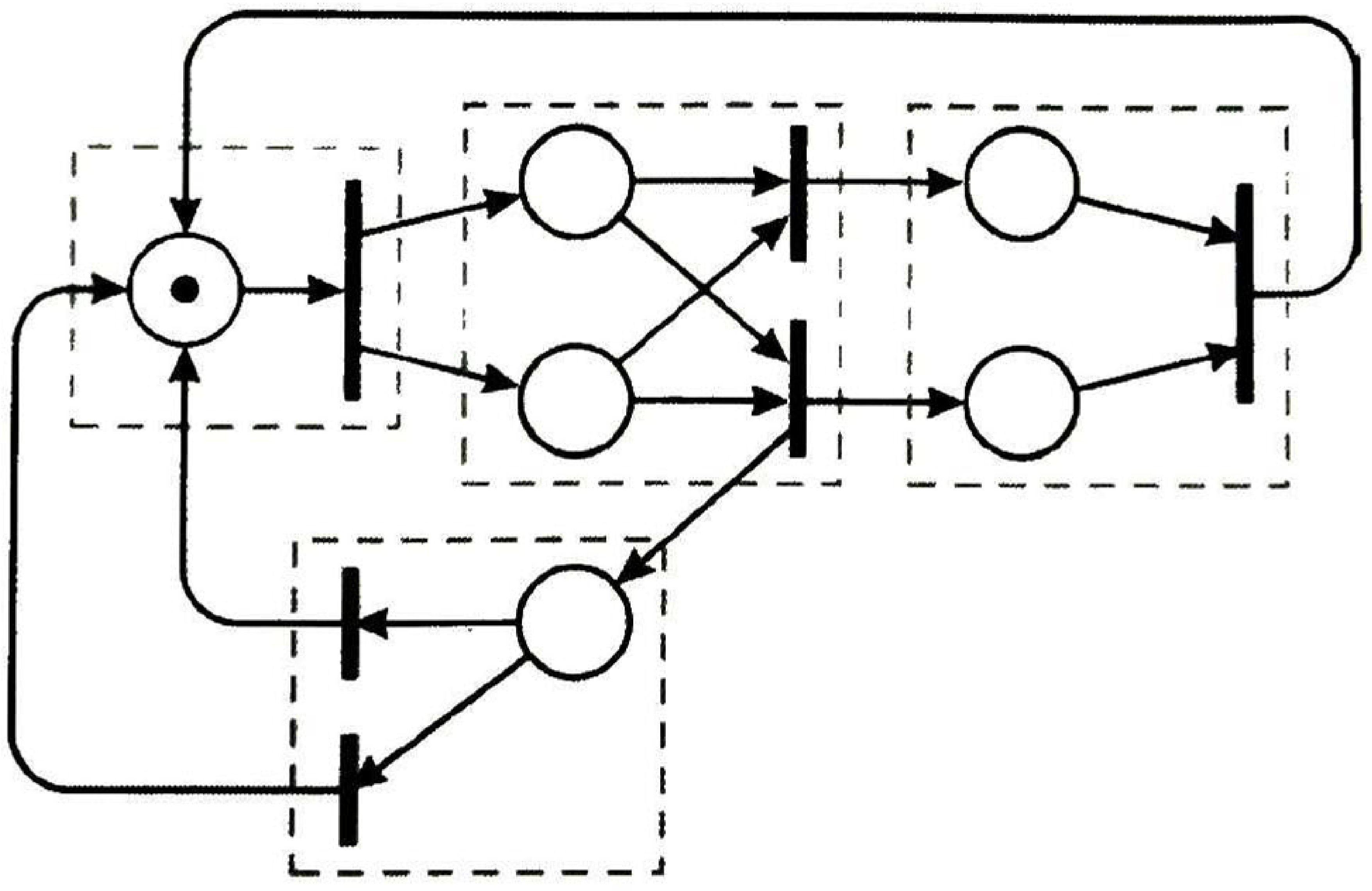

Figure 2.5: Partition of the nodes of a Petri net into clusters.

2.2.2 Evolving Rules

The execution of a Petri net is controlled by the number and distribution of tokens in the Petri net, and such number and position of tokens may change during the execution. A Petri net evolves by *firing enabled* transitions. Now, the enabling rule and firing rule of a transition will be introduced, these rules govern the tokens flow in the Petri net.

1. *Enabling Rule:* A transition t is said to be *enabled* if each input place p of t contains at least the number of tokens equal to the weight of the directed arc connecting p to t, i.e.
   $\forall p \mid p \in \bullet t : M(p) \geq Pre(p,t)$
2. *Firing Rule:*
   (a) An enabled transition t may or may not fire depending on the system nature or control law; and
   (b) The firing of an enabled transition t removes $Pre(p,t)$ tokens from each input place. It also adds $Post(t,p)$ tokens in each output place.

Mathematically, the firing of a transition t at the marking M yields to a new marking M', this firing of transition t is denoted by $M \xrightarrow{t} M'$. The *Marking Equation* defines the relationship between an initial marking M and a reached marking M' by means of the firing of transitions:

$$M'(p) = M(p) - Pre(p,t) + Post(t,p) \qquad \forall p \mid p \in P \tag{2.1}$$

Notice that since only enabled transition may fire, the number of tokens in each place remains nonnegative when a transition is fired. The transition firing cannot remove a token that not exists.

A sequence of transitions $\sigma = t_1 t_2 ... t_n$ is a firing sequence of (N, M_0) if and only if there exists a sequence of markings $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 ... \xrightarrow{t_n} M_n$ such that the marking M_i enables t_{i+1}. In this case M_n is said to be *reachable* from M_0 by firing σ, and is denoted by $M_0 \xrightarrow{\sigma} M_n$. The expression $M \xrightarrow{\sigma}$ denotes a *firable sequence* σ from a marking M.

A marking is called *dead marking* if it enables no transition in the net. Analogously a transition is called *dead transition* if no reachable marking enables it.

Definition 2.8 *Parikh vector*

A vector $\vec{\sigma} : P \rightarrow \mathbb{N}$. is called the Parikh vector of the firable sequence σ. This Parikh vector maps every transition $t \in T$ into the number of occurrences of t in σ.

Example 2.3 *For the Petri net in Figure 2.6, the Parikh vector of the sequence $\sigma_1 = t_1t_3t_1t_2t_1t_3$ is $\vec{\sigma_1} = (3,1,2,0)$, while the Parikh vector of the sequence $\sigma_2 = t_4$ is $\vec{\sigma_2} = (0,0,0,1)$. Now suppose that the sequence $\sigma = \sigma_1\sigma_2 = t_1t_3t_1t_2t_1t_3t_4$, then the Parikh vector of σ is $\vec{\sigma} = \vec{\sigma_1} + \vec{\sigma_2} = (3,1,2,0) + (0,0,0,1) = (3,1,2,1)$.*

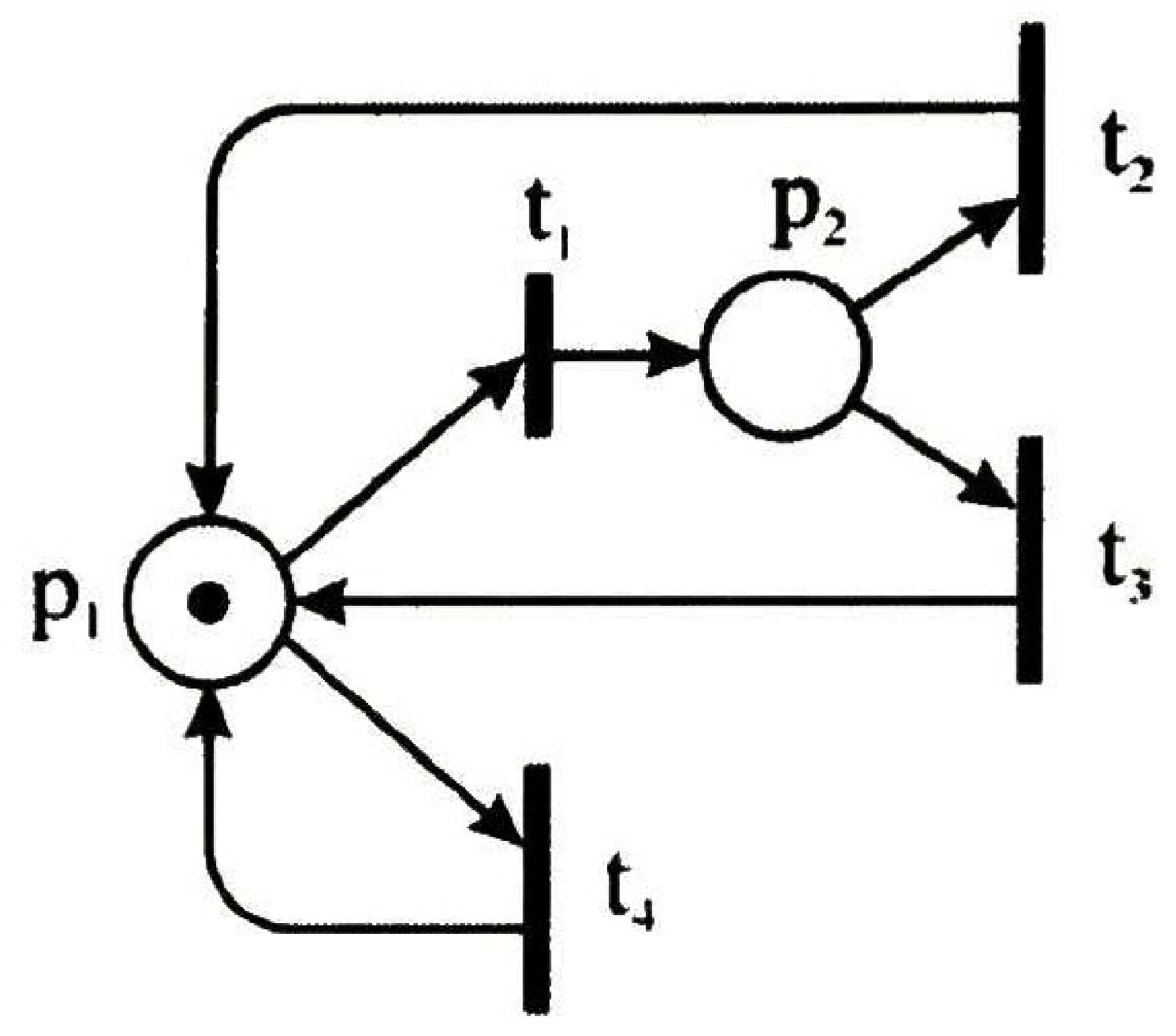

Figure 2.6:

If $M_0 \xrightarrow{\sigma} M$, then it can be written in the vector form as $M = M_0 + \mathbf{N} \quad \vec{\sigma}$, which is referred as the *state equation* of the net. Now, observe that for every transition t, the fired vector $\mathbf{t} = \mathbf{N} \cdot \vec{t}$; is the column of $\mathbf{N}$ corresponding to transition t multiplied by the number of occurrences of t in σ.

A reachable marking M_n can be computed from a marking M_0 using the state equation and the Parikh vector of the firable sequence $\sigma = t_1t_2t_3...t_n$ such that $M_0 \xrightarrow{\sigma} M_n$

$$\begin{aligned} M_1 &= M_0 + \mathbf{N} \cdot \vec{t_1} \\ M_2 &= M_1 + \mathbf{N} \cdot \vec{t_2} = M_0 + \mathbf{N} \cdot (\vec{t_1} + \vec{t_2}) \\ M_3 &= M_2 + \mathbf{N} \cdot \vec{t_3} = M_0 + \mathbf{N} \cdot (\vec{t_1} + \vec{t_2} + \vec{t_3}) \\ &\vdots \\ M_n &= M_{n-1} + \mathbf{N} \cdot \vec{t_n} = M_0 + \mathbf{N} \cdot (\vec{t_1} + \vec{t_2} + \vec{t_3} + ... + \vec{t_n}) \end{aligned}$$

Therefore, the resulting expression

$$M_n = M_0 + \mathbf{N} \cdot \vec{\sigma} \tag{2.2}$$

is called the *state equation* of the Petri net.

2.2.3 Invariants

An invariant of a dynamic system is an assertion that holds for every reachable state. For DEDS's modelled by Petri nets, it is possible to compute certain vectors of rational numbers (directly from the structure) which induce invariants. Below are shown two techniques of invariants analysis based on the determination of valid relationships independent of the net evolution.

Place Invariants

Given an arbitrary Petri net, it is difficult to characterize all the vectors Y such that $Y \quad M$ remains constant for every reachable marking M. However, it is easy to derive a sufficient condition from the marking equation.

Consider the Petri net of Figure 2.7. It is easy to see that for every reachable marking M the equation $M(p_2) + M(p_3) = 1$ holds, i.e., it is an invariant of the system. This equation can be rewritten as

$$\begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} M(p_1) \\ M(p_2) \\ M(p_3) \end{bmatrix} = 1 \text{ or just } \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \; M = 1$$

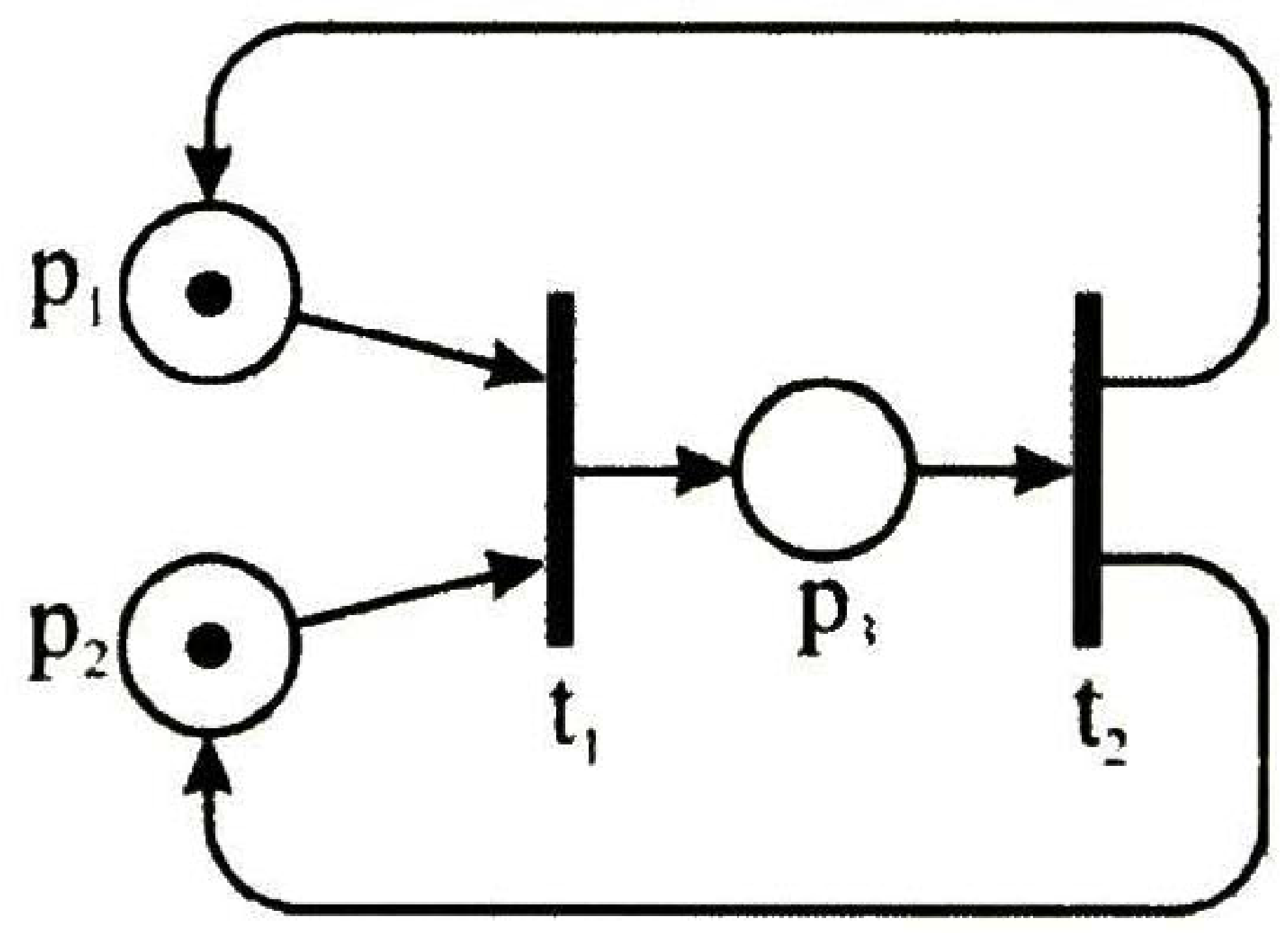

Figure 2.7: The vector $Y = (0, 1, 1)$ is an P-semiflow.

Definition 2.9 *A P-semiflow Y of a net N is a nonnegative rational-valued solution Y of the equation*

$$Y \cdot \mathbf{N} = 0 \tag{2.3}$$

Transition Invariants

T-semiflows of a Petri net N are vectors X satisfying $\mathbf{N} \cdot X = 0$. It seems natural to study if T-semiflows also have interesting properties. It will be shown that T-semiflows are related to the occurrence of sequences which reproduce a marking (cyclicity property), i.e., those that lead from a marking to itself.

Definition 2.10 *A T-semiflow X of a Petri net N is a nonnegative rational-valued solution of the equation*

$$\mathbf{N} \cdot X = 0 \tag{2.4}$$

The set of T-semiflows of a Petri net constitutes again a vector space over the field of rational numbers.

2.2.4 Representational Power

Concurrence, decision making, and synchronization, are some DEDS's characteristics, and they can be effectively modelled by Petri nets. Figure 2.8 shows some Petri nets structures representing these DEDS's main characteristics.

1. *Sequential Execution.* In Figure 2.8 a), transition t_2 can fire only after the firing of t_1. Imposing the precedence constraint "t_2 after t_1" This Petri net models the precedence relationship among tasks.
2. *Conflict.* Set of transitions having the same predecessors. In Figure 2.8 b), transitions t_1 and t_2 are in conflict. Both are enabled but the firing of any transition leads to the disabling of the other transition. Such a situation will arise, for example, when two tasks have the same precedence relationship.
3. Concurrence. In Figure 2.8 c), the transitions t_1, and t_2 are concurrent. Note that a necessary condition for transitions to be concurrent is the existence of a forking transition that deposits a token in two or more output places. This situation arises when two tasks can be achieved at the same time.
4. *Synchronization.* On a DEDS, often the precedence relationship defines that a later task can be achieved only after two or more previous tasks are completed. The resulting synchronization of activities can be captured by transitions of the type shown in Figure 2.8 d). Here, t_1 is enabled only when each of p_1 and p_2 receives a token. Essentially, transition t_1 models the joining operations.
5. *Merging.* When tasks from several streams are assigned to the same resource, the resulting situation can be depicted as in Figure 2.8 e).
6. *Confusion.* It is a situation where concurrence and conflicts coexist. An example is depicted in Figure 2.8 f). Both t_1 and t_3 are concurrent while t_1 and t_2 are in conflict, and t_2 and t_3 are also in conflict.
7. *Mutually Exclusive.* Two tasks are mutually exclusive if they cannot be performed at the same time due to constraints on the usage of shared resources. Figure 2.8 g) shows this structure. For example, a robot may be shared by two machines for loading and unloading.

2.3 Subclasses

2.3.1 Marked Graphs

A Marked Graph is a Petri Net whose places have exactly one input transition and exactly one output transition.

Definition 2.11 *A Petri Net $N = (P, T, Pre, Post)$ is a Marked Graph if and only if* $\forall p \in P : |\bullet p| = |p \bullet| = 1$.

In Marked Graphs synchronizations are allowed while decisions are not. A system is a decision-free system if its Petri net model is a marked graph. In a marked graph, tokens at a given place are generated by a predefined transition (its only input transition) and consumed by a predefined transition (its only output transition).

Example 2.4 *Figure 2.9 shows a marked graph.*

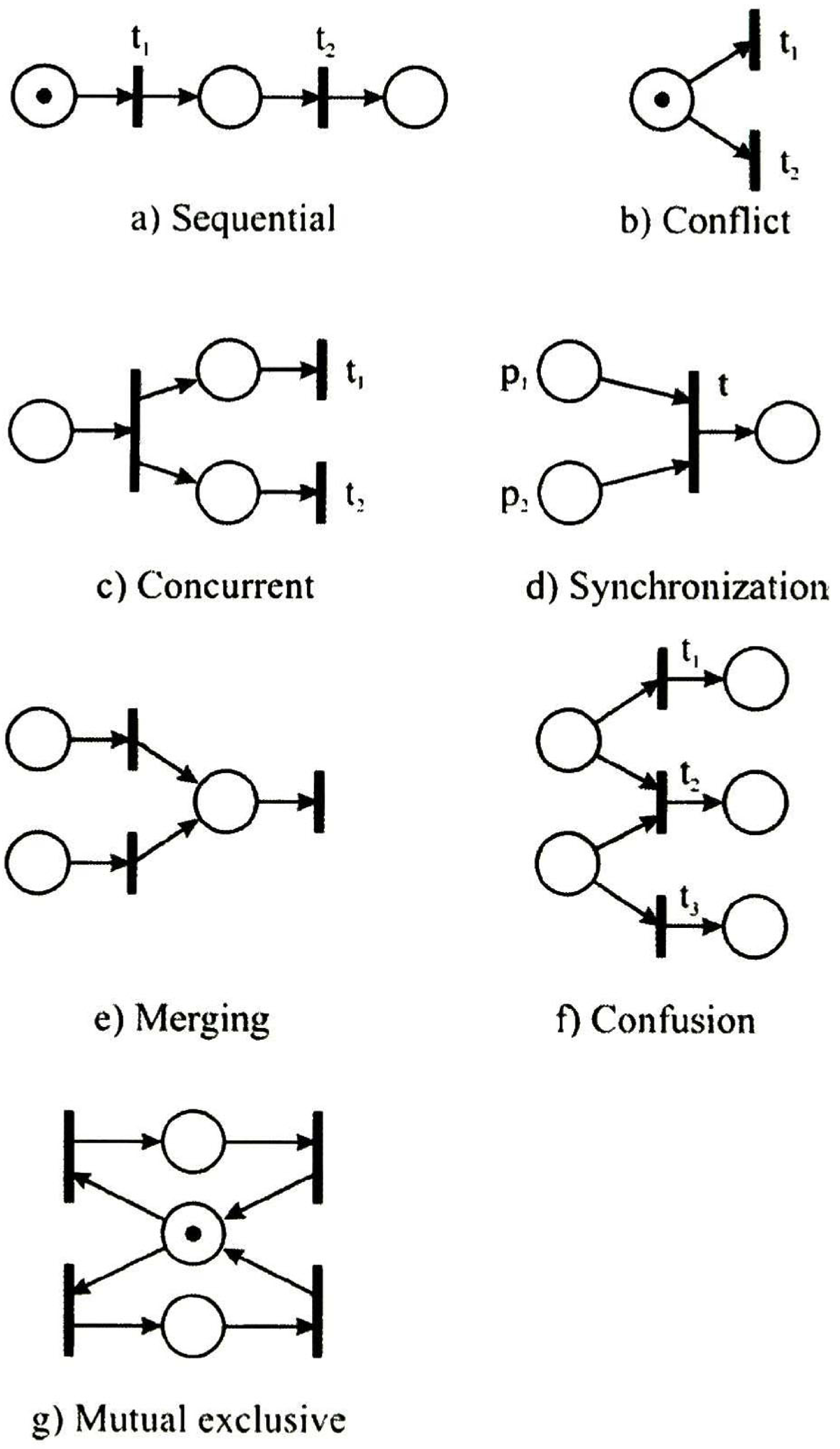

Figure 2.8: Petri nets primitives representing DEDS's main characteristics.

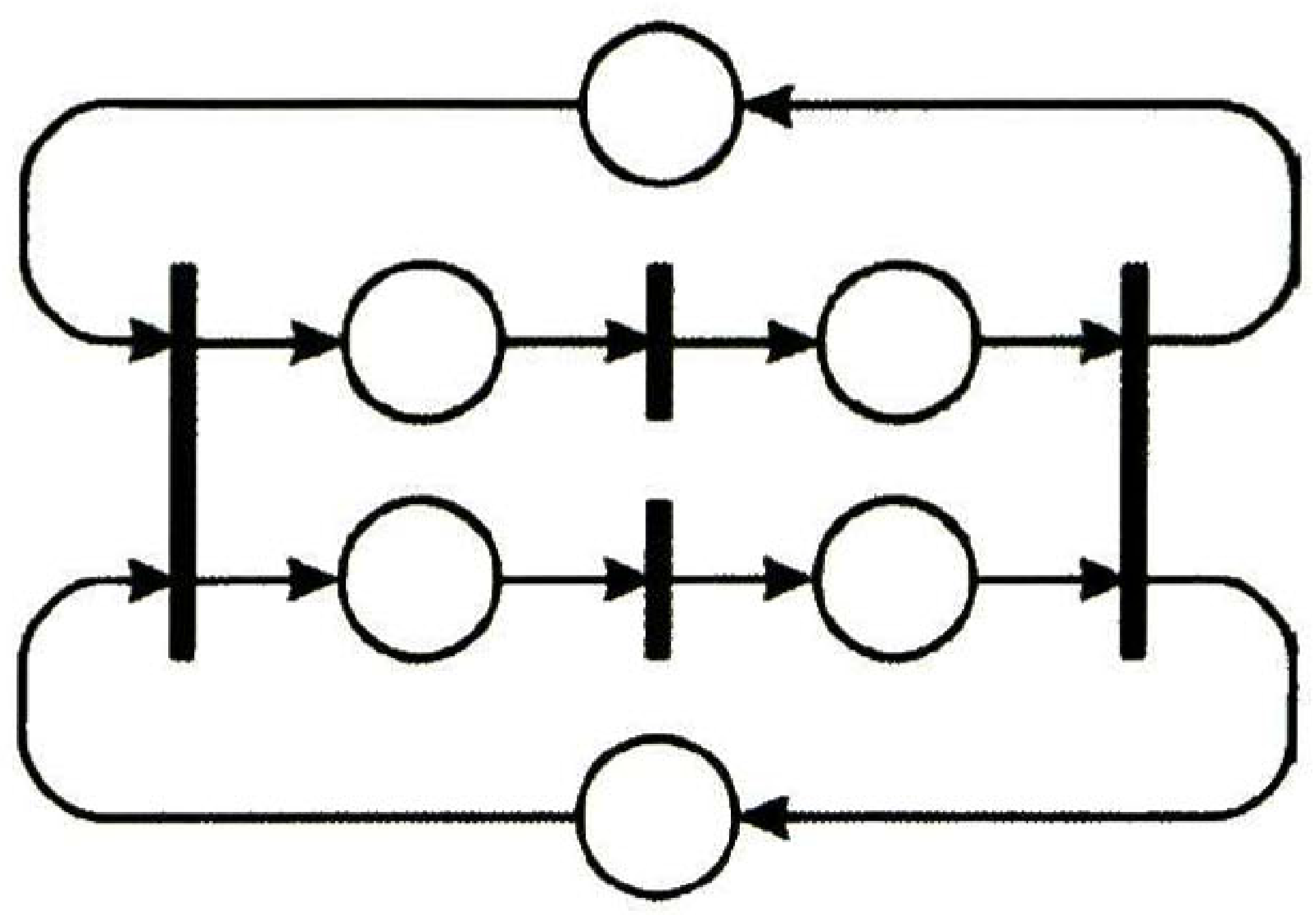

Figure 2.9: A Marked Graph

2.3.2 State Machines

A State Machine is a Petri Net whose transitions have exactly one input place and exactly one output place.

Definition 2.12 *A Petri Net $N = (P, T, Pre, Post)$ is a State Machine if and only if* $\forall t \in T : |\bullet t| = |t \bullet| = 1$

In State Machines decisions are allowed while synchronizations are not. The fundamental property of state machines is that they are strictly conservative. This subclass has been named State Machines due to its similarity with State Machines in the sense of automata theory.

Example 2.5 *Figure 2.10 shows a state machine.*

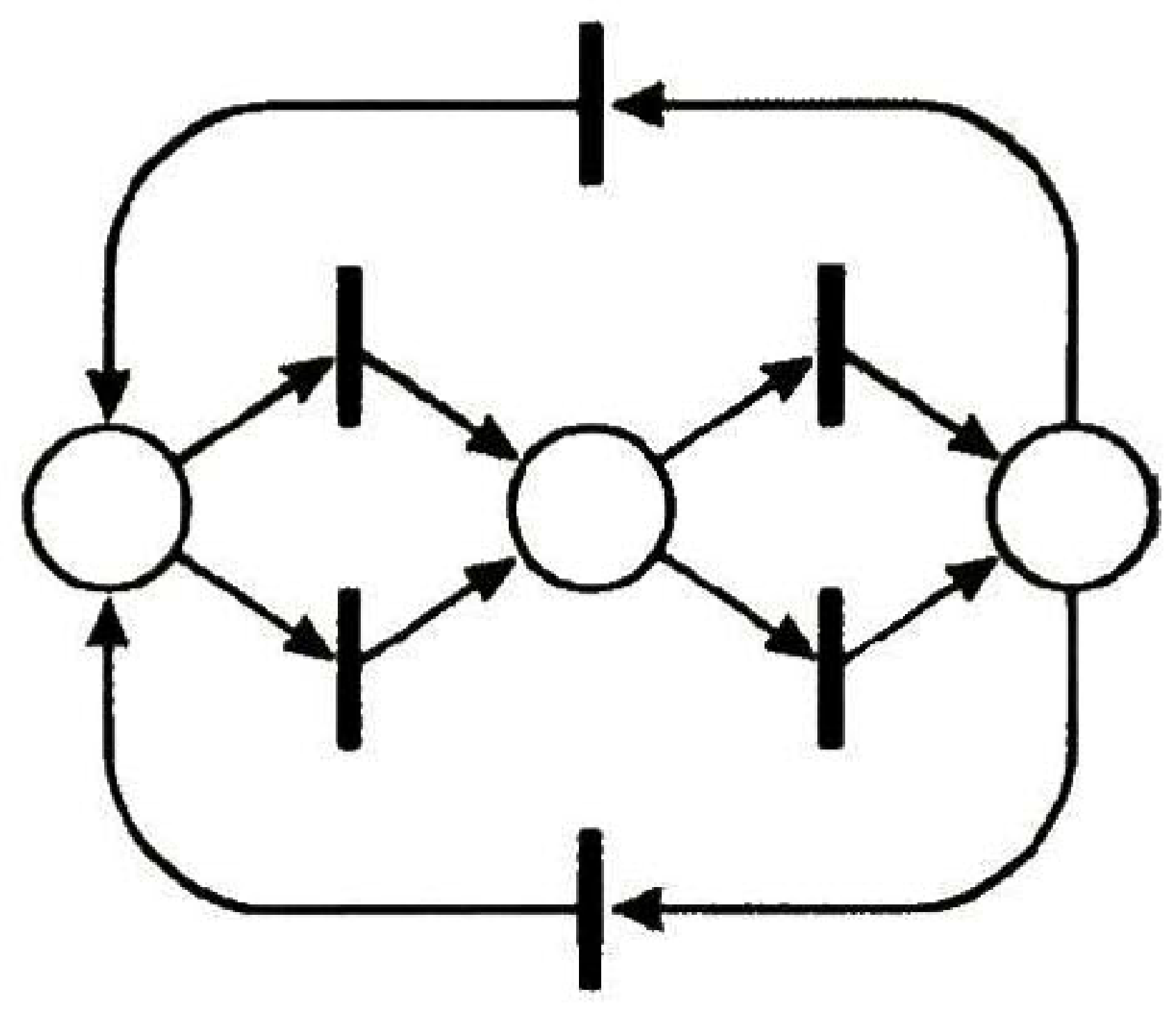

Figure 2.10: A strongly connected state machine.

2.3.3 Free-Choice

Marked Graphs allow only synchronizations while State Machines allow only decisions. Free-Choice is a subclass including both synchronization and decision, but in such a manner that they do not coexist.

Definition 2.13 *A Petri Net $N = (P, T, Pre, Post)$ is a Free Choice Petri Net if and only if* $\forall p \in P : |p \bullet| > 1 \Rightarrow \bullet(p\bullet) = \{p\}$

Free Choice subclass is defined to rule the following situations out: in them, the result of the choice between two transitions can never be influenced by the rest of the system. The easiest way to enforce this is to keep places with more than one output transition apart from transitions with more than one input place.

Example 2.6 *Figure 2.11 shows a free-choice Petri net.*

2.4 Timed Petri Nets

Time has been introduced in different forms into Petri Nets, the most common is associating time to transitions, because transitions represent the actions of the system.

Timed Petri nets can be divided into two classes: *Deterministic Timed Petri Nets* (DTPN's), in which each transition, place or directed arc is associated with deterministic firing time or time interval; and *Stochastic*

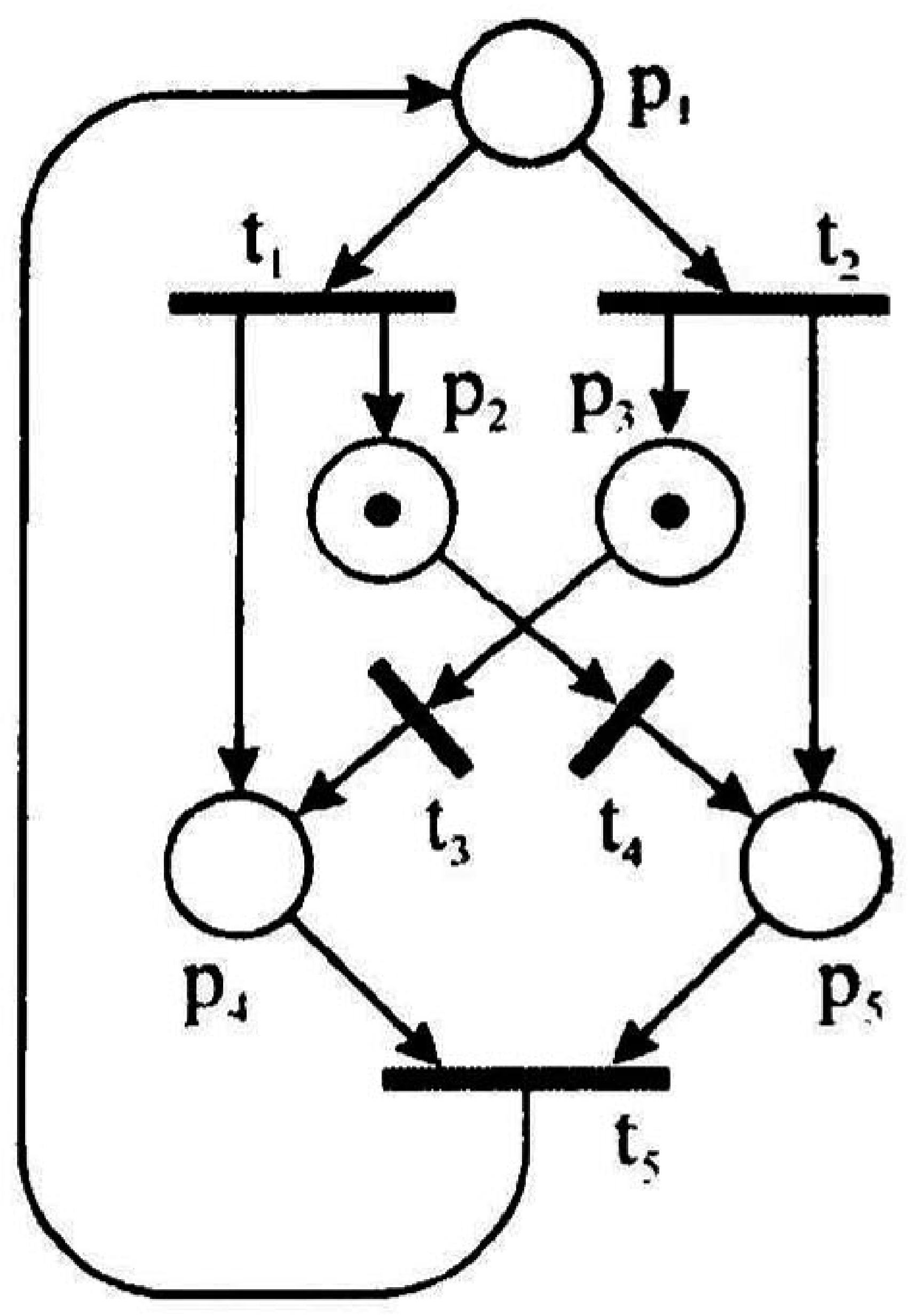

Figure 2.11: A free choice Petri net.

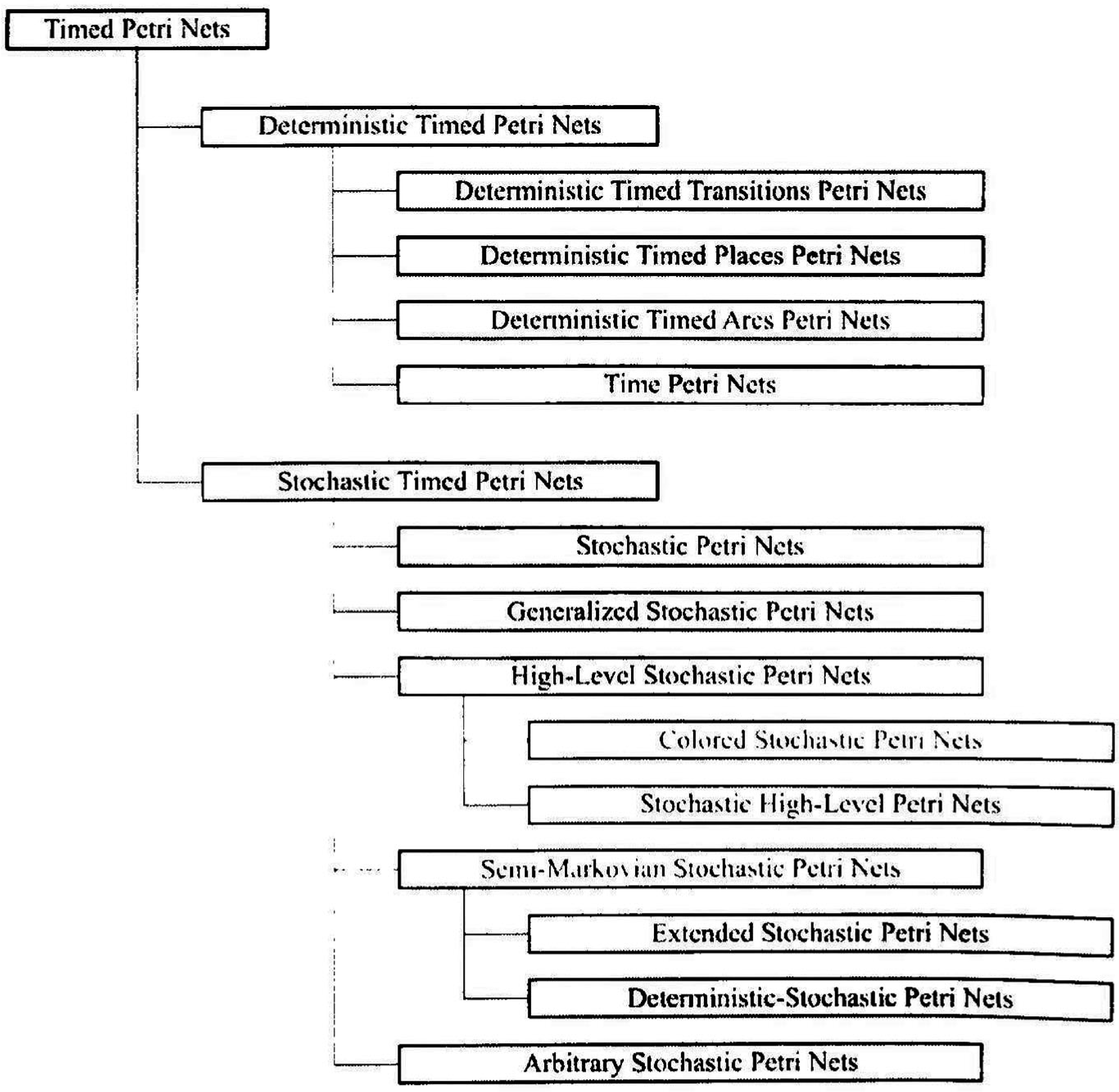
Timed Petri Nets
Deterministic Timed Petri Nets
Deterministic Timed Transitions Petri Nets
Deterministic Timed Places Petri Nets
Deterministic Timed Arcs Petri Nets
Time Petri Nets
Stochastic Timed Petri Nets
Stochastic Petri Nets
Generalized Stochastic Petri Nets
High-Level Stochastic Petri Nets
Colored Stochastic Petri Nets
Stochastic High-Level Petri Nets
Semi-Markovian Stochastic Petri Nets
Extended Stochastic Petri Nets
Deterministic-Stochastic Petri Nets
Arbitrary Stochastic Petri Nets

Figure 2.12: Classification of timed Petri nets.

Timed Petri Nets (STPN's), in which only transitions are associated with random firing times. Figure 2.12 shows a classification of timed Petri nets.

The introduction of the deterministic time function into Petri Nets was first attempted by Ramchandani [Ram74]. In his approach, labels were associated to each transition, denoting the fact that transitions are often used to represent actions, and actions take time to be achieved. The obtained extended Petri nets are called *Deterministic Timed Transitions Petri Nets.*

A different time function was proposed by Coolahan and Roussopoulos [CR83]. In their approach time labels are assigned to places in a Petri net, recognizing the fact that places in their models are mostly used to represent processes that consume time. The obtained extended Petri nets are called *Deterministic Timed Places Petri Nets.*

2.4.1 Deterministic Timed Transitions Petri Nets

Associating time to transitions indicates:

- A firing delay (time elapsed between enabling and firing a transition) [Ram74] [RH80]. The firing of a transition t_i holds $Pre(p_j, t_i)$ tokens into the input places p_j at the time interval $[x - d_i, x)$, it removes $Pre(p_j, t_i)$ tokens from the input places p_j at the time x, and it puts $Post(t_i, p_k)$ tokens into the output places p_k at the time x. See figure 2.13.

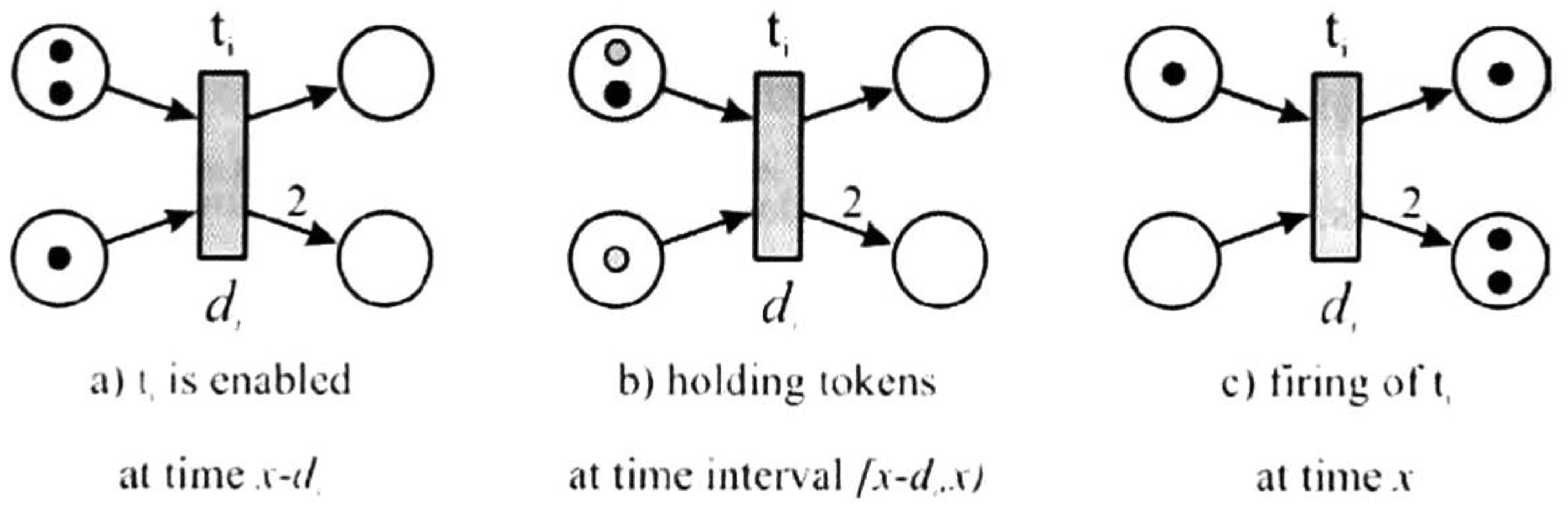

Figure 2.13: Time as a firing delay.

- A firing time. The firing of a transition t_i removes $Pre(p_j, t_i)$ tokens from the input places p_j at the time x and puts $Post(t_i, p_k)$ tokens into the output places p_k at the time $x + d_i$. See figure 2.14.

Definition 2.14 *A deterministic timed transition Petri net (DTTPN) is the 6-tuple $N = (P, T, Pre, Post, M_0, D)$, where:*

$(P, T, Pre, Post)$: is a Petri net
M_0: is an initial marking.
D: $T \rightarrow R^+$ is the function that associates transitions with deterministic time delays.

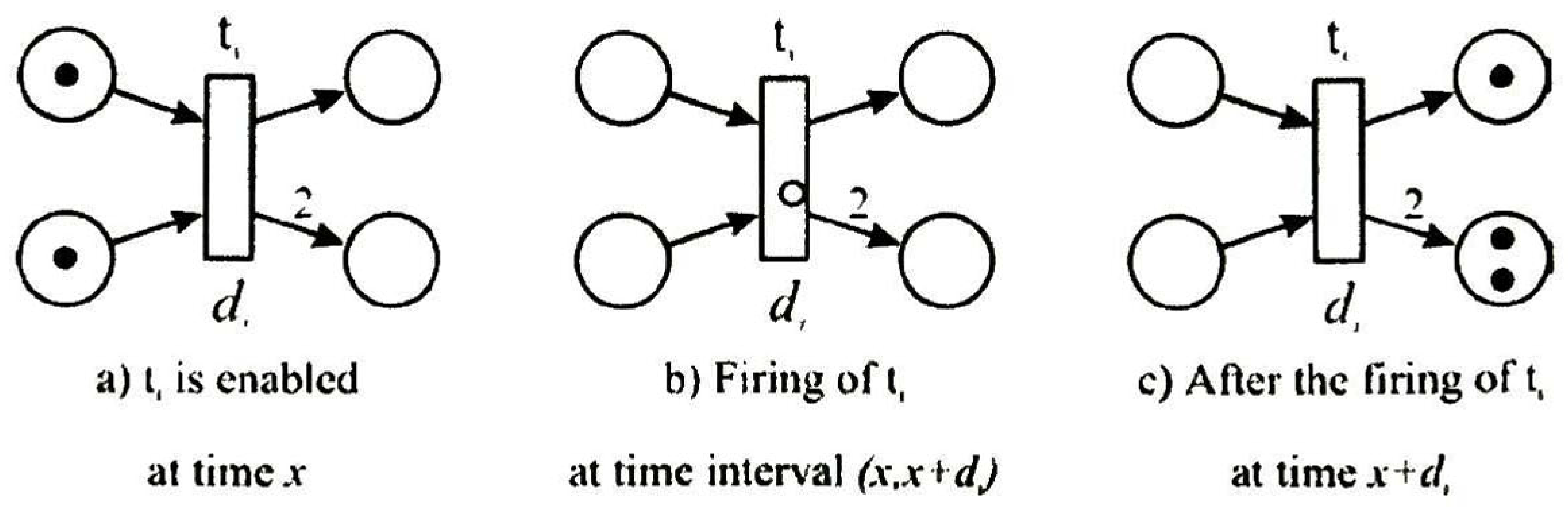

Figure 2.14: Time as a firing time.

For a DTTPN, the transition firing rules for ordinary Petri nets are extended to include the following one: A transition t_i in a DTTPN can fire at time x if and only if

1. for any input place p_j of this transition, there must be a number of tokens equal or greater than the weight of the directed arc connecting p_j to t_i continuously during the time interval $[x - d_i, x)$; where d_i is the associated firing time of transition t_i;
2. for any input place connected by an inhibitor arc of this transition, no token must reside in the input place continuously during the time interval $[x - d_i, x)$; and
3. at time x, each of its input places p_j donates a quantity of tokens equal to the weight of the directed arc connecting p_j to t_i, while each of its output places p_k receives a quantity of tokens equal to the weight of the directed arc connecting t_i to p_k.

2.4.2 Deterministic Timed Places Petri Nets

In place timed Petri nets, time is associated to places and indicates the time that a token must remain in a place before it can enable a transition.

Example 2.7 *A timed transition is equivalent to two transitions and a timed place, see Figure 2.15 a); the firing duration of transition t_i is represented by the time the token remains no available into the place p_{ti}. Analogously, a timed place is equivalent to two places and a timed transition, see Figure 2.15 b); the delay associated to p_j is represented by the firing duration associated to t_{pj}.*

2.5 Scheduling in Petri Nets

2.5.1 Marked Graphs

Optimal Cycle Time

For a marked graph (decision-free system), the maximum performance can be computed quite easily [RH80].

Theorem 2.1 *[RH80] For a decision-free Petri net, the number of tokens in a circuit remains constant after any firing sequence.*

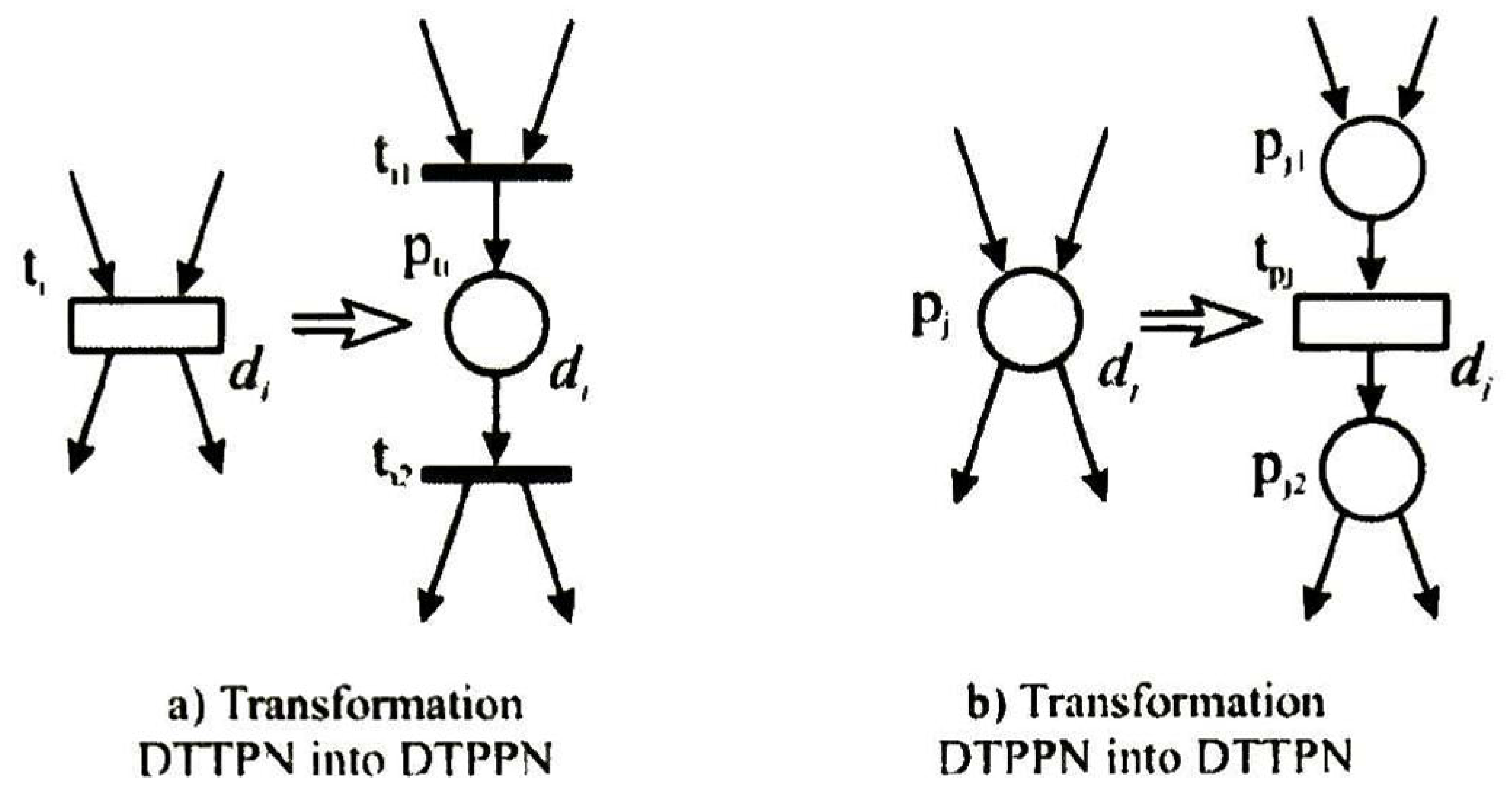

Figure 2.15:

Theorem 2.2 *[RH80] All transitions in a decision-free Petri net have the same cycle time.*

For marked graphs, the minimum cycle time (maximum performance) is equal to the cycle time of the slowest p-semiflow [RH80], and can be computed in polynomial time by solving the following linear equations system using the simplex method [Mag84]:

$$\pi = \max_{Y} \quad X \cdot \mathbf{N}^{-} D$$
$$\text{s.t:}$$
$$X \ \ \mathbf{N} = 0$$
$$X \cdot M_0 = 1$$
$$X \geq 0$$

where D is the vector containing the firing delays of transitions. Note that this method does not need to enumerate all circuits to find out the optimal cycle time.

Optimal Schedule

Note, however that knowing the optimal cycle time does not mean knowing the optimal schedule. Below an example illustrates this assertion.

Example 2.8 *Assume that transitions t_1, t_2, t_3, t_4 of figure 2.2 have a delay of 0, 3, 1, 1 respectively. The optimal cycle time of the net is 4 (time associated to the circuit formed by transitions $t1, t_2, t_4$ and the respective input and output places). There exist an infinite number of schedules whose cycle time is equal to the optimal cycle time [RH80].*

Fortunately, for marked graphs there is no need to solve linear equations systems, the following rule obtains the optimal schedule.

Theorem 2.3 *[Car84], [Car88] "Firing transitions as soon as become enabled" yields to an optimal schedule.*

The application of this rule is based on the fact that there is no decision in the net. Hence, the firing of a transition cannot yield to a wrong decision; the firing of transitions as soon as they become enabled avoids delays in the slowest circuit.

2.5.2 State Machines

Opposite to marked graphs, state machines have decisions instead synchronizations. The analysis performance of state machines is achieved through the execution of T-invariants. More emphasis will be paid on state machines since this subclass is the cornerstone of the next chapter.

Optimal Cycle Time

The computation of the optimal cycle time can be achieved in polynomial time solving the following linear programming problem [Cam90]

$$\begin{aligned} \pi = \max_{Y} \quad & X \cdot \mathbf{N}^{-} D \\ \text{s.t:} & \\ X \quad & \mathbf{N} = 0 \\ X \cdot M_0 & =1 \\ X & \geq 0 \end{aligned}$$

The previous problem searches the slowest P-semiflow. The solution of the previous problem is the optimal cycle time since a state machine has only one P-semiflow, there is no synchronization in the net, and every marking belongs to the same and unique home marking (every state is reachable).

Since there exist no synchronization in the net, there exist no operation involving the maximum between elements; there exist only transitions' time additions. Hence, the optimal cycle time is a weighted addition of the transitions' delay.

Optimal Schedule Computation Algorithm

The equation $\mathbf{N} \cdot X = \mathbf{0}$ provides a set of T-semiflows, and a basis of minimal T-semiflows.

Theorem 2.4 *[TV84] Every strongly connected state machine is covered by T-invariants.*

State machines have the following characteristics:

1. A positive basis of T-semiflows can be found for a state machine [Sil85].
2. The optimal cycle time, of the executions satisfying the visit ratio constraints, can be computed in polynomial time [Cam90].
3. The visit ratio is a nonnegative linear combination of a basis β_T of elemental T-semiflows [Ram93].

Example 2.9 *Figure 2.16 shows a state machine whose T-semiflows are:*
$\langle \mathcal{T}_1 \rangle = \{t_1, t_4, t_5\}$,
$\langle \mathcal{T}_2 \rangle = \{t_1, t_3, t_{10}, t_{17}, t_{12}, t_5\}$,
$\langle \mathcal{T}_3 \rangle = \{t_{11}, t_{17}, t_{12}\}$,
$\langle \mathcal{T}_4 \rangle = \{t_6, t_{13}, t_{12}, t_5\}$,

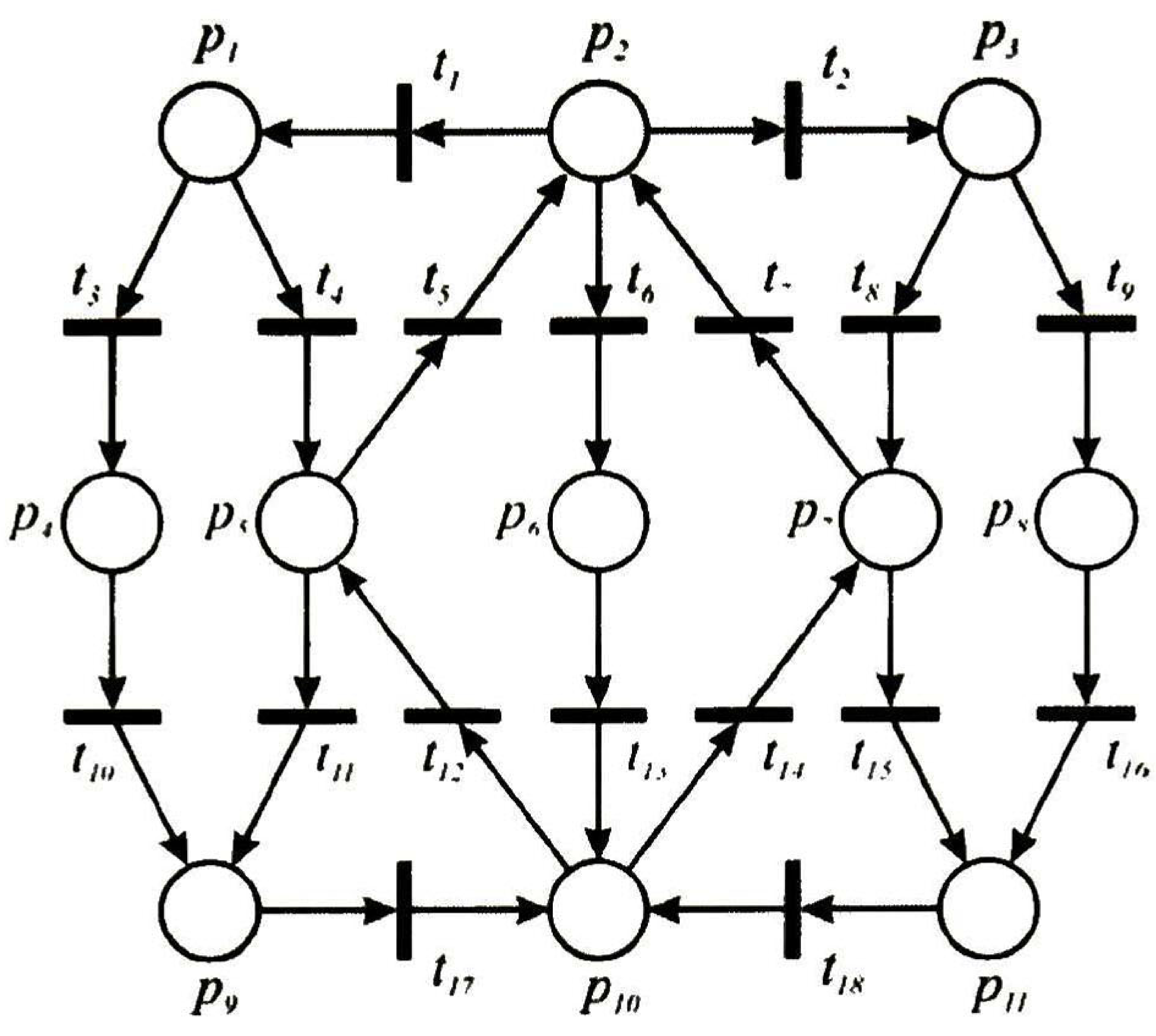

Figure 2.16: Example of a state machine.

$\langle \mathcal{T}_5 \rangle = \{t_2, t_8, t_7\}$,
$\langle \mathcal{T}_6 \rangle = \{t_2, t_9, t_{16}, t_{18}, t_{14}, t_7\}$,
$\langle \mathcal{T}_7 \rangle = \{t_{15}, t_{18}, t_{14}\}$,
$\langle \mathcal{T}_8 \rangle = \{t_6, t_{13}, t_{14}, t_7\}$,

and, the conflicts are:
$C_1 = \{t_3, t_4\}$,
$C_2 = \{t_1, t_2, t_6\}$,
$C_3 = \{t_8, t_9\}$,
$C_4 = \{t_5, t_{11}\}$,
$C_5 = \{t_7, t_{15}\}$,
$C_6 = \{t_{12}, t_{14}\}$.

then, the computed visit ratio for this state machine is:
$v_r = [2, 2, 1, 1, 3, 2, 3, 1, 1, 1, 3, 5, 2, 5, 3, 1, 4, 4]$.

Appling the algorithm proposed by Ramírez [Ram93], the visit ratio is decomposed into the following T-semiflows
$v_r = \mathcal{T}_1 + \mathcal{T}_2 + 3\mathcal{T}_3 + \mathcal{T}_4 + \mathcal{T}_5 + \mathcal{T}_6 + 3\mathcal{T}_7 + \mathcal{T}_8$

Proposition 2.1 *Firing the transitions of a T-invariant as soon as they become enabled; and executing α_i times every T-invariant can obtain an optimal schedule. If a new T-invariant becomes enabled, then its execution must start. Once a T-invariant has been executed α_i times, then it must not be considered again in the actual cycle; the execution continues with the remaining T-invariants.*

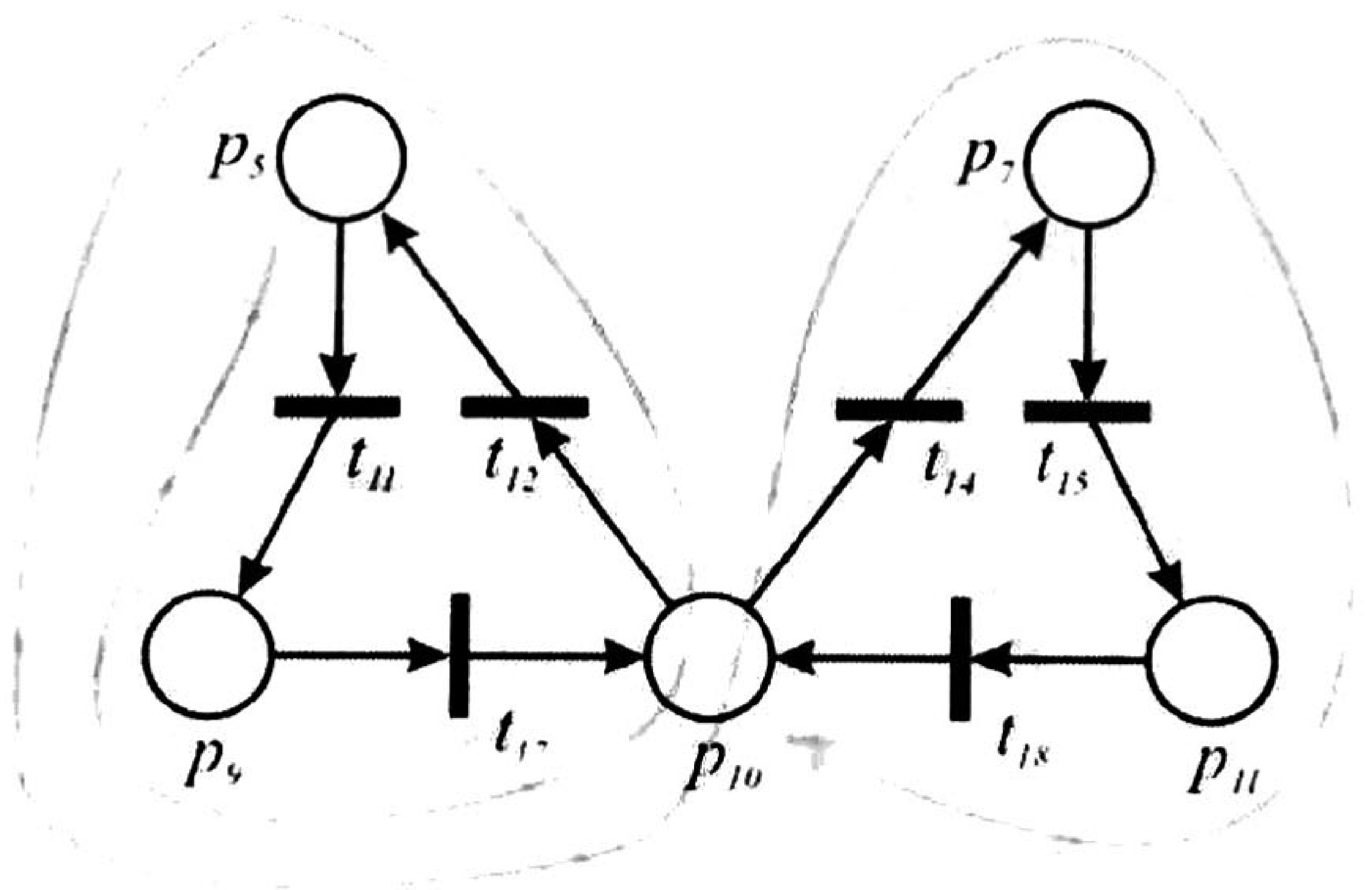

Figure 2.17: A schedule that does not fulfil the visit ratio.

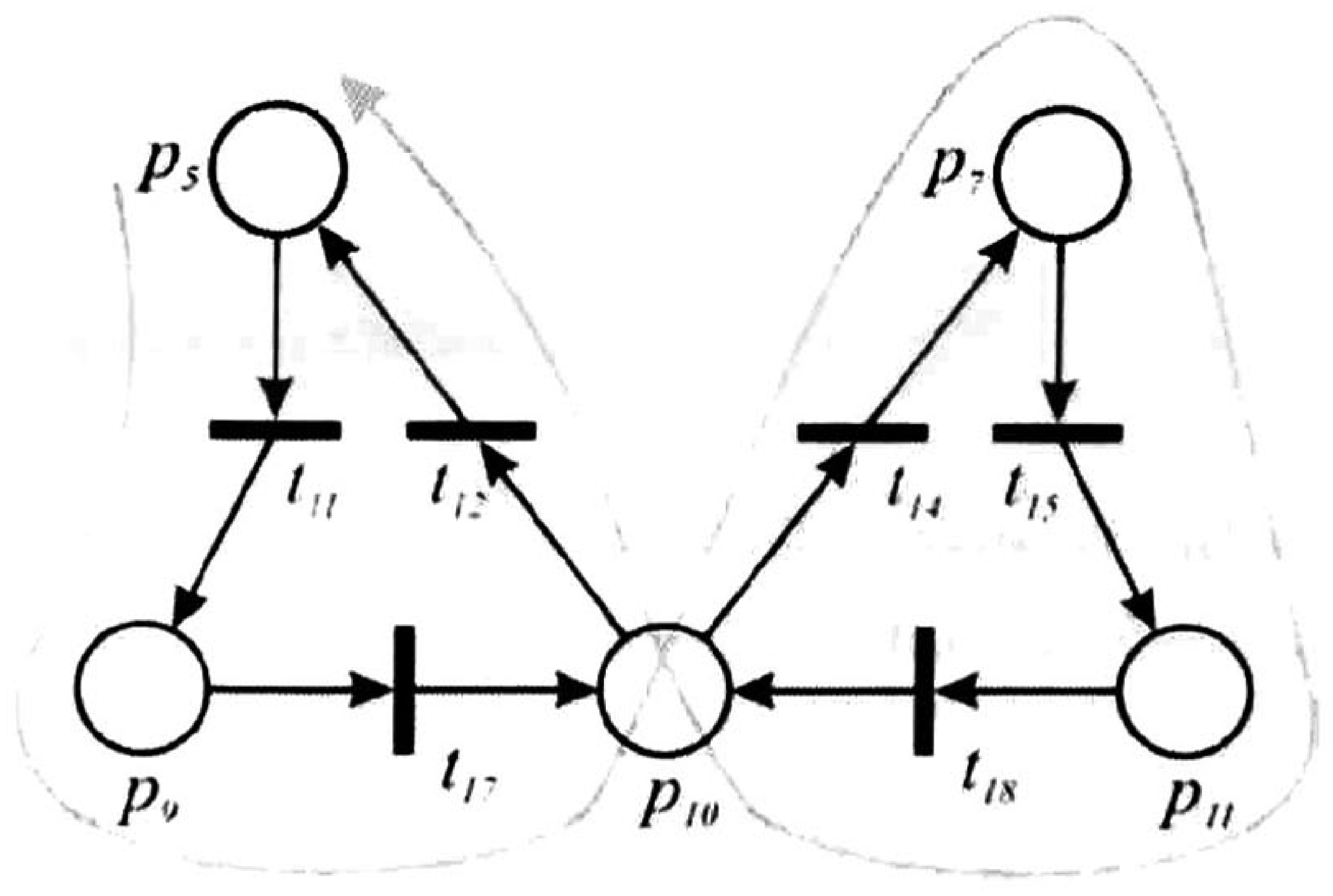

Figure 2.18: A schedule that fulfils the visit ratio.

Example 2.10 *For the state machine of Figure 2.16, with just one token into place p_5, the proposed algorithm works as follows:*

1. $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$ and $\mathcal{T}_4$ are enabled.
2. It begins to execute $\mathcal{T}_3$ 3 times and $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_4$ are pushed into the stack. During the first execution of $\mathcal{T}_3$, place p_{10} enables $\mathcal{T}_6$, $\mathcal{T}_7$, and $\mathcal{T}_8$.
3. $\mathcal{T}_3$ is pushed into the stack and it is not taken into account.
4. Anyone of recently enabled T-invariants is executed and the others are pushed into the stack. For example, $\mathcal{T}_7$ is selected and executed, $\mathcal{T}_6$, $\mathcal{T}_8$ are pushed into the stack. Since no other T-invariant becomes enabled during the execution of $\mathcal{T}_7$, then $\mathcal{T}_7$ is executed three times with no interruption. When $\mathcal{T}_7$ finishes its execution, it returns the token back into p_{10}.
5. Now $\mathcal{T}_6$ is popped from the stack and executed. When place p_2 becomes marked $\mathcal{T}_5$ is enabled. (At this moment $\mathcal{T}_5$ is the unique T-invariant, the remaining have been pushed into the stack).
6. $\mathcal{T}_6$ is pushed into the stack.
7. $\mathcal{T}_5$ is executed. When $\mathcal{T}_5$ finishes its execution, it returns the token back into p_2, and no T-invariant is enabled. Therefore, the popping of the stack begins.
8. $\mathcal{T}_6$ is popped from the stack, its execution is finished returning the token back into the place p_{10}.
9. $\mathcal{T}_8$ is popped from the stack and executed.
10. The three executions of $\mathcal{T}_3$ are finished with no interruption, returning the token back into the place p_5.
11. $\mathcal{T}_4$ is popped from the stack and executed, returning the token back into the place p_5.
12. $\mathcal{T}_1$ is popped from the stack and executed, returning the token back into the place p_5.
13. $\mathcal{T}_2$ is popped from the stack and executed, returning the token back into the place p_5.

The final sequence is:

$$\underbrace{t_{11}, t_{17}}_{\mathcal{T}_3}, \underbrace{t_{14}, t_{15}, t_{18}, t_{14}, t_{15}, t_{18}, t_{14}, t_{15}, t_{18}}_{3\mathcal{T}_7}, \underbrace{t_{14}}_{\mathcal{T}_6}, \underbrace{t_7, t_2, t_8}_{\mathcal{T}_5}, \underbrace{t_7, t_2, t_9, t_{16}, t_{18}}_{\mathcal{T}_6}, \underbrace{t_{14}, t_7, t_6, t_{13}}_{\mathcal{T}_8},$$
$$\underbrace{t_{12}, t_{11}, t_{17}, t_{12}, t_{11}, t_{17}, t_{12}}_{3\mathcal{T}_3}, \underbrace{t_5, t_6, t_{13}, t_{12}}_{\mathcal{T}_4}, \underbrace{t_5, t_1, t_4}_{\mathcal{T}_1}, \underbrace{t_5, t_1, t_3, t_{10}, t_{17}, t_{12}}_{\mathcal{T}_2}$$

The optimal cycle time is reached by firing transitions as soon as they become enabled in the order indicated by this sequence. Therefore, the obtained sequence is an optimal schedule.

The T-invariants depicted below transitions in the previous firing sequence correspond to the executed T-invariant. When it appears more than once is because T-invariants enable other T-invariants during their execution. The last time that a T-invariant appears in the firing sequence, it meets the number of times that such T-invariant occurs in the visit ratio.

Case for non-safe State Machines

Until now, the obtained method is valid only for safe (1-bounded) state machines. However, the method is easily extended to the non-safe state machines case.

The solution is the separately application of the previous algorithm to every token; resulting in a set of sequences, one sequence for each token such that all sequences can be executed in parallel. Every sequence in this set of sequences has the same period, since they have exactly the same transitions (but in different order).

The cycle time of a safe state machine is equal to the period of the found sequence. While, for non-safe state machines, the existence of more than one sequence redefines the cycle time to the time period of a sequence divided by the number of tokens in the net. For state machines and every bounded and cyclic Petri net, the schedule becomes k periodic due to such schedule is repeated each $k \cdot \pi^*$ time units, where π^* is the optimal cycle time. Note that the initial marking is repeated each cycle time.

Example 2.11 *Consider again the machine state shown in Figure 2.16. Suppose that both places p_5 and p_6 have a token each one, then two sequences are computed, one for the marking $M(p_5) = 1$ and other for $M(p_6) = 1$. The case when p_5 is marked was analysed in the case for safe state machines; for the case when p_6 is marked, the obtained sequence is:*

$$\underbrace{t_{13}}_{\mathcal{T}_4}, \underbrace{t_{12}}_{\mathcal{T}_3}, \underbrace{t_5}_{\mathcal{T}_1}, \underbrace{t_2, t_8, t_7}_{\mathcal{T}_5}, \underbrace{t_1, t_4}_{\mathcal{T}_1}, \underbrace{t_{11}, t_{17}, t_{12}, t_{11}, t_{17}, t_{12}, t_{11}, t_{17}}_{3\mathcal{T}_3}, \underbrace{t_{14}, t_{15}, t_{18}, t_{14}, t_{15}, t_{18}, t_{14}, t_{15}, t_{18}}_{3\mathcal{T}_7},$$
$$\underbrace{t_{14}, t_7, t_2, t_9, t_{16}, t_{18}}_{\mathcal{T}_6}, \underbrace{t_{12}, t_5, t_1, t_3, t_{10}, t_{17}}_{\mathcal{T}_2}, \underbrace{t_{12}, t_5, t_6}_{\mathcal{T}_4}, \underbrace{t_{13}, t_{14}, t_7, t_6}_{\mathcal{T}_8}$$

The optimal schedule is obtained executing both sequences concurrently; the unique constraint is to execute transitions as soon as they become enabled in the order indicated by their respective sequence.

2.5.3 Free Choice Petri Nets

Analysis Complexity

The computation complexity to solve the optimal schedule, and the optimal cycle time problems for free choice Petri nets increases substantially. Both problems, optimal schedule, and optimal cycle time computation are NP-complete [Mag87].

Theorem 2.5 *[Mag87] The minimum cycle time computation of a live and safe free choice Petri net is an NP-Complete problem.*

For the computation of a lower bound π_k of the free choice Petri net cycle time, the following linear programming problem will be used [Cam90].

$$\pi_k^* \geq \pi_k = \max(Y \quad \mathbf{N}^- \cdot diag(D) \cdot v_k) \tag{2.5}$$
$$s.t.$$

$$\begin{aligned} Y \cdot \mathbf{N} &= 0 \\ Y \cdot M_0 &= 1 \\ Y &\geq 0 \end{aligned}$$

where D is a matrix of dimension $|T| \times |T|$; the element $D[i,i] = D(i)$ (the time associated to transition t_i); and $D[i,j] = 0$ if $i \neq j$.

This problem searches for the slowest P-semiflow and takes it as a lower bound of the cycle time. When there are no decisions (marked graphs) nor synchronizations (state machines), this cycle time bound is a tight bound [RH80], [Cam90], but in the general case it is only a lower bound of the minimum cycle time.

A. Ramírez [Ram93] proposed a classification for free choice Petri net whose optimal schedule and optimal cycle time computation can be achieved in polynomial time performing local optimisations.

Free Choice A*

Definition 2.15 *A free choice Petri net is A* if it do not have selection places between fork-join couples in any elemental T-invariant.*

The characterization of a free choice A* can be achieved in polynomial time. Just finding out a basis of elemental T-invariant and proving that erasing a selection place of the original net, then such T-invariant becomes acyclic.

Since there exist only one T-semiflow in the subsystem (for a safe free choice A*), the optimal cycle time of the net can be computed as the weighted addition of the T-invariants' cycle time. Furthermore, the optimal schedule reaches the optimal cycle time by executing T-invariants as soon as they become enabled.

Free Choice B*

Definition 2.16 *A free choice Petri net B* is a safe and live free choice Petri net that has just one selection place between every fork-join couple.*

Since a token can enable different T-semiflows while different P-invariants of the previous enabled T-semiflow still enabled, then free choice B* Petri nets can have more than one T-semiflow working at the same time.

Let define Q_d as the set of T-invariants including the selection place and the fork-join couple, and let define $Q_{\bar{d}}$ as the set of T-invariants including the selection place and excluding the fork-join couple.

If the optimal cycle time of the net can be computed as the weighted addition of the T-invariants belonging to Q_d, then the execution of the T-invariants belonging to $Q_{\bar{d}}$ is carried out in the looseness times

If p_s is a selection place between a fork-join couple and $|p_s \bullet| \geq 1$, then this is the same case that Magott proved been equivalent to the bin packing problem. The heuristic proposed by [Ram93] finds out a good schedule, but in the general case the problem remains NP Complete.

Free Choice C*

Definition 2.17 *Let $N = (P, T, Pre, Post)$ be a free choice Petri net with just a unique fork-join couple $[t_k, t_h]$, and let P_s be the set of selection places in the net. N is a free choice C^* if and only if:*

- t_k and t_h belongs to every elemental T-invariant of the net, and
- $\forall p_i, p_j \in P_s$, p_i and p_j do no belong to the same P-semiflow of a given T-invariant.

This is a class of free choice Petri nets which the optimal schedule can be found in polynomial time applying the following rule.

Rule: For every selection place, find out the slowest branch; the slowest branches of every selection place are executed in parallel, and they are not taken in future executions. The process is repeated until every transition has been fired according to the visit ratio.

The optimal schedule for this subclass can be found in polynomial time. A disadvantage of this subclass is that it can only model a narrow variety of DEDS.

2.6 Conclusions

Deterministic timed Petri nets can be used to compute production cycle time, identify bottlenecks, and verify timing constraints. The use of DTTPN or DTPPN is indistinct: always it is possible to transform a DTTPN model into a DTPPN model and vice versa. Therefore, the scheduling analysis will be achieved using DTTPN.

The scheduling problem for an arbitrary free choice Petri net is an NP-complete problem, but this subclass include more subclasses such that their schedule is not an NP-complete problem. Therefore, the study of the scheduling complexity on free choice Petri nets yields to develop heuristics and algorithms for specific subclasses of free choice Petri nets.

The analysis made for free choice Petri nets have been developed for systems that do not have T-semiflows working in parallel, excluding free choice B* and C* Petri nets. However, the analysis for Petri nets where there exist T-semiflows working in parallel becomes more complex due to the relationship between such T-semiflows working in parallel. Therefore, the scheduling problem for system capable to execute T-semiflows in parallel becomes a combinatorial problem.

Modelling DEDS with Petri nets solves the first and second phases of the scheduling problem (assigning resources to task and defining the precedence relationship amog tasks respectively).

Chapter 3

MODELING WITH SYNCHRONISED STATE MACHINES

This chapter defines a subclass of free choice Petri nets named Synchronised State Machines. This subclass belongs to the set of Petri nets subclasses which previous results proves that it is not possible to optimise using local optimisations.

It is shown that this subclass is capable to model a wide variety of DEDS; two detailed examples are provided to illustrate the application of this subclass.

3.1 Synchronised State Machines Petri nets

Definition 3.1 *Let $N = \{N_1, N_2, ...N_m\}$ be a set of state machines, and let $t_{i,n}$ be a transition belonging to the support of every T-semiflow on the state machine $N_n \in N$. A synchronised state machines Petri net is a Petri net obtained when all transitions $t_{i,n}$ are merged into a single one.*

If N_1 and N_2 are two state machines, then the resulting net of the merging function is a synchronised state machines Petri net. If N_1 is a state machine, N_2 is a synchronised state machines Petri net, and the merging function merges a transition $t_i \in N_1$ and the synchronisation transition $t_k \in N_2$, then the resulting net of the merging function is a synchronised state machines Petri net. If N_1 and N_2 are two synchronised state machines Petri nets, and the merging function merges the synchronization transitions of N_1 and N_2, then the resulting net of the merging function is also a synchronised state machines Petri net.

Example 3.1 *Let N_1, N_2 and N_3 be three state machines as show figure 3.1. Since transition $t_{i,5}$ belongs to the support of every elemental T-semiflow of the state machine i, then the state machines N_1, N_2 and N_3 can be merged by merging transitions $t_{1,5}$ $t_{2,5}$ $t_{3,5}$ into a single one. The merging of the state machines N_1 and N_2 through transitions $t_{1,5}$ and $t_{2,5}$ generates the synchronised state machines Petri net N_4 shown in figure 3.2. The merging of the state machine N_3 and the synchronised state machines Petri net N_4 through transitions $t_{3,5}$ and $t_{1,5\ 2,5}$ generates the synchronised state machines Petri net N_5 shown in figure 3.3.*

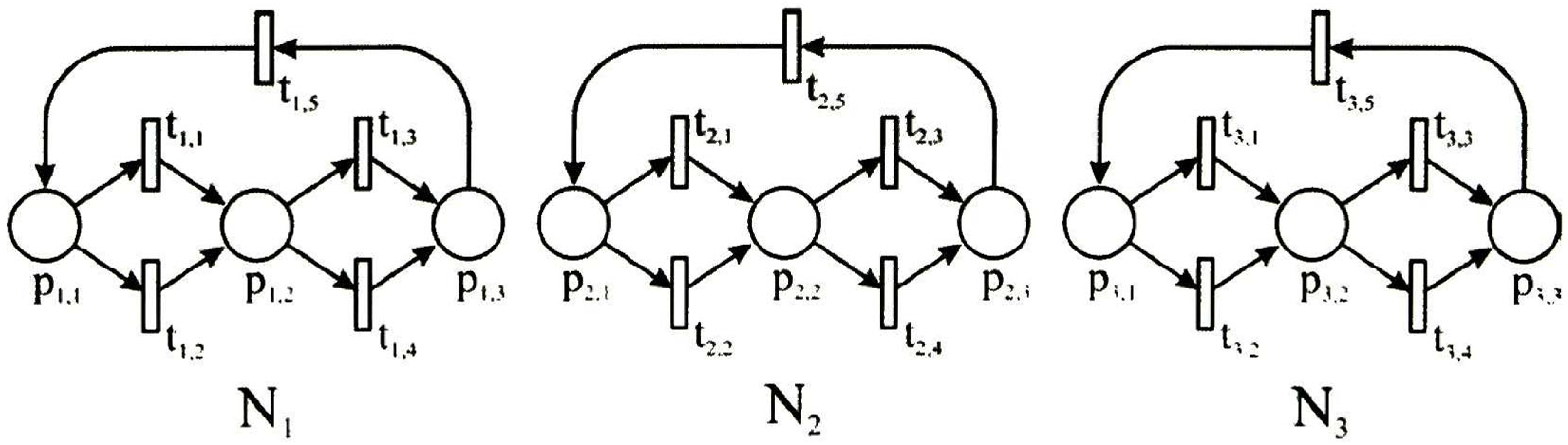

Figure 3.1: Three state machines

Synchronised State Machines Petri nets belongs to the set of Petri nets subclasses which [Ram93] proves that it is not possible to optimise using local optimisations, since the synchronisation transition represents the fork-join couple and there exist more than one selection place in every P-semiflow inside the fork-join couple.

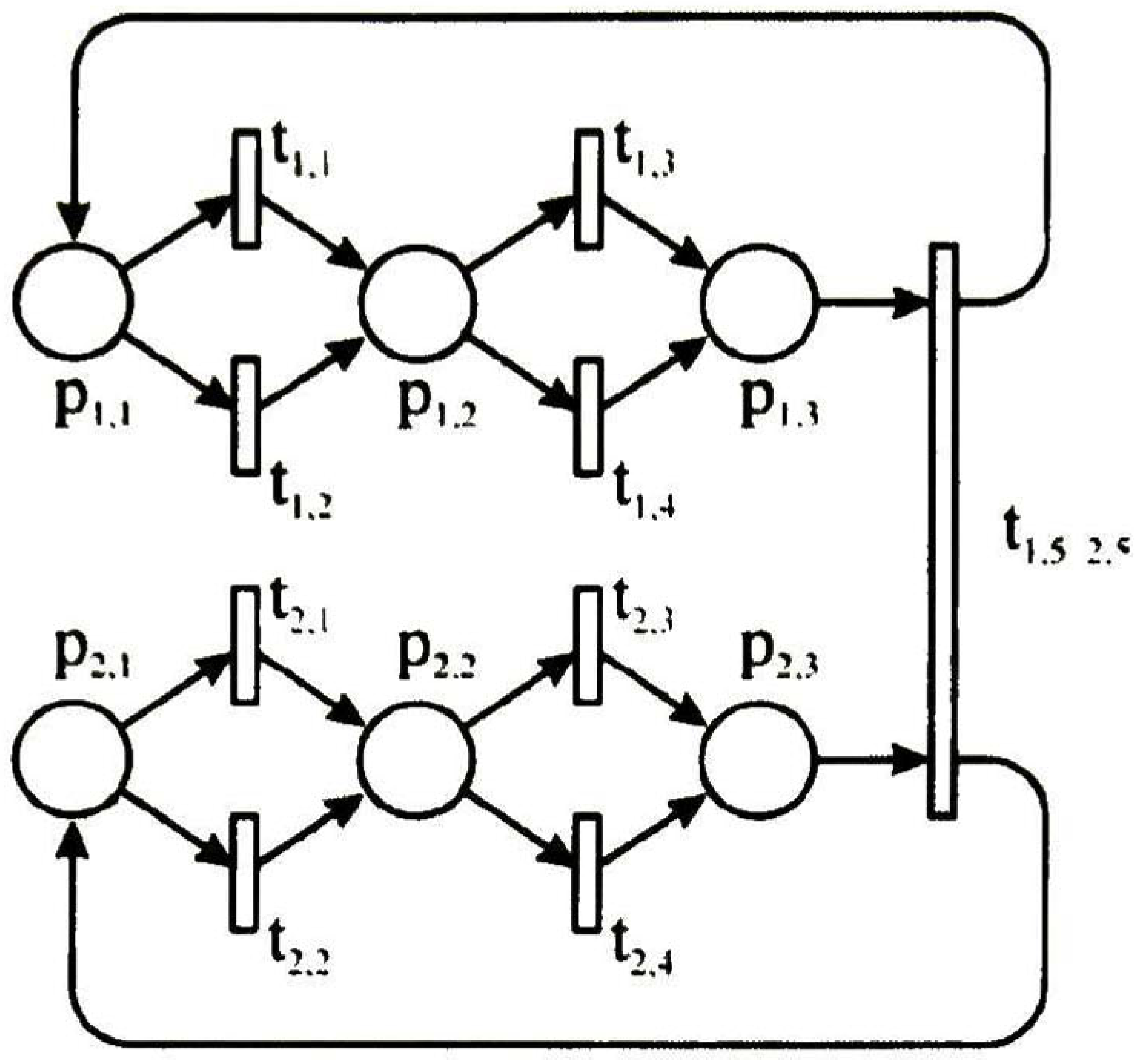

Figure 3.2: A synchronised state machines Petri net resulting from merging the state machines N_1 and N_2.

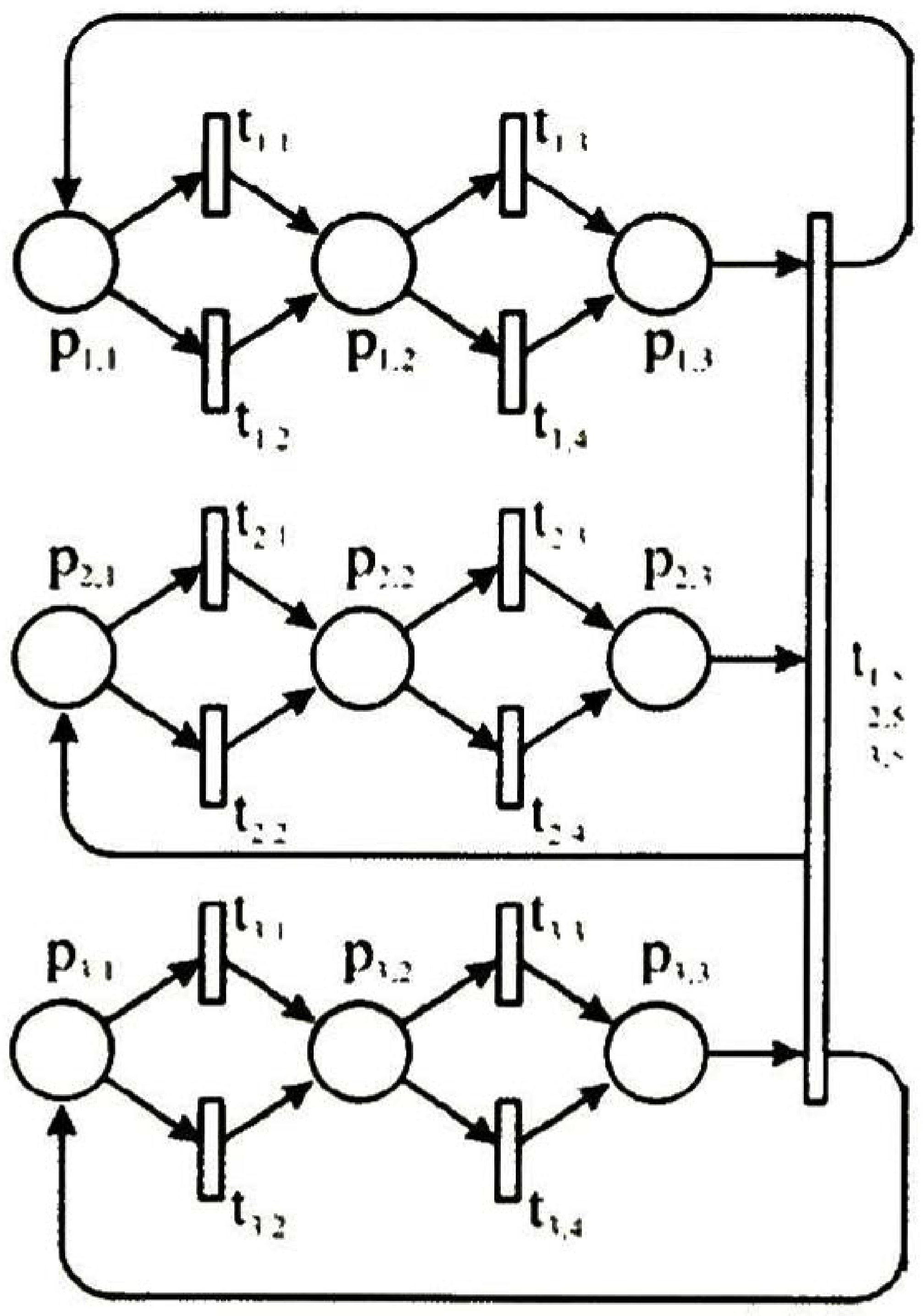

Figure 3.3: A synchronised state machines Petri net resulting from merging the state machine N_3 and the synchronised state machines Petri net N_4.

3.2 Cases of Study

3.2.1 Sales Company

The policy of the company defines a route for each salesperson. Every route has a set of clients and there exist a precedence relation for visiting clients. At the end of the journey all of the salespersons together have a meeting to make bills. For simplicity, let to analyse the case of just two salespersons.

The model of the route for the first salesperson is shown in figure 3.4. The model of the route for the second salesperson is shown in figure 3.5.

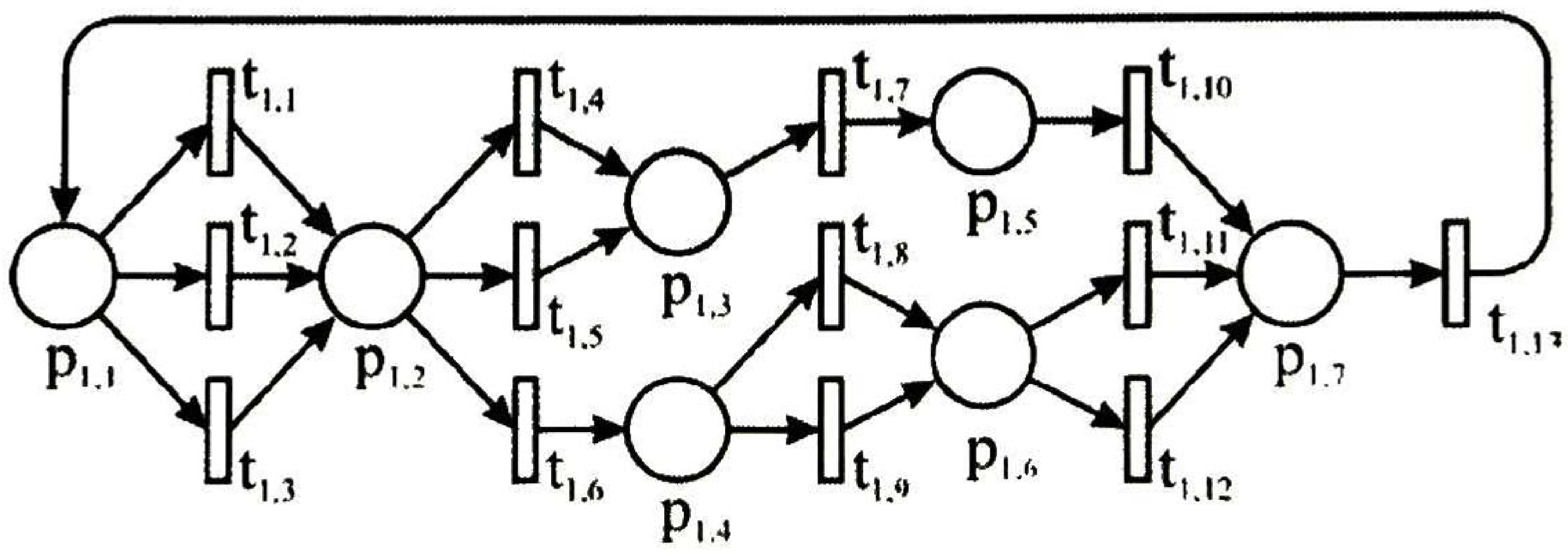

Figure 3.4: Model of the first route.

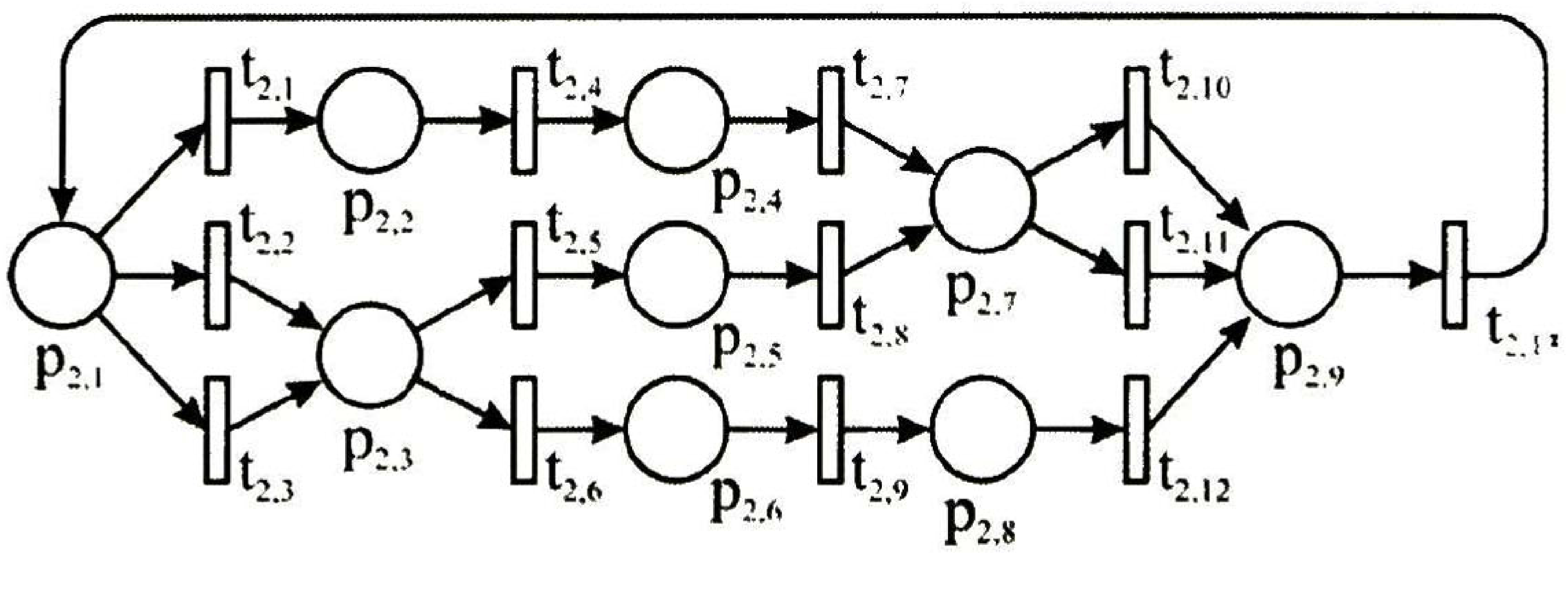

Figure 3.5: Model of the second route.

The company model is built by merging the models of the two routes. The Petri net shown in figure 3.6 represents model of the company.

The policy of the company establishes that for every conflict, the proportion of resources utilization must be equal. Therefore, the firing proportion for every transition in a conflict is 1.

Model Description

The company model is a DTTPN $N = (P, T, Pre, Post, M_0, D)$, where T and D are described as show the following syntax.

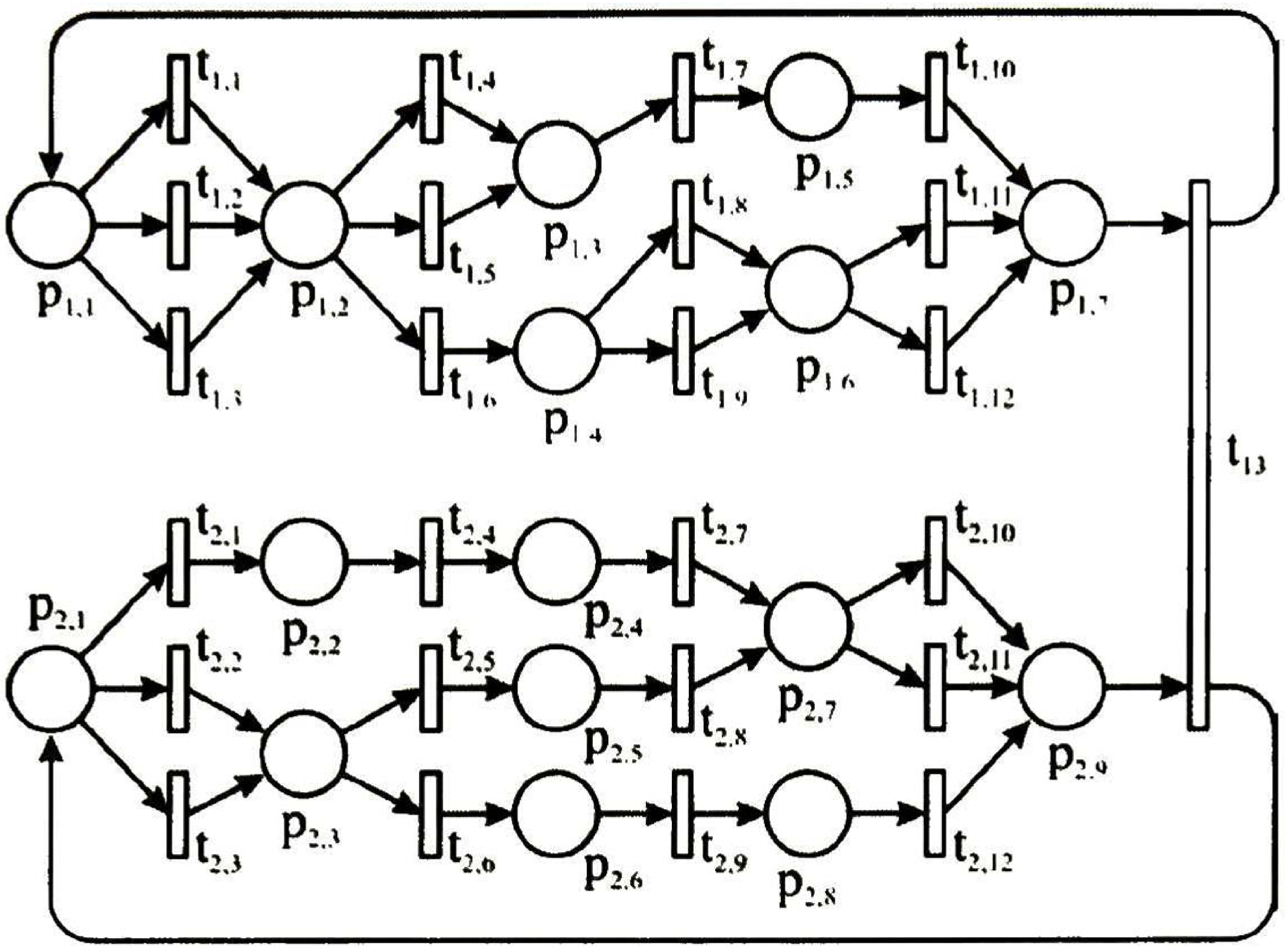

Figure 3.6: Sales company model.

Syntax: *Transition t_i (time delay d_i): Action.*

$t_{1,1}(0.5)$: Salesman 1 visits and assists a client 1.
$t_{1,2}(1)$: Salesman 1 visits and assists a client 2.
$t_{1,3}(1.5)$: Salesman 1 visits and assists a client 3.
$t_{1,4}(3)$: Salesman 1 visits and assists a client 4.
$t_{1,5}(0.5)$: Salesman 1 visits and assists a client 5.
$t_{1,6}(2.5)$: Salesman 1 visits and assists a client 6.
$t_{1,7}(4)$: Salesman 1 visits and assists a client 7.
$t_{1,8}(3.5)$: Salesman 1 visits and assists a client 8.
$t_{1,9}(1)$: Salesman 1 visits and assists a client 9.
$t_{1,10}(2)$: Salesman 1 visits and assists a client 10.
$t_{1,11}(2)$: Salesman 1 visits and assists a client 11.
$t_{1,12}(2)$: Salesman 1 visits and assists a client 12.

$t_{2,1}(2)$: Salesman 2 visits and assists a client 1.
$t_{2,2}(1.5)$: Salesman 2 visits and assists a client 2.
$t_{2,3}(1.5)$: Salesman 2 visits and assists a client 3.
$t_{2,4}(3.5)$ · Salesman 2 visits and assists a client 4.
$t_{2,5}(0.5)$: Salesman 2 visits and assists a client 5.
$t_{2,6}(2.5)$: Salesman 2 visits and assists a client 6.
$t_{2,7}(4)$: Salesman 2 visits and assists a client 7.
$t_{2,8}(3.5)$: Salesman 2 visits and assists a client 8.
$t_{2,9}(1)$: Salesman 2 visits and assists a client 9.

$t_{2,10}(2)$: Salesman 2 visits and assists a client 10.
$t_{2,11}(4)$: Salesman 2 visits and assists a client 11.
$t_{2,12}(2)$: Salesman 2 visits and assists a client 12.

$t_{13}(1)$: Salesman 1 and 2 make the bills.

3.2.2 Manufacturing Company

The organization chart of a manufacturing company is shown in figure 3.7. This company has a well defined procedure to manage the involved processes which is shown in figure 3.8. The manufacturing and administrative processes are exposed with more detail in figures 3.9 and 3.10 respectively.

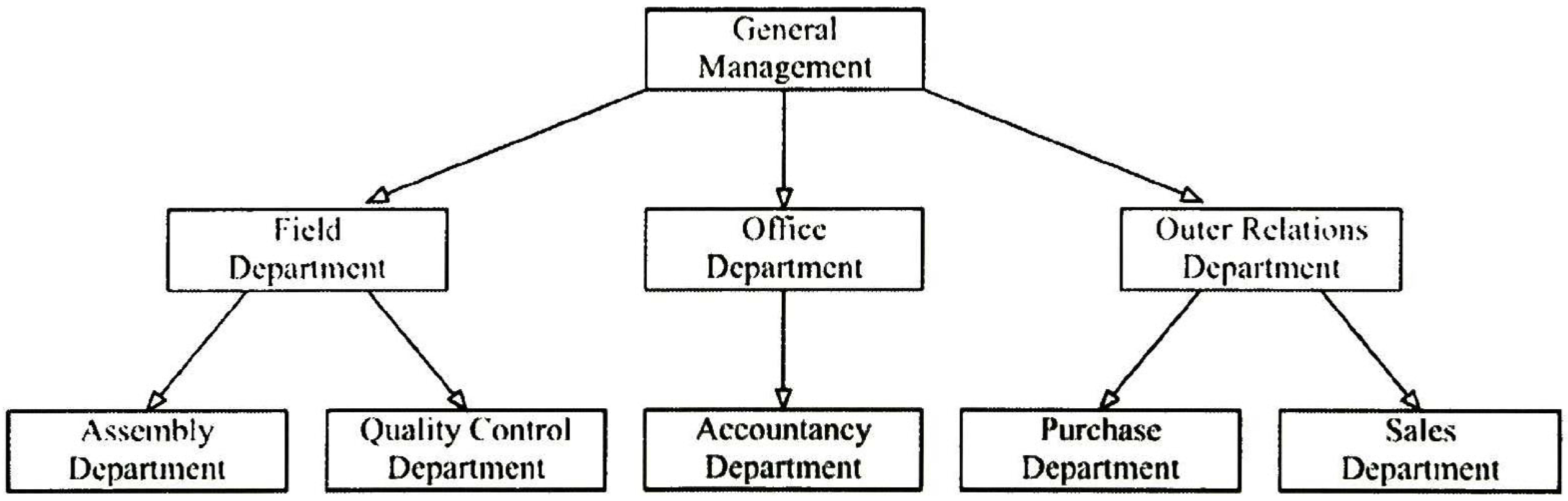

Figure 3.7: Organization chart of a producer company.

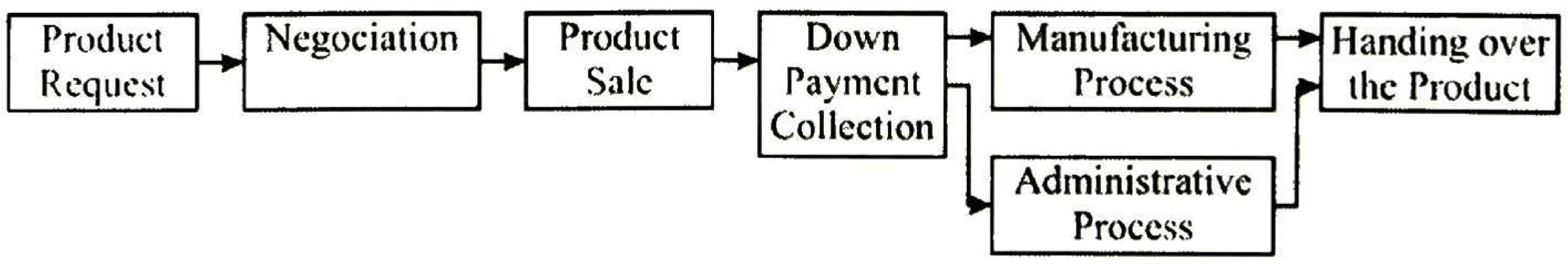

Figure 3.8: Procedure to manage the involved processes.

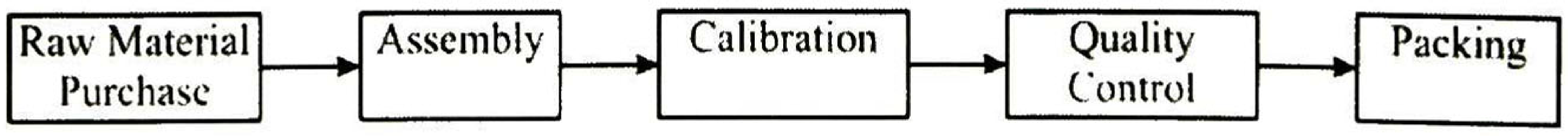

Figure 3.9: Manufacturing Process.

Below is shown a brief description of the departments in the organization chart: their immediate personnel staff and their relationship with the processes.

- **General Management**

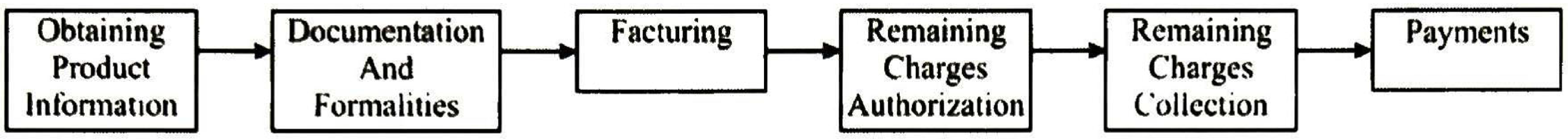
Obtaining Product Information
Documentation And Formalities
Facturing
Remaining Charges Authorization
Remaining Charges Collection
Payments

Figure 3.10: Administrative Process.

  - General Manager (Negotiation, Billing, Remaining Charges Authorization, Remaining Charges Collection, and Handing Over the Product)
  - General Manager Secretary (Documentation and Formalities,)

- **Purchase Department**
  - Supplier 1 (provides raw material)
  - Supplier 2 (provides raw material)
  - Supplier 3 (provides raw material)
  - Supplier 4 (provides raw material)
- **Sales Department**
  - Salesman (assists the product request)
- **Field Department**
  - Supervisor (supervises the field department processes)
- **Assembly Department**
  - Manufacturing Supplier (Assembly, Calibration, Packing, and Quality Tests)
  - Automatic Machine (Assembly, Calibration, and Packing)
  - Semi-Automatic Machine (Assembly, and Calibration,)
  - Manual Assembly Line (Assembly)
  - Manual Calibration Line (Calibration)
  - Packing Line 1 (Packing)
  - Packing Line 2 (Packing)
- **Quality Department**
  - Laboratory 1 (Quality Tests)
  - Laboratory 2 (Quality Tests)
  - Laboratory 3 (Quality Tests)
- **Office Department**

  - Office Manager (Remaining Charges Authorization, Remaining Charges Collection)
  - Office Manager Secretary (Documentation and Formalities, Billing,)
  - Data Collector 1 (Collection of Product Information)
  - Data Collector 2 (Collection of Product Information)
  - Data Collector 3 (Collection of Product Information)

- **Accountancy Department**
  - Counter (Billing)
  - Collector (Remaining Charges Collection)
  - Pay desk (Down Payment Collection, Remaining Charges Collection, Payments)

Once the relationship between the personnel staff and the processes has been defined, then there exist a set of elements in the personnel staff authorized to carry out such process. Figures 3.11, 3.12, and 3.13 show the authorized personnel staff for the process.

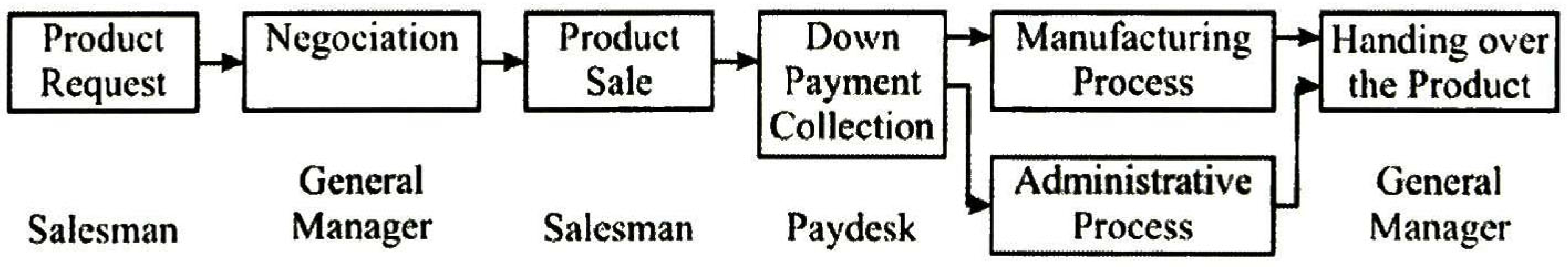

Figure 3.11: Procedure to manage the involved processes and their respective authorized personnel staff.

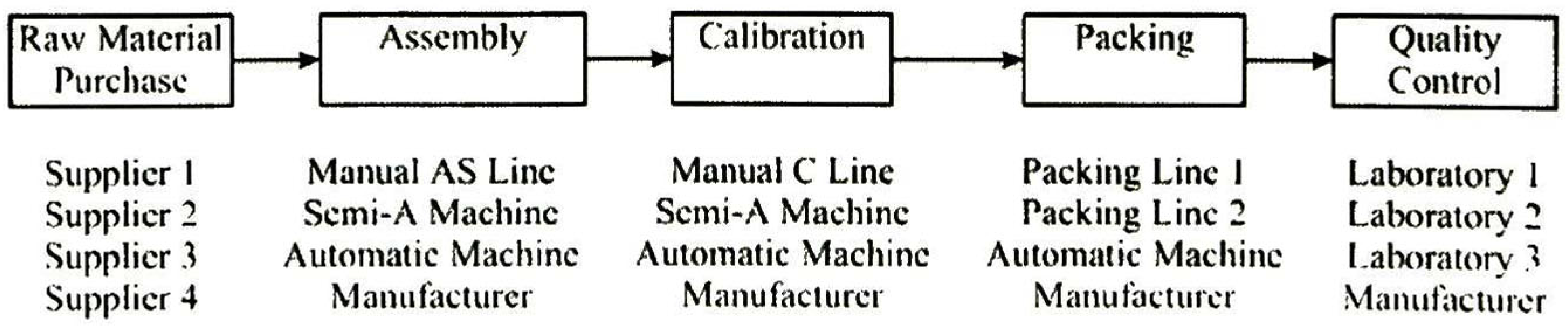

Figure 3.12: Manufacturing Process and their respective authorized personnel staff.

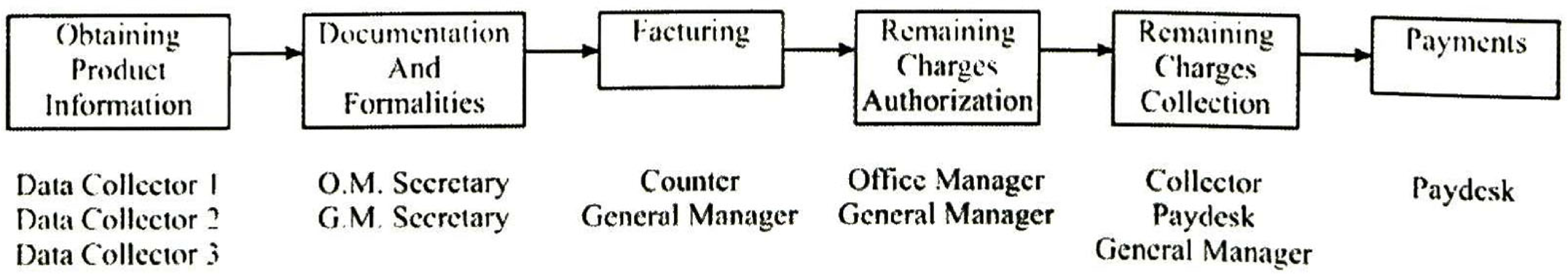

Figure 3.13: Administrative Process and their respective authorized personnel staff.

The Petri net shown in figure 3.14 represent the processes model of the company. Where transitions represents the processes, and actions with their respective time amount.

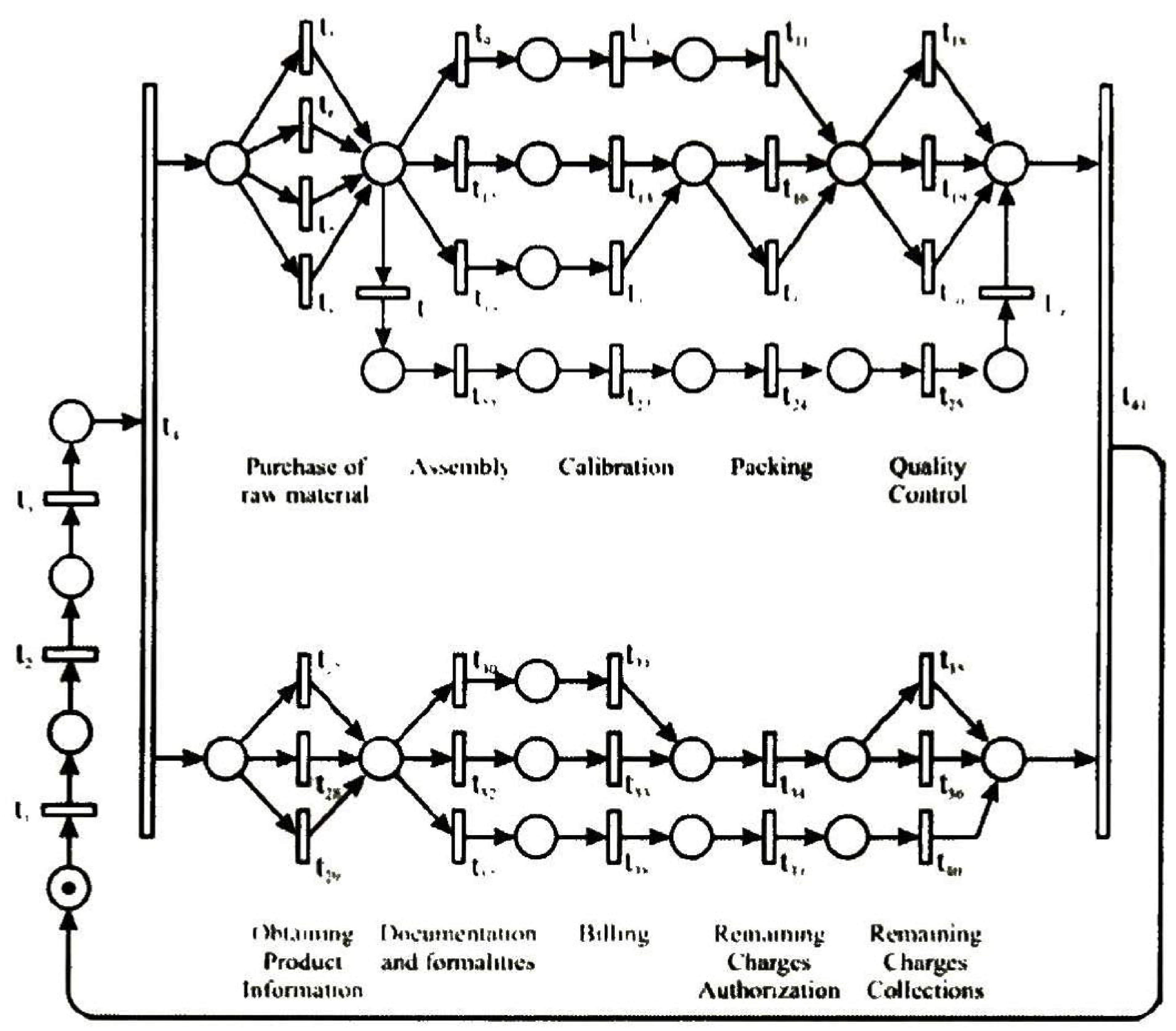
Purchase of raw material
Assembly
Calibration
Packing
Quality Control
Obtaining Product Information
Documentation and formalities
Billing
Remaining Charges Authorization
Remaining Charges Collections

Figure 3.14: Manufacturing company model.

Model Description

The company model is a DTTPN $N = (P, T, Pre, Post, M_0, D)$, where T and D are described as show the following syntax.

Syntax: *Transition t_i (time delay d_i): Action.*

$t_1(2)$: Salesman assists the Product Request.
$t_2(2)$: General Manager negotiates the product features.
$t_3(1)$: Salesman negotiates the product price, and sales the product.
$t_4(4)$: Pay desk collects a down payment.
$t_5(1)$: Supplier 1 provides raw material.
$t_6(4)$: Supplier 2 provides raw material.
$t_7(5)$: Supplier 3 provides raw material.
$t_8(14)$: Supplier 4 provides raw material.
$t_9(18)$: Automatic machine assembles the product.
$t_{10}(2)$: Automatic machine calibrates the product.
$t_{11}(1)$: Automatic machine packs the product.

$t_{12}(26)$: Semi-Automatic machine assembles the product.
$t_{13}(2)$: Semi-Automatic machine calibrates the product.
$t_{14}(32)$: Manual assembly line assembles the product.
$t_{15}(3)$: Manual calibration line calibrates the product.
$t_{16}(2)$: Packing line 1 packs the product.
$t_{17}(4)$: Packing line 2 packs the product.
$t_{18}(2)$: Laboratory 1 applies and evaluates quality tests.
$t_{19}(5)$: Laboratory 2 applies and evaluates quality tests.
$t_{20}(6)$: Laboratory 3 applies and evaluates quality tests.
$t_{21}(1)$: A transportation company sends the raw material to a outsourcing manufacturer company.
$t_{22}(10)$: Manufacturer company assembles the product.
$t_{23}(1)$: Manufacturer company calibrates the product.
$t_{24}(1)$: Manufacturer company packs the product.
$t_{25}(1)$: Manufacturer company applies and evaluates quality tests.
$t_{26}(1)$: A transportation company brings back the product from the manufacturer company.
$t_{27}(1)$: Data Collector 1 collects product, required documentation and required formalities information.
$t_{28}(4)$: Data Collector 2 collects product, required documentation and required formalities information.
$t_{29}(5)$: Data Collector 3 collects product, required documentation and required formalities information.
$t_{30}(1)$: Office Manager Secretary accomplishes the documentation and formalities.
$t_{31}(1)$: Counter accomplishes the billing.
$t_{32}(2)$: Office manager accomplishes the documentation and formalities.
$t_{33}(2)$: Office manager accomplishes the billing.
$t_{34}(2)$ Office manager authorizes the remaining charges collection.
$t_{35}(3)$: Collector collects the remaining charges.
$t_{36}(8)$ · Pay desk collects the remaining charges.
$t_{37}(3)$: General manager secretary accomplishes the documentation and formalities.
$t_{38}(3)$: General manager secretary accomplishes the billing.
$t_{39}(1)$: General manager authorizes the remaining charges collection.
$t_{40}(2)$: General manager collects the remaining charges.
$t_{41}(1)$: General manager hands over the product.
Since the policy of the company establishes that for every conflict, the proportion of resources utilization must be equal, and then the firing proportion for every transition in a conflict is 1.

Model Abstraction

Once the company structure is modelled by a Petri net, the model must be simplified (preserving the physical meaning) in order to make the analysis easier. Therefore, reduction will be achieved preserving just conflicts in order to help the general manager to find out the best sequence which minimize the production cycle time.

Reducing the model, meiosis technique is applied in order to better apply a sequence of fusion technique later on. The sequence of transformations applied to the model is shown below and figure 3.15 shows the resulting

Petri net representing the reduced modell..

$Fusion(t_{41}, t_4)[t_a, 10]$
$Fusion(t_9, t_{11})[t_b, 21]$
$Fusion(t_{12}, t_{13})[t_c, 28]$
$Fusion(t_{14}, t_{15})[t_d, 35]$
$Fusion(t_{21}, t_{26})[t_e, 15]$
$Meiosis(\bullet t_{34}, t_{34})$
$Fusion(t_{30}, t'_{34})[t_f, 4]$
$Fusion(t_{32}, t_{34})[t_g, 6]$
$Fusion(t_{37}, t_{40})[t_h, 9]$

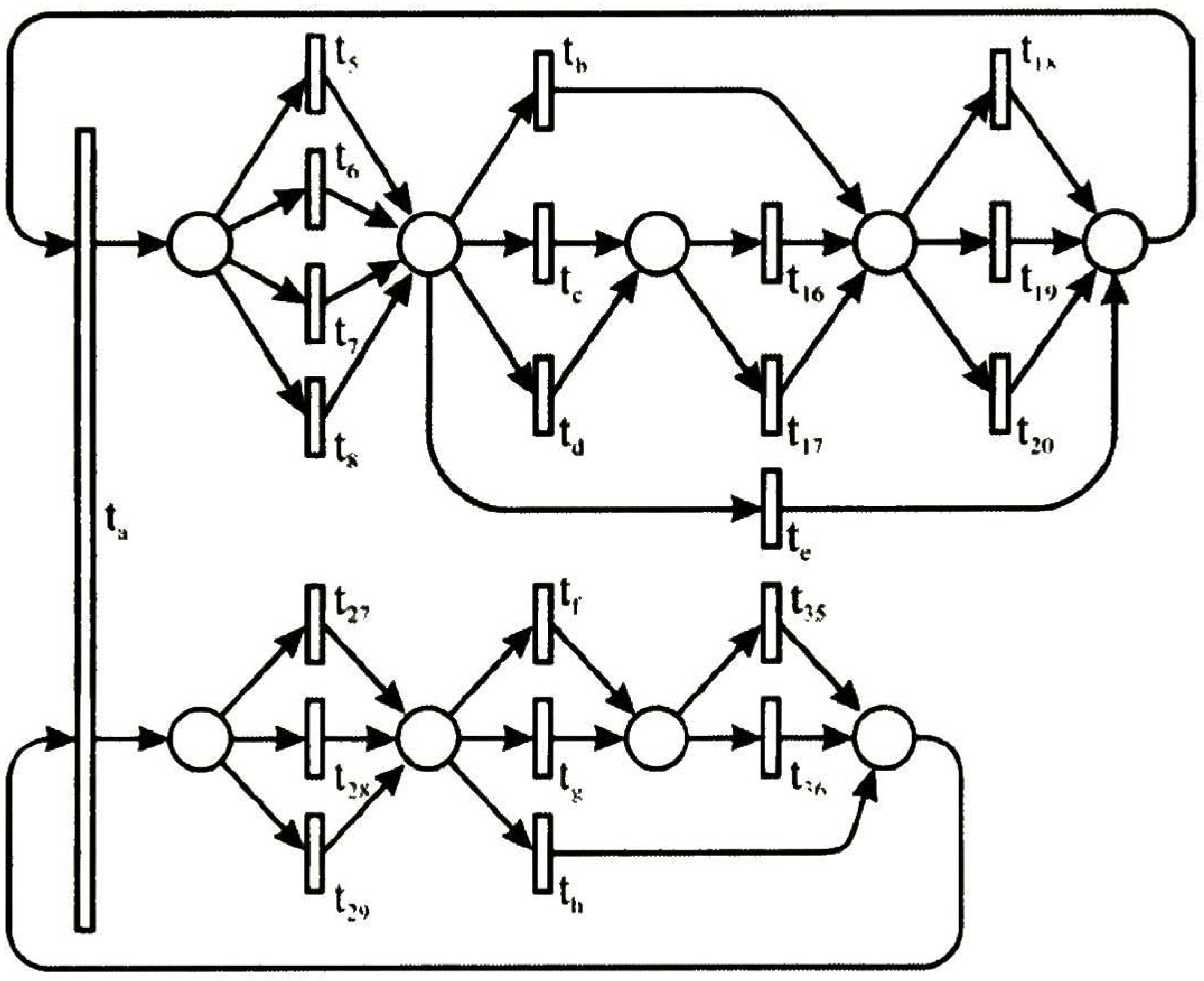

Figure 3.15: Company reduced model.

3.3 Conclusions

Synchronised state machines is a subclass of free choice Petri net capable to model a wide variety of DEDS, including other subclasses as state machines, free choice C*, etc. Systems modelled by synchronised state machines can become more complex by merging subsystems modelled by state machines or other synchronised state machines.

Synchronised state machines facilitate the description and analysis of a complex model by allowing differentiating subsystems easily.

Chapter 4

SCHEDULING OF DEDS MODELED WITH SYNCHRONISED STATE MACHINES

"Local optimality" property guaranties the system optimality through the optimisation of subsystems. Local optimality property can be easily analysed in systems modelled by timed Petri nets. This analysis is done through the cyclicity property using a structural method: invariants method.

This chapter proves that synchronised state machines exhibit the local optimality property. An heuristic is proposed to compute a good schedule for systems whose model is represented by this subclass of free choice Petri nets.

It is also studied the relationships between P-semiflows and T-semiflows in a free choice Petri net. The analysis resides on the use of a T-cover; it seems a natural extension of the works done for marked graphs and state machines.

4.1 Introduction

Although results for free choice Petri nets are quite pessimistic, this subclass is studied to determine the causes of the NP-Completeness, or at least to find:

- subclasses of free choice Petri nets (different from marked graphs and state machines) where the optimal schedule and the optimal cycle time can be computed in polynomial time.
- heuristics based on the previous analysis, allowing to find good schedules.

An optimistic bound of the minimum cycle time can also be found in polynomial time. This bound could be used to guide the search of a good schedule algorithm.

There exists a result [Ram93] stating that performing local optimisations do not lead to a global system optimisation. But in many cases, however, systems can be globally optimised through local optimisations. The interest of studying systems exhibiting local optimality is due to they can be optimised easier and faster.

For a problem whose complexity is greater than polynomial, the divide and concord idea is useful since the problem can be solved faster through local optimisations than through a global optimisation. In other words, if $\tau(X)$ is the required time to solve the schedule problem of the global system, and $\tau(x_i)$ is the required time to solve the schedule problem of a subsystem x_i, where $\cup_{i=0}^{n} x_i = X$ and $x_i \cap x_j = \emptyset \; \forall x_i \neq x_j$ then

$$\tau(X) \geq \sum_{i=0}^{n} \tau(x_i)$$

Definition 4.1 *[Ram93b] If a system can be divided into subsystems and the optimal performance of the subsystems guaranty the optimal performance of the global system, then the system exhibits the local optimality property with respect to such splitting.*

The procedure to optimise a system through local optimisation consists in three phases:

1. To divide a system into subsystem,
2. to optimise the subsystems, and
3. to integrate the optimal solutions of the subsystems.

Where the subsystems are represented by timed Petri nets whose optimal cycle time and optimal schedule can be computed through well defined procedures.

Previous results [Ram93] of systems with parallel branches establishes that there exist a relationship between such branches. Therefore, these branches cannot be optimized independently.

Example 4.1 *Let the synchronised state machines Petri net shown in figure 4.1 represent the abstract model of a DEDS. The model has two state machines synchronised by transition t_1. Let $D = [0, 3, 4, 3, 4, 7]$ be the time vector and let $V_r = [2, 1, 1, 1, 1, 2]$ the visit ratio. The cycle time of the T-semiflows in the T-cover of the global system is equal to the slowest P-semiflow of such T-semiflow, and such relationship impedes to optimise the state machines separately.*

Therefore, if the first state machine has four elemental T-semiflows whose cycle time is:
$\mathcal{T}_1 = [1, 1, 0, 1, 0, 0]$, $c_1 = 6$
$\mathcal{T}_2 = [1, 1, 0, 0, 1, 0]$, $c_2 = 7$
$\mathcal{T}_3 = [1, 0, 1, 1, 0, 0]$, $c_3 = 7$
$\mathcal{T}_4 = [1, 0, 1, 0, 1, 0]$. $c_4 = 8$
and the second state machine has just one elemental T-semiflow whose cycle time is:
$\mathcal{T}_5 = [1, 0, 0, 0, 0, 1]$, $c_5 = 7$.

Then, since every elemental T-semiflow of the first state machine is synchronised with $\mathcal{T}_5$, then, in order to avoid delays, the T-invariants of the first state machine must have a cycle time as equal as possible to c_5.

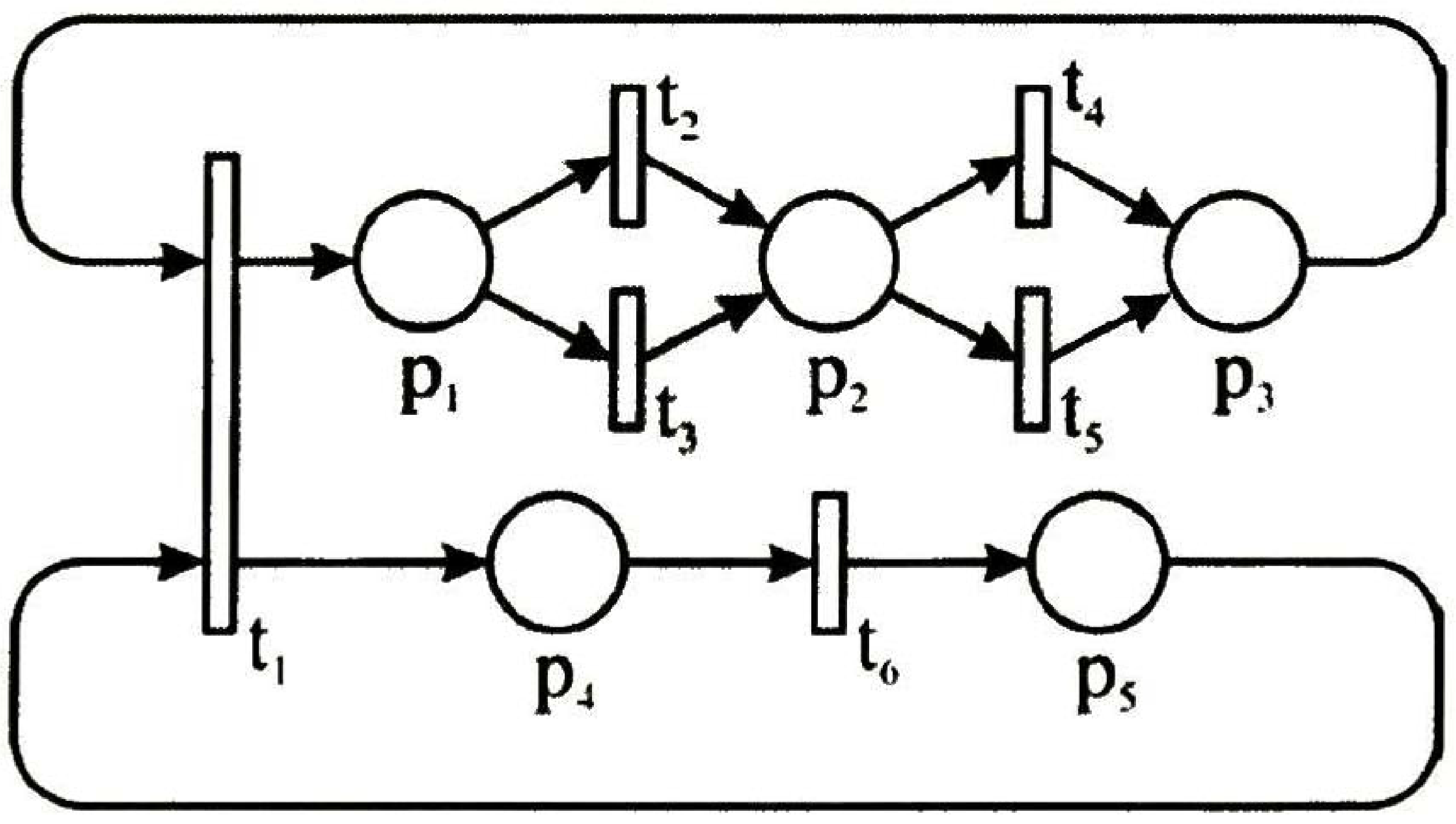

Figure 4.1: Abstract model of a DEDS.

4.1.1 Discovery

In spite of the NP Completeness of the scheduling problem in free choice Petri nets, it is possible to study other systems' features. One of such features is the cycle time, the one we are interested in.

By building several T-covers and schedules for several systems with parallel branches, a discovery was found: The T-covers with the minimal cycle time were those which its P-semiflow have the same cycle time in the different T-semiflows of the T-cover.

Example 4.2 *The Petri net shown in figure 4.1 has two possible schedules represented by two T-covers. Figure 4.2 shows the cycle time of the P-semiflows of every T-semiflow belonging to the first T-cover, while figure 4.3* *shows the cycle time of the P-semiflows of every T-semiflow belonging to the second T-cover. The white bars represent the cycle time of one P-semiflow and the black bars represent the cycle time of the other P-semiflow.*

Note that the variance of the cycle time of a P-semiflow in the different T-semiflows of the T-cover is minimal, then the cycle time of such T-cover is minimal.

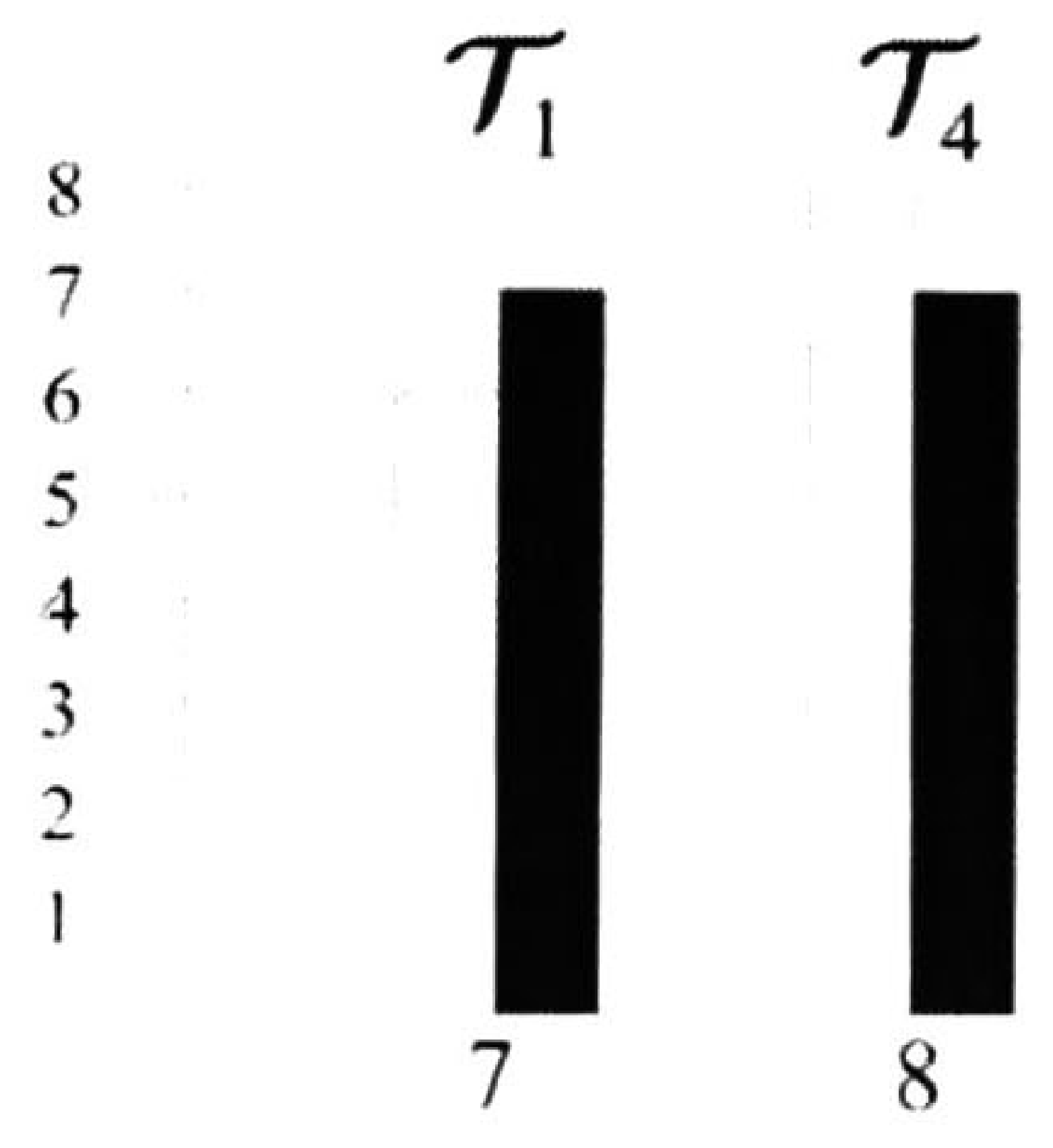

Figure 4.2: A T-cover whose cycle time is 15.

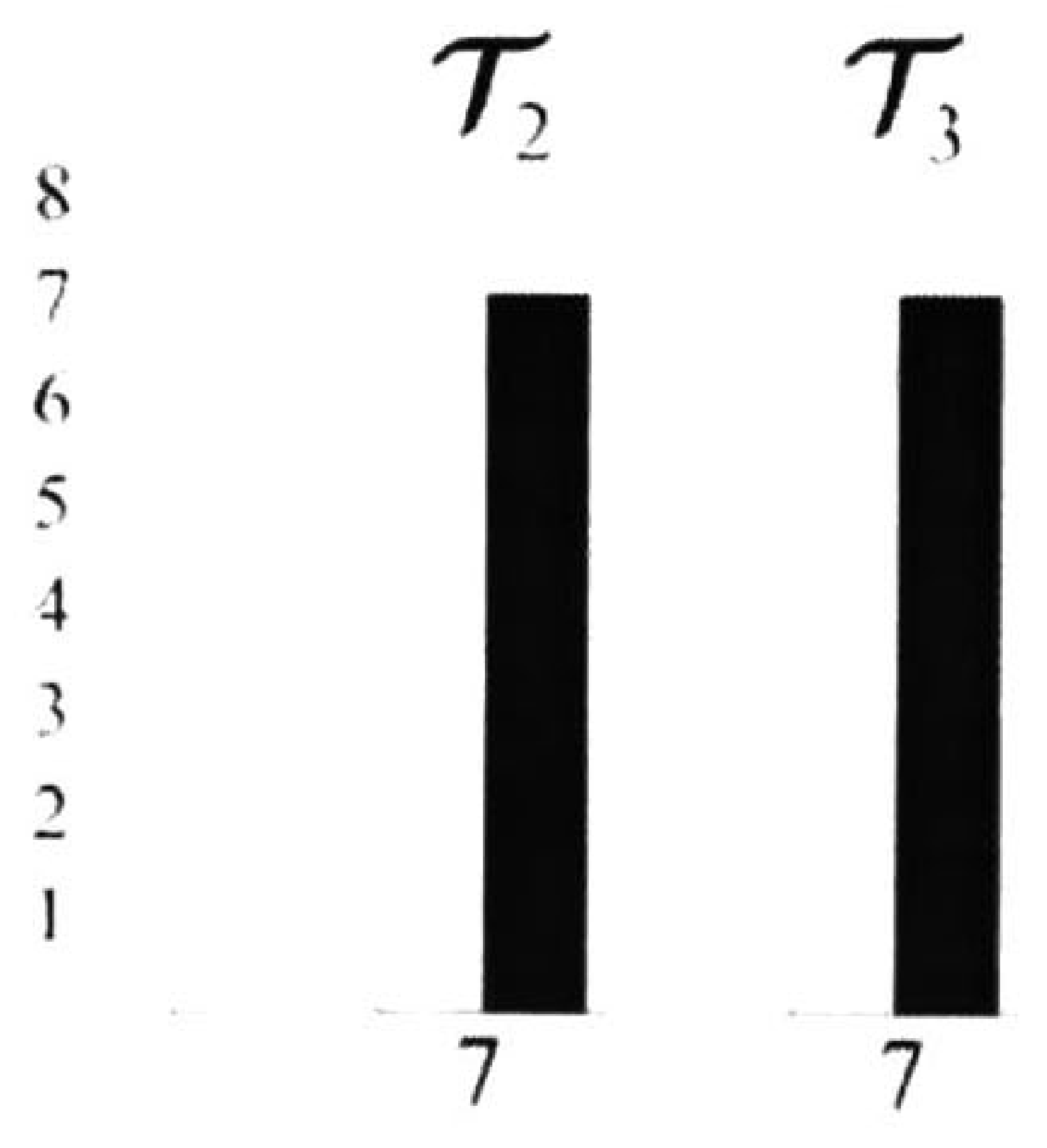

Figure 4.3: A T-cover whose cycle time is 14.

Therefore, this chapter proves that it is possible to use local optimisations and find a good (or optimal) schedule for systems with parallel branches through a statistic perspective.

4.2 Synchronised State Machines Analysis

4.2.1 State Machines Analysis

Theorem 4.1 *For every state machine $N = (P, T, Pre, Post)$, if there exist a transition $t_i \in T$ belonging to the support of every elemental T-semiflow, then the number of elemental T-invariants required to build an elemental T-cover fulfilling the visit ratio is always the same.*

Proof. Let $V_r = [v_1, ..., v_m]$ be the visit ratio such that $v_i > 0$. Since every elemental T-semiflow is a vector whose entries are mappings to the set $\{0, 1\}$, then v_i represents the number of elemental T-semiflows that transition t_i requires to fulfil the visit ratio.
Since transition t_i belongs to the support of every elemental T-semiflow of the net, then the entry v_i represents the number of elemental T-semiflows required to build the T-cover. Therefore, the number of elemental T-invariants required to build an elemental T-cover (fulfilling the visit ratio) is constant. ■

Example 4.3 *Figure 4.4 shows a state machine where t_5 belongs to the support of every elemental T-semiflow. Let $V_r = [1, 1, 1, 1, 2]$ be the visit ratio, then the number of elemental T-semiflows required to build a T-cover is 2. For this Petri net, there exist two elemental T- covers fulfilling the visit ratio $C_1 = \{\mathcal{T}_1, \mathcal{T}_4\}$, and $C_2 = \{\mathcal{T}_2, \mathcal{T}_3\}$ where*
$\mathcal{T}_1 = [1, 0, 1, 0, 1]$,
$\mathcal{T}_2 = [1, 0, 0, 1, 1]$,
$\mathcal{T}_3 = [0, 1, 1, 0, 1]$,
$\mathcal{T}_4 = [0, 1, 0, 1, 1]$.

It can be seen that both C_1 and C_2 have 2 elemental T-semiflows, the same number of the value in the visit ratio for transition t_5.

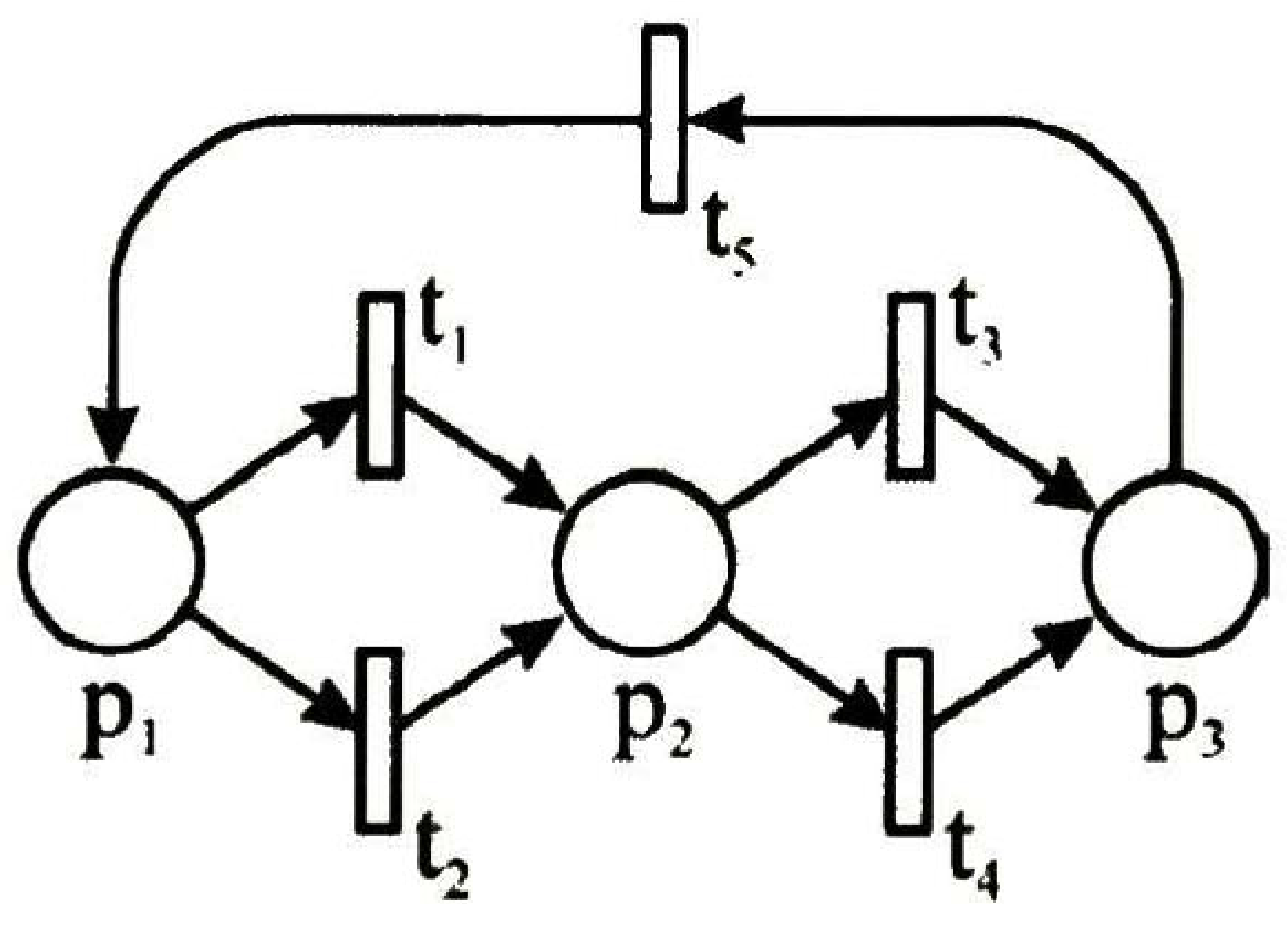

Figure 4.4: A state machine where transition t_5 belongs to the support of every elemental T-semiflow.

Corollary 4.1 *Let $N = (P, T, Pre, Post)$ be a state machine, Cl be a specific cluster, and $T_C = \{ t_i \mid t_i \in Cl \}$ be the set of transitions t_i belonging to Cl. If for every elemental T-semiflow X of N there exist a transition t_i such that $t_i \in \langle X \rangle$ and $t_i \in T_C$, then the number of elemental T-invariants required to build an elemental T-cover fulfilling the visit ratio is always the same.*

Proof. If for every elemental T-semiflow X of N there exist a transition t_i such that $t_i \in \langle X \rangle$ and $t_i \in T_c$, then the number n of elemental T-semiflows required to build the T-cover is given by

$$n = \sum_{i=1}^{k} v_i$$

where v_i represents the value in the visit ratio associated to transition t_i,

k represents the number of transitions in the cluster, and
n represents the number of elemental T-semiflows required to build the T-cover, since an elemental T-semiflow cannot contain two or more transitions belonging to the same cluster. ■

Example 4.4 *Figure 4.5 shows a state machine with two clusters $Cl_1 = \{p_1, t_1, t_2\}$ and $Cl_2 = \{p_2, t_3, t_4, t_5\}$ such that every elemental T-semiflow has a transition t_i belonging to T_{C1} and T_{C2}. If for every conflict, the firing proportion is equal for every transition then the visit ratio is $V_r = [3, 3, 2, 2, 2, 2, 2, 2]$. Now, since the support of every elemental T-semiflow of the net has a transition belonging to T_{C1}, hence the number of elemental T-semiflows require to build a T-cover fulfilling the visit ratio is $n = v_1 + v_2 = 3 + 3 = 6$. Furthermore, the support of every elemental T-semiflow of the net also has a transition belonging to T_{C2}, hence the number of elemental T-semiflows require to build a T-cover fulfilling the visit ratio is also $n = v_3 + v_4 + v_5 = 2 + 2 + 2 = 6$*

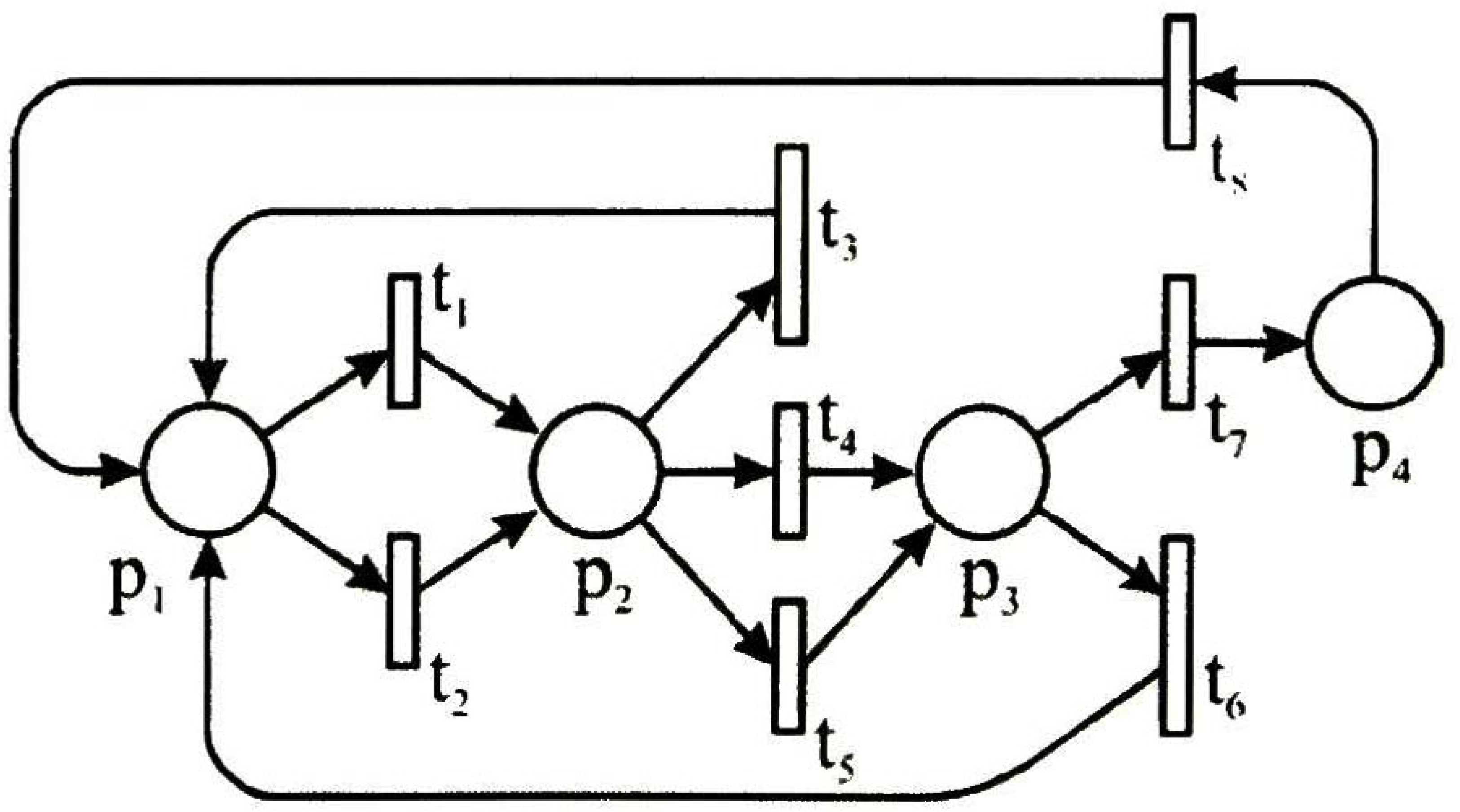

Figure 4.5: A state machine where there exist a transition in the support of every elemental T-semiflow belonging to a specific cluster.

Note that in the general case of a state machine, the number of elemental T-semiflow required to build an elemental T-cover fulfilling the visit ratio cannot be computed; since there could be different T-semiflows whose supports have no transition belonging to the same cluster.

Example 4.5 *$T_{C1} = \{t_1, t_2\}$, $T_{C2} = \{t_3\}$, $T_{C3} = \{t_4, t_6\}$, and $T_{C4} = \{t_5\}$ are the sets of transitions associated to the clusters Cl_1, Cl_2, Cl_3, and Cl_4 respectively of the state machine shown in Figure 4.6. $\langle T_1 \rangle = \{t_1, t_3\}$ and $\langle T_2 \rangle = \{t_5, t_6\}$ are the supports of two elemental T-semiflows. The transitions in the supports of such elemental T-semiflows do not belong to the common cluster. Therefore, the number of elemental T-semiflows required to build a T-cover fulfilling the visit ratio cannot be computed with the previous result.*

If there exist two clusters Cl_1 and Cl_2 such that the support of every elemental T-semiflow containing transitions of Cl_1 and do not contain transitions of Cl_2, or vice versa, and if the support of every elemental T-semiflow of the net has a transition belonging to Cl_1 or Cl_2; then the number of elemental T-semiflow required to build a T-cover fulfilling the visit ratio can be computed extending the previous result. This conclusion can be extended

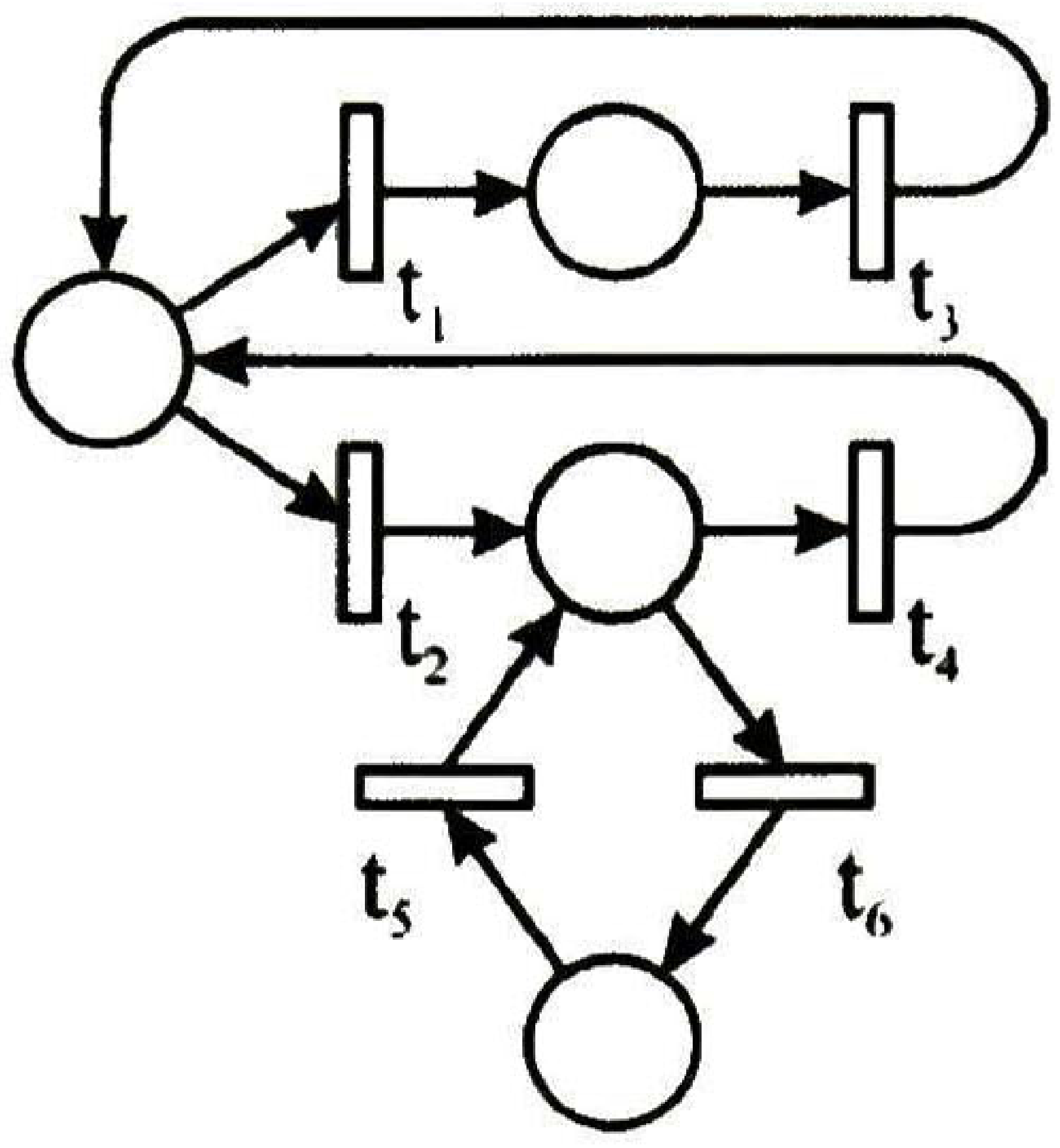

Figure 4.6: A State Machine where the number of elemental T-semiflows in the T-cover fulfilling the visit ratio cannot be computed easily.

for more than two clusters, but in the general case this information is not known or difficult to obtain, therefore it becomes unpractical.

Theorem 4.2 *For every state machine, every elemental T-cover fulfilling the visit ratio has the same cycle time.*

Proof. Since there are no synchronizations, there will be only transitions' times additions. For commutability properties of the addition, the cycle time is the same for any T-cover fulfilling the visit ratio since the cycle time is equal to the weighted addition of transitions' times. Where the visit ratio states the weight, and the execution of the T-cover must be done applying the rule "*to execute t-invariants and to fire transitions as soon as they become enabled*". ■

From theorem 4.2, the cycle time C of a state machine fulfilling the visit ratio v_r is defined by the following equation:

$$C = v_r \ D \tag{4.1}$$

Where D is the delay vector associated to the DTTPN.

Definition 4.2 *Let $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, ... \mathcal{T}_n\}$ be a T-cover fulfilling the visit ratio, and let c_i be the cycle time of a T-semiflow $\mathcal{T}_i \in \mathcal{T}$ The average T-invariants cycle time ξ of the set $\mathcal{T}$ is defined by the following equation*

$$\xi = \frac{1}{n}\sum_{i=1}^{n} c_i$$

where n is the cardinality of $\mathcal{T}$.

Proposition 4.1 *Let ξ be the average T-invariants cycle time of an elemental T-cover. For a state machine, the average T-invariants cycle time ξ is the same for every T-cover fulfilling the visit ratio.*

Proof. The average cycle time ξ of a set of T-semiflows is given by

$$\xi = \frac{1}{n}\sum_{i=1}^{n} c_i$$

where

n : is the number of T-semiflows, and

c_i : is the cycle time of the i-th T-semiflow.

For a state machine, from theorem 4.1 n is constant, and from theorem 4.2 $\sum_{i=1}^{n} c_i$ is constant for any T-cover fulfilling the visit ratio. Consequently, ξ is the same for any T-cover. Furthermore, since the visit ratio is a nonnegative linear combination of elemental T-semiflows, then the average cycle time of the elemental T-semiflows belonging to a T-cover fulfilling the visit ratio is

$$\xi = \frac{1}{n}\sum_{i=1}^{n} c_i = \frac{C}{n} = \frac{v_r \cdot D}{n}$$

■

Example 4.6 *Let $D = [2, 6, 4, 1, 0]$ be the time vector and let $V_r = [1, 1, 1, 1, 2]$ the visit ratio associated to the state machine shown in Figure 4.4. There exist two elemental T-covers fulfilling the visit ratio providing the same cycle time:,*

$C = v_r \cdot D = [1, 1, 1, 1, 2] \cdot [2, 6, 4, 1, 0] = 13$

$\mathcal{C}_1 = \{\mathcal{T}_1, \mathcal{T}_4\}$	$C_1 = 13,$
$\mathcal{C}_2 = \{\mathcal{T}_2, \mathcal{T}_3\}$	$C_2 = 13.$

Where:

$\mathcal{T}_1 = [1, 0, 1, 0, 1]$	$c_1 = 6,$
$\mathcal{T}_2 = [1, 0, 0, 1, 1]$	$c_2 = 3,$
$\mathcal{T}_3 = [0, 1, 1, 0, 1]$	$c_3 = 10,$
$\mathcal{T}_4 = [0, 1, 0, 1, 1]$	$c_4 = 7,$

Knowing the optimal cycle time of the net (13 time units) and the number of T-semiflows required to build the T-cover (2 T-semiflows), the average t-invariants cycle time ξ is 7.5 time units.

Proposition 4.2 *The minimization of the cycle time of the slowest T-semiflow in a T-cover of a state machine is equivalent to minimize the standard deviation of the cycle time of the T-semiflows in such T-cover.*

Proof. The average cycle time ξ of a set of T-semiflows is given by

$$\xi = \frac{1}{n}\sum_{i=1}^{n} C_i$$

where

n : is the number of T-semiflows, and

C_i : is the cycle time of the i-th T-semiflow.

For a state machine of the synchronised state machines Petri net, n is constant (see theorem 4.1), $\sum_{i=1}^{n} C_i$ is constant for any T-cover (see theorem 4.2), and ξ is constant (see proposition 4.1).

Let C_j be the cycle time of the slowest T-semiflow $\mathcal{T}_j$, and $\Delta_j = C_j - \xi$ be the difference between C_j and ξ. Now, ξ is defined as

$$C_j = \xi + \Delta_j$$
$$\xi = \frac{1}{n}\sum_{\substack{i=1\\ i\neq j}}^{n} C_i + \xi + \Delta_j$$

then

$$\xi + \Delta_j = \xi - \frac{1}{n}\sum_{\substack{i=1\\ i\neq j}}^{n} C_i$$

If the cycle time of the slowest T-semiflow $\mathcal{T}_j$ is forced to be equal to ξ then the time Δ_j is distributed in the faster T-semiflows, increasing their cycle time, and forcing them to be slower.

$$\xi = \xi - \left(\frac{1}{n}\sum_{\substack{i=1\\ i\neq j}}^{n} C_i + \Delta_j \right)$$

If this procedure is done for every T-semiflow $\mathcal{T}_j$ whose cycle time C_j is greater than ξ, then every T-semiflow $\mathcal{T}_k$ whose cycle time C_k is smaller than ξ will increase its cycle time until been equal to ξ. Consequently, the cycle time of every T-semiflow will be equal to ξ, and the cycle time variance of the T-semiflows will be minimal. ■

4.2.2 T-cover of State Machines

The visit ratio $v_{r,n}$ of the n-th state machine N_n belonging to a synchronised state machines Petri net N is a mapping from the visit ratio v_r of the synchronised state machines Petri net to the space state of the n-th state machine:

$$v_{r,n} = v_r|_{N_n}$$

Since the visit ratio v_r is a positive linear combination of elemental T-semiflows, the visit ratio of the n-th state machine $v_{r,n}$ can contain an elemental T-semiflows of such state machine more than once. Hence, in order not to compute the same elemental T-semiflows of the state machine more than once, the visit ratio used to compute the T-cover of the state machine is the vector resulting from dividing the visit ratio $v_{r,n}$ between the greatest common divisor of its entries. After the T-cover is found, each elemental T-semiflow of the T-cover is multiplied by such greatest common divisor.

Remark. It is said that a T-semiflow whose cycle time c_t fits into ξ is such that $\xi > c_t$ and $\xi - c_t$ is minimal. From example 4.6, $\xi = 7.5$, and there exist four T-semiflows $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$, and $\mathcal{T}_4$ whose cycle times are 6, 3, 10, and 7 respectively, therefore the T-semiflow whose cycle time better fits into ξ is $\mathcal{T}_4$. ■

From proposition 4.2 is proposed an algorithm for finding out a T-cover whose T-semiflows have a minimal variance

Algorithm 4.1 *Building the best T-cover of a state machine belonging to a synchronised state machines Petri nets fulfilling the visit ratio.*

1. From theorem 4.1, obtain the number n of elemental T-semiflows required to build a T-cover fulfilling the visit ratio.
2. Compute the cycle time C of the state machine using the equation 4.1.
3. Compute the average T-invariants cycle time ξ of the T-cover as follows
$$\xi = \frac{C}{n}$$
4. Define n_r as the number of remaining T-semiflows to complete the T-cover,
C_r as the sum of the cycle time of the remaining T-semiflows to complete the T-cover,
ξ_r as the average cycle time of the remaining T-semiflows to complete the T-cover,
V_{rr} as a vector representing the values of the visit ratio for the remaining T-semiflows to complete the T-cover.
5. Compute a T-semiflow $\mathcal{T}_i$ such that c_i fits into ξ_r and $\langle \mathcal{T}_i \rangle \subseteq \langle V_{rr} \rangle$
6. In order to compensate $\Delta_i = \xi - c_i$ (the difference between the average cycle time and the cycle time of the last computed T-semiflow). Redefine new values for C_r, n_r, and ξ_r as follows
$$\begin{aligned} C_r &\leftarrow C_r - C_i \\ n_r &\leftarrow n_r - 1 \\ \xi_r &\leftarrow \frac{C_r}{n_r} \\ V_{rr} &\leftarrow V_{rr} - \mathcal{T}_i \end{aligned}$$
7. If $n = 0$
    (a) then the algorithm finishes
    (b) else Go to step 5.

With the exception of the step 5 the algorithm is polynomial.

Theorem 4.3 *Let N be a state machine. The problem of finding out a T-semiflow $\mathcal{T}_\imath$ of N such that its cycle time c_i fits into a constant ξ is NP-Complete.*

Proof. Since the maximum weight that a transition can have in an elemental T-semiflow is 1,then an elemental T-semiflow is a proper subset of the set of transitions T.
Being a proper subset, a T-semiflow $\mathcal{T}_i$ is a combination of transitions holding the equation 2.4 ($\mathbf{N} \cdot \mathcal{T}_\imath = 0$).
If a T-semiflow $\mathcal{T}_\imath$ is a combination of transitions holding the equation 2.4, and the addition of the transitions' time must fit into a constant ξ, then this problem can be interpreted as the bin packing problem, where the ξ

represents the capacity of the container and transitions t_i represent the pieces to be stored into the container. The bin packing problem has been proved to be NP-Complete. ■

From proposition 4.2, the optimal T-cover must be built finding out T-invariants whose execution time fit into the average t-invariants cycle time ξ. From theorem 4.3, solving this problem cannot be achieved in polynomial time. In spite of the NP-Completeness of finding out such T-semiflow, an heuristic is proposed based on artificial intelligence (Algorithm A*) and linear programming (simplex method): the algorithm A* has been modified to better suit the purpose and its evaluation function is the simplex method.

The proposed algorithm generates all the possible prefixes of the optimal sequence of transitions, it computes the cycle time of the fastest elemental T-semiflow whose support contains the alphabet of such sequences, it eliminates prefixes that do not lead to desired solution, it eliminates circuits when there exist better solutions in the prefixes, and it explores the prefixes which better suit the objective.

Algorithm 4.2 *Finding an elemental T-semiflow whose cycle time fits into a constant ξ.*

Phase 1: Initializing the variables and seeding the seeds of all possible feasible sequences.

1. $U_d = 1$
   Define U_d as the unit delay. Note that every timed transition is a U_d multiple.
2. For every transition t_i, create a set $\eta_i = \{t_j \mid t_j \in p_j \bullet \wedge p_j \in t_i \bullet\}$
   The set η_i contains transitions t_j such that t_j is a concatenable transition for a sequence whose last transition is t_i.
3. $\Sigma = \Psi = \{\}$
   Create two empty sets Σ and Ψ. Where Σ contains prefixes of sequences σ_i such that σ_i is not a circuit, and Ψ contains sequences ψ_j such that ψ_j is a circuit.
4. $\forall t_i \in T, \;\; \sigma_i = t_i \mid \sigma_i \in \Sigma$
   For every transition t_i, create a sequence $\sigma_i = t_i \in \Sigma$. This step seeds the prefixes of all possible sequences in the net.

Phase 2: Eliminating sequences such that their fastest elemental T-semiflow was previously found.

5. For every sequence $\sigma_l \in \Psi$
   (a) For every sequence $\sigma_k \in \Sigma$
      i. If $|\sigma_l| = |\sigma_k|$ and $Prefix(\sigma_l, |\sigma_l| - 1) = Prefix(\sigma_k, |\sigma_k| - 1)$ then $\Sigma = \Sigma \setminus \{\sigma_k\}$

Phase 3: Computing the cycle time of the fastest T-semiflow of the explored sequences and circuits

6. For every sequence $\sigma_i \in \Sigma \cup \Psi$, using the simplex method, to associate a nonnegative number τ_i representing the time of the fastest T-semiflow X_i such that $\forall t_i | t_i \in \sigma_i \Rightarrow t_i \in \langle X_i \rangle$.

Phase 4: Eliminating sequences that do not deal with the objective and finding out a solution.

7. For every sequence $\sigma_i \in \Sigma$

   (a) If $\tau_i > \xi$, then $\Sigma = \Sigma \setminus \{\sigma_i\}$.
   If the cycle time τ_i of the of the fastest T-semiflow X_i such that $\forall t_i | t_i \in \sigma_i \Rightarrow t_i \in \langle X_i \rangle$ is greater than ξ, then the sequence σ_i is erased from Σ since such prefix cannot be the prefix of the optimal sequence.

   (b) If $\tau_i \leq \xi$ and $\xi - \tau_i < U_d$, then σ_i is the optimal sequence and the algorithm finishes
   If the cycle time τ_i is smaller than ξ and the difference between ξ and τ_i is smaller than unit delay U_d, then the fastest T-semiflow X_i such that $\forall t_i | t_i \in \sigma_i \Rightarrow t_i \in \langle X_i \rangle$ is the T-semiflow which better fist into ξ, and the algorithm finishes.

Phase 5: Eliminating the circuits when there exist better solutions and finding out a solution.

8. Assign to τ_g the greatest value of τ_i of the sequences $\sigma_i \mid \sigma_i \in \Sigma \cup \Psi$.
   The value of τ_g defines the cycle time of the prefix or sequence which better fits into ξ.

9. For every $\sigma_i \in \Psi$, if $\tau_i < t_g$ then $\Psi = \Psi \setminus \sigma_i$.
   If there exist a circuit σ_i whose τ_i better fits than the one of other circuit σ_j, then the circuit σ_j is erased from Ψ since there exist a better circuit σ_i.

10. Assign to τ_Σ the greatest value of τ_i of the sequences $\sigma_i \mid \sigma_i \in \Sigma$.
    The value of τ_Σ defines time of the slowest cycle time of the prefix or sequences in Σ.

Phase 6: Exploring the sequences which better suit the objective.

11. For the sequences $\sigma_i \in \Sigma$ which $\tau_i = \tau_\Sigma$
    This step selects the prefix $\sigma_i \in \Sigma$ whose associated value τ_i is the greatest,

    (a) If $\mathcal{A}(\sigma_i) \cap \eta_k \neq \{\}$, where t_k is the last transition in the sequence σ_i,

       i. then for every $t_j \in \eta_k$, $\Sigma = \Sigma \cup \{\sigma_i t_j\}$

       ii. else $\Psi = \Psi \cup \{\sigma_i\}$

    (b) $\Sigma = \Sigma \setminus \{\sigma_i\}$

12. If $|\Sigma| = 0$ then

    (a) Assign to τ_Ψ the greatest value of τ_i of the sequences $\sigma_i \mid \sigma_i \in \Psi$.
    The value of τ_Ψ defines time of the slowest cycle time of the sequences in Ψ.

    (b) For every sequence $\sigma_i \in \Psi$

        i. If $\tau_i = \tau_\Psi$ then σ_i is the T-semiflow X_i such that $\forall t_i | t_i \in \sigma_i \Rightarrow t_i \in \langle X_i \rangle$ is the better T-semiflow and the algorithm finishes

13. Go to step 5

Example 4.7 *A gross description of the previous algorithm is depicted using the state machine shown in figure 4.7. Where the delay vector is* $D = [0, 8, 7, 5, 9, 6, 7, 3, 4, 2]$ *and* $\xi = 17$

First, the algorithm explores all the prefixes of the possible optimal sequence and computes the cycle time of the fastest T-semiflows which include the prefix in its support, figure 4.8 shows this step.

Once all the prefixes are evaluated with the simplex method and no result is the optimal one, then, the heuristic explores the prefixes which better suit with the objective. Since the objective is 17 then the nodes to explore are t_2 *and* t_5.

The second exploration is shown in figure 4.9. In this exploration, the prefix t_2, t_6 *is the first prefix found which belongs to a T-semiflow whose fastest cycle time is 17. Therefore, the T-semiflow computed in the evaluation of such prefix is the desired one and the algorithm finishes.*

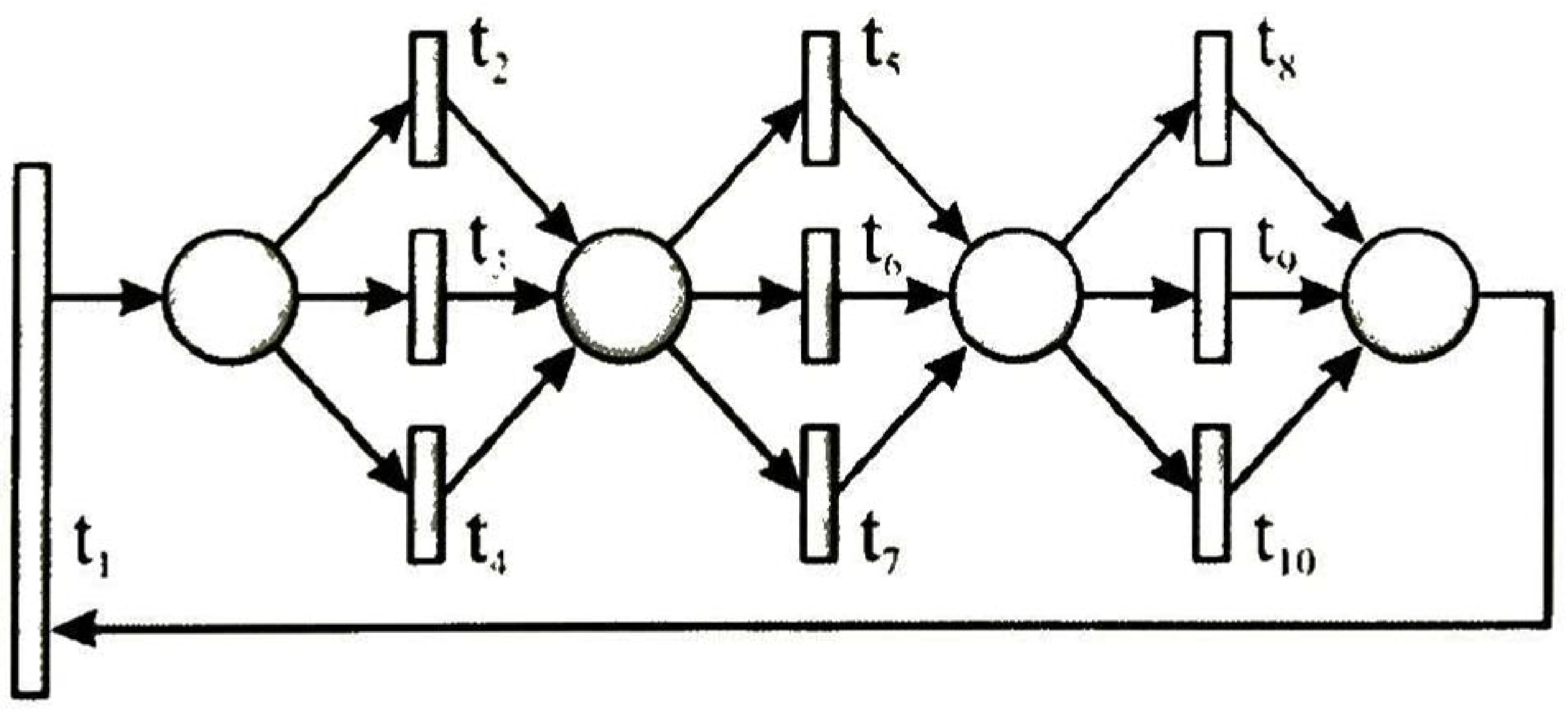

Figure 4.7: Abstract model of a DEDS.

The use of the algorithms 4.2 and 4.1 search a T-cover whose slowest T-semiflow is as fast as possible, but it does not find out the optimal T-cover where the standard deviation of the cycle time of the T-semiflows is minimal. In order to improve results, the procedure can be repeated replacing the value of the unit delay in step 1 in algorithm 4.2 by the standard deviation of the cycle time of the T-semiflows in the previous found T-cover.

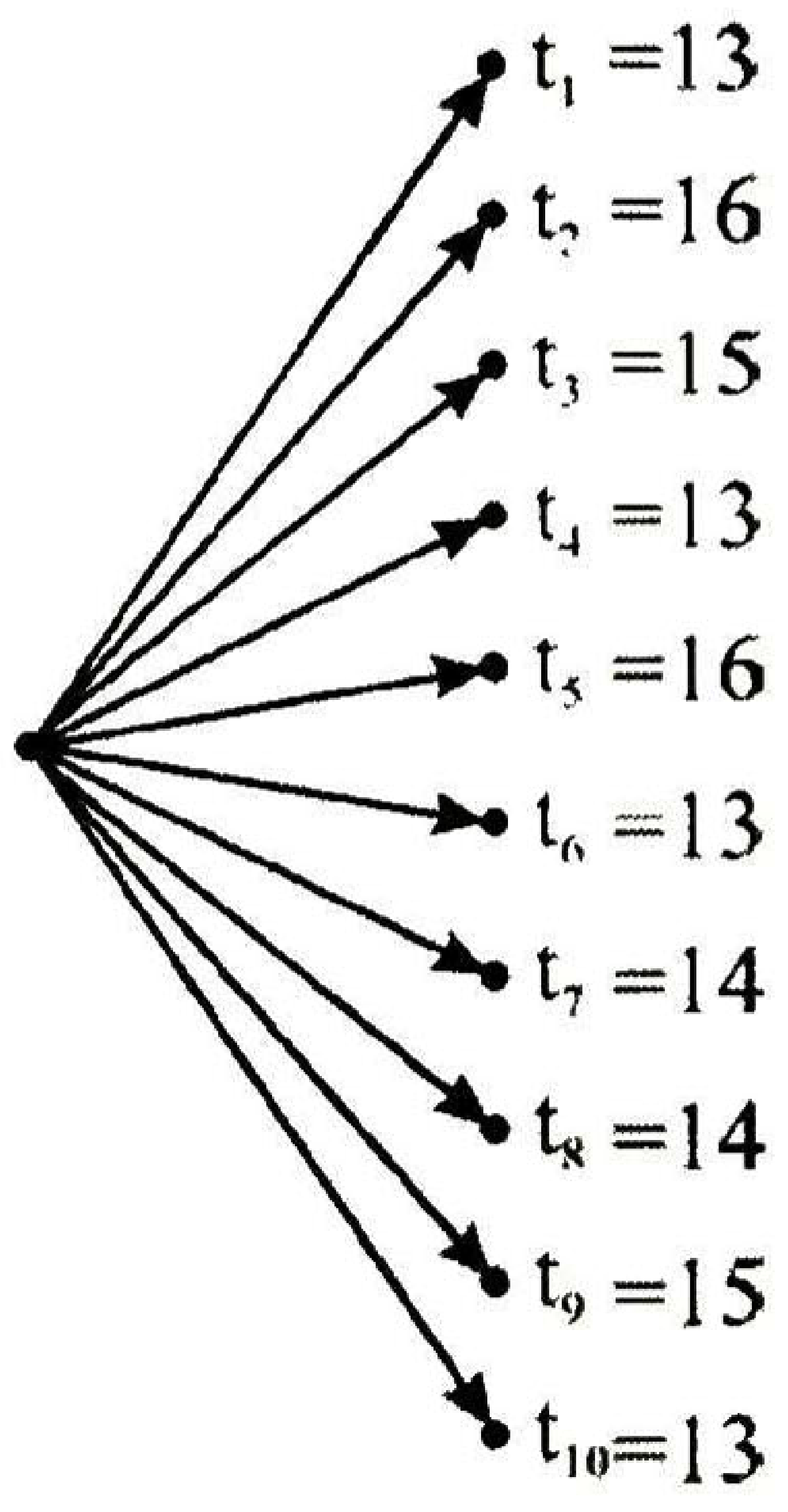

Figure 4.8: First exploration.

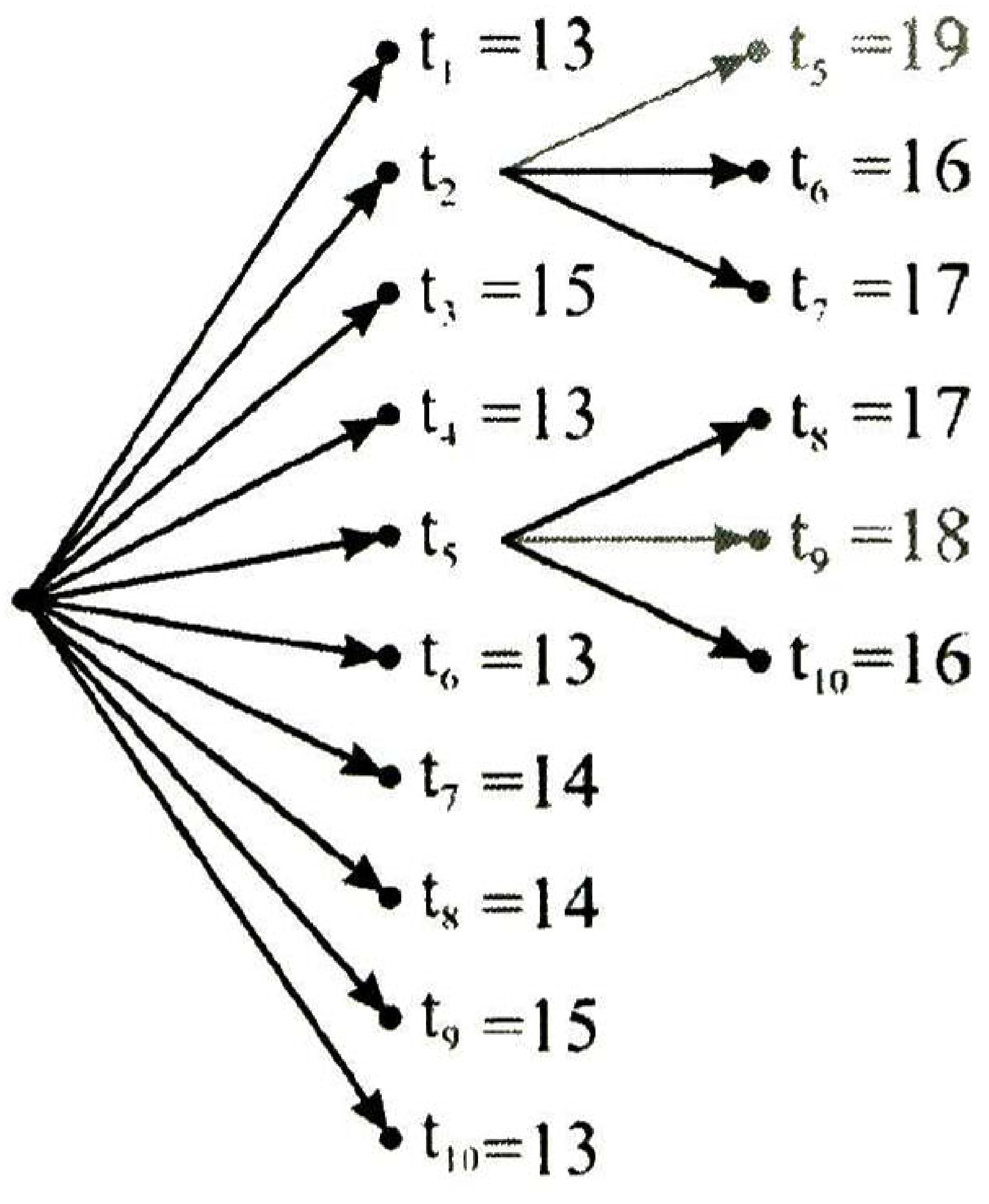

Figure 4.9: Second exploration.

Below is proposed and heuristic to improve the results of the algorithms 4.2 and 4.1.

Algorithm 4.3

1. $U_d = \frac{C}{n}$, $U_p = \infty$, $U_{2p} = \infty$
   Assign to the unit delay U_d the value of the average cycle time of the T-semiflows. Define U_p and U_{2p} as the previous and the second previous unit delay; since at the beginning there is no previous unit delay, infinity is assigned to the value of U_p and U_{2p}.

2. Apply the algorithms 4.2 and 4.1.

3. $U_{2p} \leftarrow U_p$
   $U_p \leftarrow U_d$
   $Ud \leftarrow$ the value of the standard deviation of the cycle time of the T-semiflows in the previous found T-cover.

4. if $U_p < U_d$ or $U_{2p} = U_d$

   (a) then, the second previous found T-cover is better found and the algorithm finishes.

   (b) else go to step 2.

4.3 Application to the Sales Company

Since the company policy establish that for every conflict the number of firing proportion must be equal, then the computed visit ratio is $vr = [t_{1,1}, t_{1,2}, t_{1,3}, t_{1,4}, t_{1,5}, t_{1,6}, t_{1,7}, t_{1,8}, t_{1,9}, t_{1,10}, t_{1,11}, t_{1,12}, t_{2,1}, t_{2,2}, t_{2,3}, t_{2,4}, t_{2,5}, t_{2,6}, t_{2,7}, t_{2,8}, t_{2,9}\ t_{2,10}, t_{2,11}, t_{2,12}, t_{13}] = [2, 2, 2, 2, 2, 2, 4, 1, 1, 4, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6]$. Figure 4.10 shows graphically the visit ratio.

The model of the company is a synchronised state machines Petri net, and its two synchronised state machines are shown in figures 3.4 and 3.5 respectively.

The analysis will be achieved partially. First, for the state machine shown in figure 3.4, and later for the other shown in figure 3.5.

First State Machine

The visit ratio for the first state machine is $v_{r1} = [t_{1,1}, t_{1,2}, t_{1,3}, t_{1,4}, t_{1,5}, t_{1,6}, t_{1,7}, t_{1,8}, t_{1,9}, t_{1,10}, t_{1,11}, t_{1,12}, t_{13}] = [2, 2, 2, 2, 2, 2, 4, 1, 1, 4, 1, 1, 6]$.

Applying the heuristic 4.3 for the first state, the following T-cover was found:

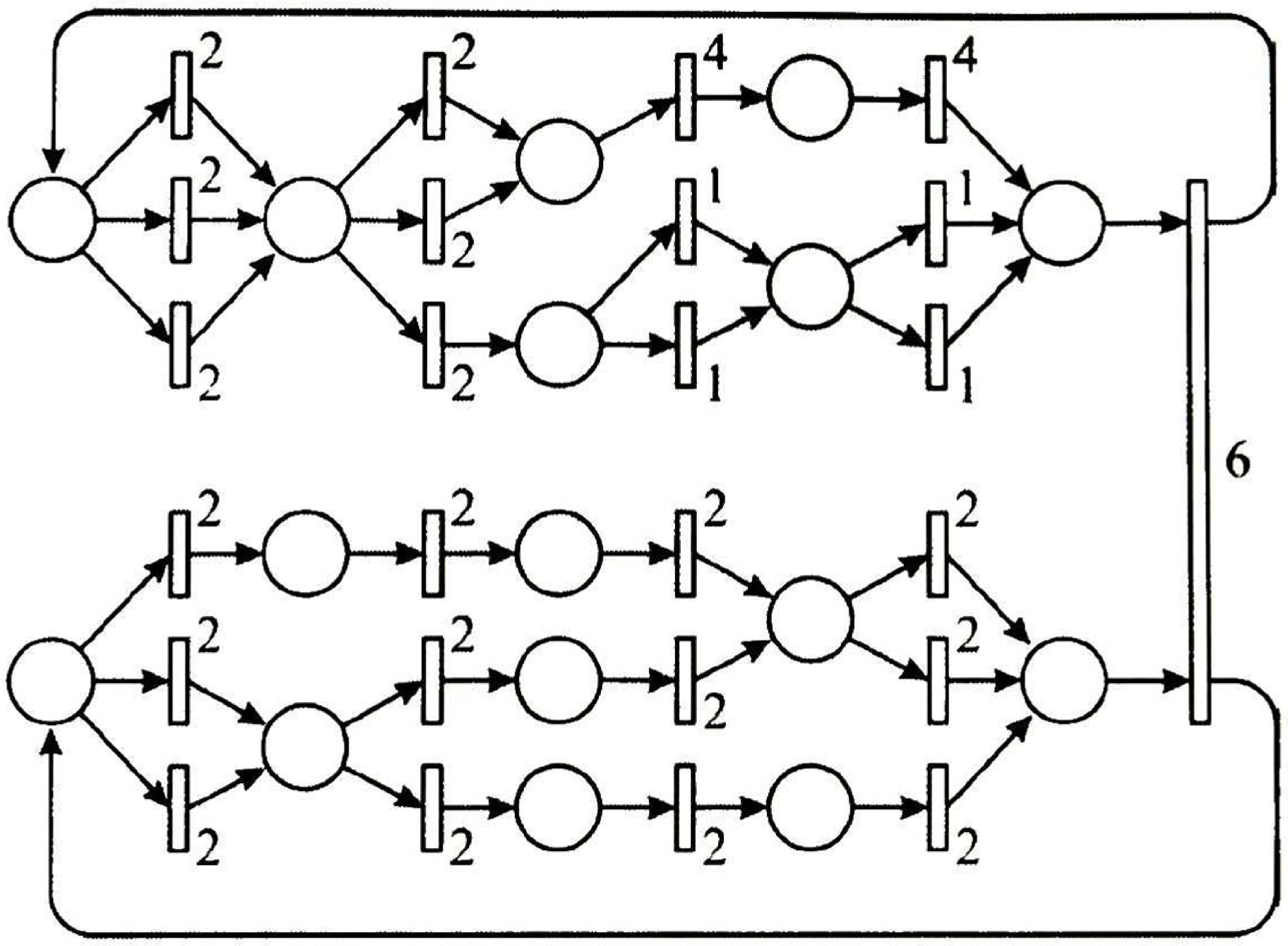

Figure 4.10: Visit ratio of the sales company model.

C_r	n_r	ξ_r	$\langle \mathcal{T}_n \rangle$	c_i
56.5	6	9.41	$\{t_{1,3}, t_{1,5}, t_{1,7}, t_{1,10}, t_{13}\}$	9
47.5	5	9.5	$\{t_{1,1}, t_{1,6}, t_{1,8}, t_{1,12}, t_{13}\}$	9.5
38	4	9.5	$\{t_{1,3}, t_{1,5}, t_{1,7}, t_{1,10}, t_{13}\}$	9
29	3	9.66	$\{t_{1,2}, t_{1,6}, t_{1,9}, t_{1,11}, t_{13}\}$	7.5
21.5	2	10.75	$\{t_{1,1}, t_{1,4}, t_{1,7}, t_{1,10}, t_{13}\}$	10.5
11	1	11	$\{t_{1,2}, t_{1,4}, t_{1,7}, t_{1,10}, t_{13}\}$	11

Second State Machine

The visit ratio for the second state machine is $v_{r2} = [t_{2,1}, t_{2,2}, t_{2,3}, t_{2,4}, t_{2,5}, t_{2,6}, t_{2,7}, t_{2,8}, t_{2,9}, t_{2,10}, t_{2,11}, t_{2,12}, t_{13}] = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6]$. Since the visit ratio is a linear combination of elemental T-semiflows, then it will be divided by the greatest common divisor (2) in order not to find an elemental T-semiflow more than once. Therefore, the visit ratio which will be used to compute the elemental T-cover is $v_{r2} = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3]$.

Applying the heuristic 4.3 for the second state, the following T-cover was found:

C_r	n_r	ξ_r	$\langle \mathcal{T}_n \rangle$	c_i
31	3	10.3	$\{t_{2,3}, t_{2,6}, t_{2,9}, t_{2,12}, t_{13}\}$	8
23	2	11.5	$\{t_{2,2}, t_{2,5}, t_{2,8}, t_{2,11}, t_{13}\}$	10.5
12.5	1	12.5	$\{t_{2,1}, t_{2,4}, t_{2,7}, t_{2,10}, t_{13}\}$	12.5

Since the greatest common divisor of the visit ratio for the second state machine is 2 and the number of T-semiflows in the T-cover is 3, then the 6 T-semiflows of the global T-cover are now partially covered on the second state machine.

Building the global T-cover

Since the partial T-covers of the first and second state machines is done, then the support of the T-semiflows of the global T-cover is the union of the supports of one T-semiflow of the first state machine and other T-semiflow of the second state machine. In order to avoid delays, the synchronization of the elemental T-semiflows of the partial T-covers is achieved by synchronising the slowest remaining T-semiflow of each partial T-cover. Therefore, the global T-cover is represented by the following T-semiflows:

$\mathcal{T}_n$	$\langle \mathcal{T}_n \rangle$	$c_{i,1}$	$c_{i,2}$	c_i	$Delay_1$	$Delay_2$	$Delay$
$\mathcal{T}_1$	$\{t_{1,2}, t_{1,6}, t_{1,9}, t_{1,11}, t_{2,3}, t_{2,6}, t_{2,9}, t_{2,12}, t_{13}\}$	7.5	8	8	0.5	0	0.5
$\mathcal{T}_2$	$\{t_{1,3}, t_{1,5}, t_{1,7}, t_{1,10}, t_{2,3}, t_{2,6}, t_{2,9}, t_{2,12}, t_{13}\}$	9	8	9	0	1	1
$\mathcal{T}_3$	$\{t_{1,3}, t_{1,5}, t_{1,7}, t_{1,10}, t_{2,2}, t_{2,5}, t_{2,8}, t_{2,11}, t_{13}\}$	9	10.5	10.5	1.5	0	1.5
$\mathcal{T}_4$	$\{t_{1,1}, t_{1,6}, t_{1,8}, t_{1,12}, t_{2,2}, t_{2,5}, t_{2,8}, t_{2,11}, t_{13}\}$	9.5	10.5	10.5	1	0	1
$\mathcal{T}_5$	$\{t_{1,1}, t_{1,4}, t_{1,7}, t_{1,10}, t_{2,1}, t_{2,4}, t_{2,7}, t_{2,10}, t_{13}\}$	10.5	12.5	12.5	2	0	2
$\mathcal{T}_6$	$\{t_{1,2}, t_{1,4}, t_{1,7}, t_{1,10}, t_{2,1}, t_{2,4}, t_{2,7}, t_{2,10}, t_{13}\}$	11	12.5	12.5	1.5	0	1.5

Since there exist a synchronization in the abstract model, the cycle time cannot be computed as a weighted addition of the transition's delay. Since no T-invariant can be executed in parallel, the cycle time of the global T-cover is equal to the addition of the cycle time of the T-semiflows.

Global cycle time = 63 time units
Total delay= 7.5 time units

The optimal schedule is such that it minimizes the delays in the system. The optimal cycle time can be reached by executing T-semiflow as soon as they become enabled.

4.4 Application to the Manufacturing Company

The cycle time performance can be measured easier on the abstract model since there exist less nodes and computations can be achieved faster.

Since the company policy establish that for every conflict the number of firing proportion must be equal, then the computed visit ratio is $vr = [t_a, t_5, t_6, t_7, t_8, t_b, t_c, t_d, t_{16}, t_{17}, t_{18}, t_{19}, t_{20}, t_{27}, t_{28}, t_{29}, t_f, t_g, t_h, t_{35}, t_{36}]$ = [12, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4]. Figure 4.11 shows graphically the visit ratio.
The abstract model belongs to the synchronised state machines Petri nets class, and its two synchronised state machines are shown in figures 4.12 and 4.13 respectively.
The analysis will be achieved partially. First, for the state machine shown in figure 4.12, and after for the other shown in figure 4.13.

First State Machine

The visit ratio for the first state machine is $v_{r1} = [t_a, t_5, t_6, t_7, t_8, t_b, t_c, t_d, t_{16}, t_{17}, t_{18}, t_{19}, t_{20}]$ = [12, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]. Since the visit ratio is a linear combination of elemental T-semiflows, then it will

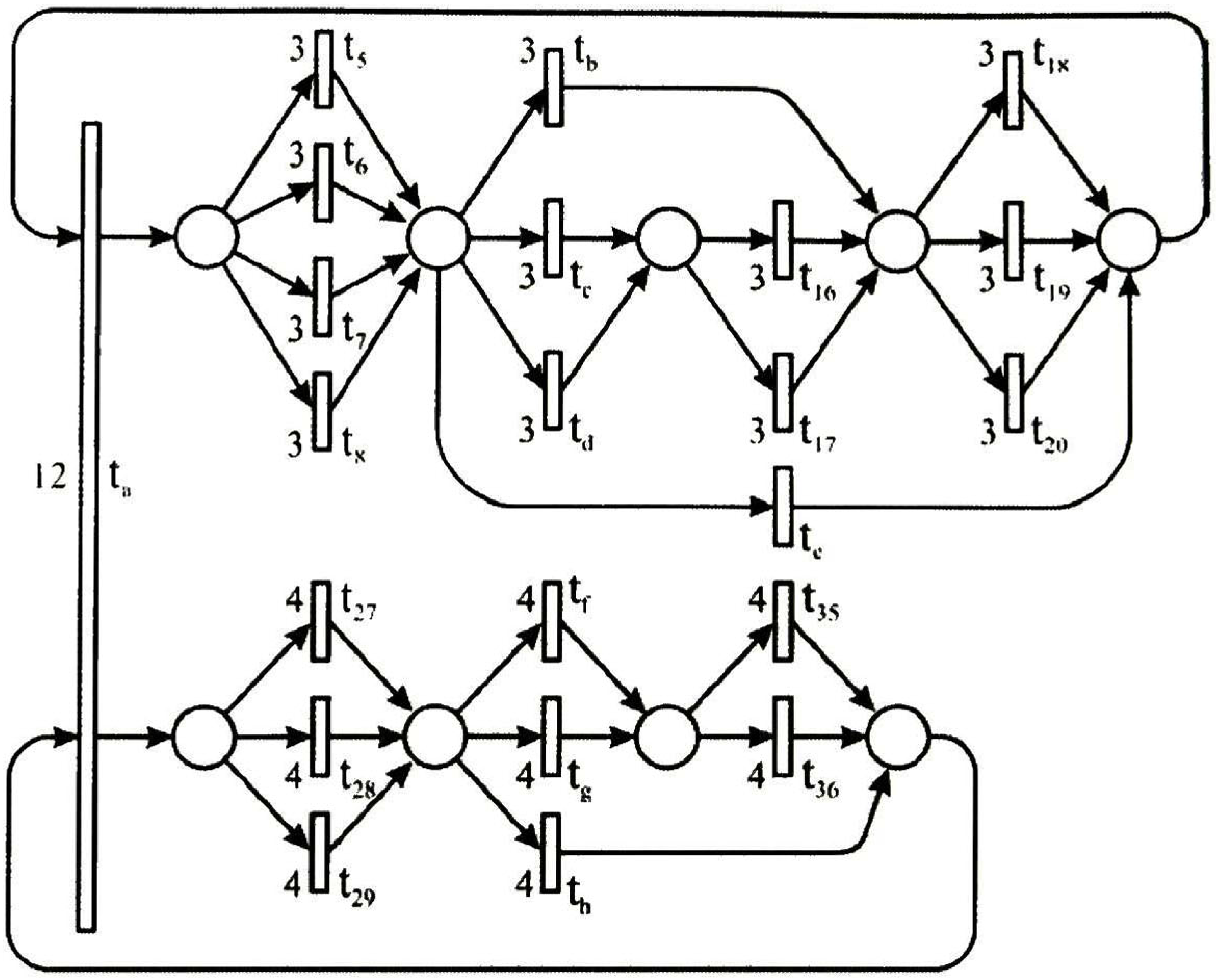

Figure 4.11: The visit ratio.

be divided by the greatest common divisor (3) in order not to find an elemental T-semiflow more than once. Therefore, the visit ratio which will be used to compute the elemental T-cover is v_{r1} = [4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1].

Applying the heuristic 4.3 for the first state, the following T-cover was found:

C_r	n_r	ξ_r	$\langle T_n \rangle$	c_i
184	4	46	$\{t_a, t_5, t_c, t_{17}, t_{18}\}$	44
140	3	$\frac{140}{3}$	$\{t_a, t_6, t_b, t_{20}\}$	44
96	2	48	$\{t_a, t_8, t_e\}$	39
57	1	57	$\{t_a, t_7, t_d, t_{16}, t_{19}\}$	57

Since the greatest common divisor of the visit ratio for the first state machine is 3 and the number of T-semiflows in the T-cover is 4, then the 12 T-semiflows of the global T-cover are now partially covered on the first state machine.

Second State Machine

The visit ratio for the second state machine is v_{r2} = [t_a, t_{27}, t_{28}, t_{29}, t_f, t_g, t_h, t_{35}, t_{36}] = [12, 4, 4, 4, 4, 4, 4, 4, 4]. Since the visit ratio is a linear combination of elemental T-semiflows, then it will be divided by the greatest common divisor (4) in order not to find an elemental T-semiflow more than once. Therefore, the visit ratio which will be used to compute the elemental T-cover is v_{r2} = [3, 1, 1, 1, 1, 1, 1, 1, 1].

Applying the heuristic 4.3 for the second state, the following T-cover was found:

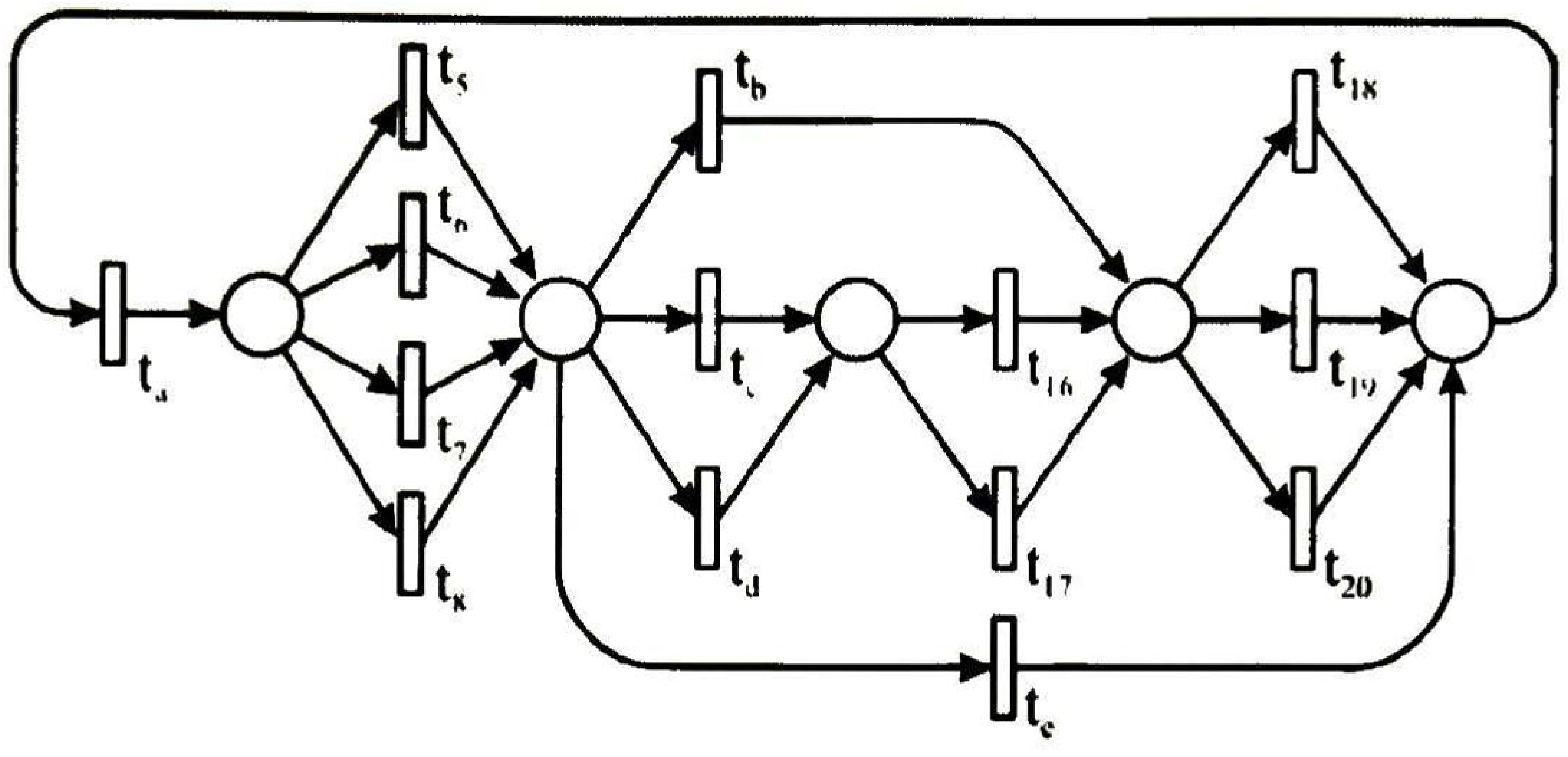

Figure 4.12: First synchronized state machine on the abstract model.

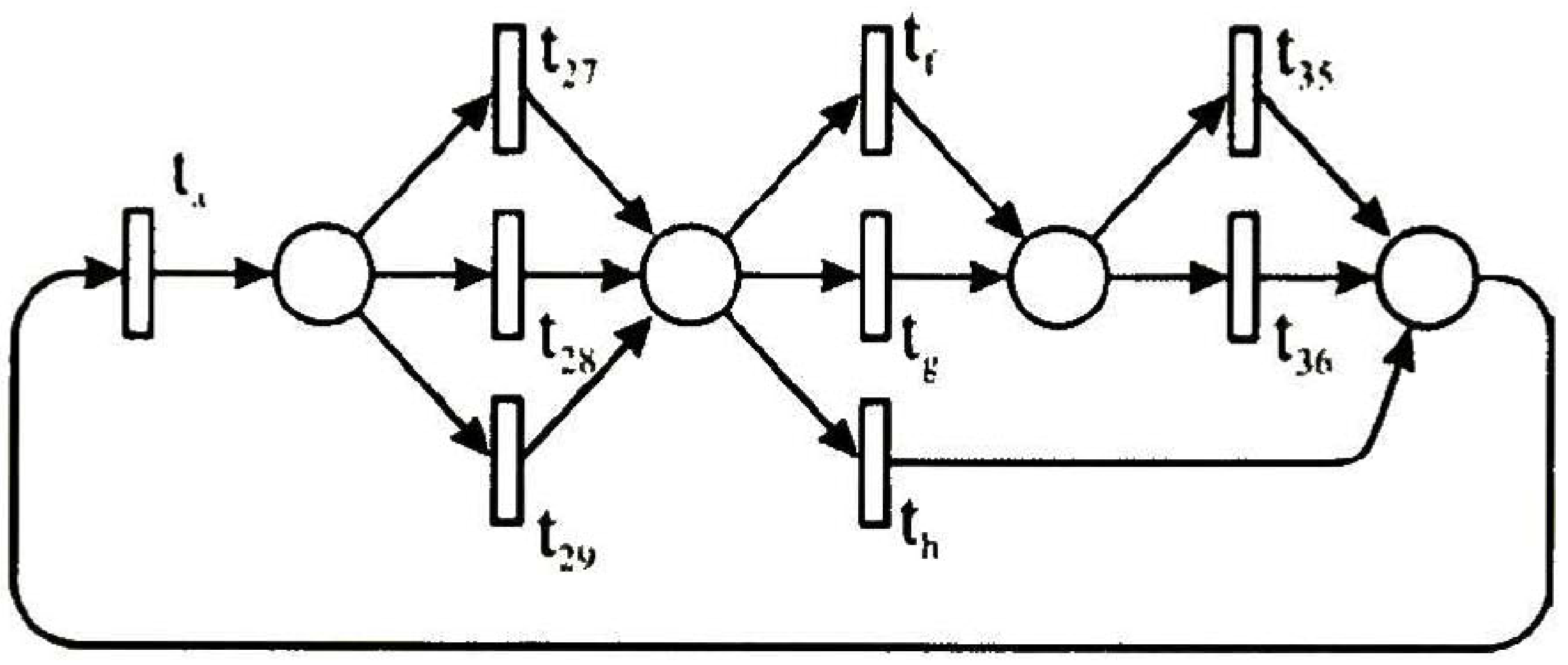

Figure 4.13: Second synchronized state machine on the abstract model.

C_r	n_r	ξ_r	$\langle T_n \rangle$	c_i
71	3	$\frac{71}{3}$	$\{t_a, t_{27}, t_f, t_{36}\}$	23
48	2	24	$\{t_a, t_{28}, t_g, t_{35}\}$	22
26	1	26	$\{t_a, t_{29}, t_h\}$	26

Since the greatest common divisor of the visit ratio for the second state machine is 4 and the number of T-semiflows in the T-cover is 3, then the 12 T-semiflows of the global T-cover are now partially covered on the second state machine.

Building the global T-cover

Since the partial T-covers of the first and second state machines is done, then the support of the T-semiflows of the global T-cover is the union of the supports of one T-semiflow of the first state machine and other T-semiflow of the second state machine. In order to avoid delays, the synchronization of the elemental T-semiflows of the partial T-covers is achieved by synchronising the slowest remaining T-semiflow of each partial T-cover. Therefore, the global T-cover is represented by the following T-semiflows:

$\mathcal{T}_n$	$\langle\mathcal{T}_n\rangle$	$c_{i,1}$	$c_{i,2}$	c_i	$Delay_1$	$Delay_2$	$Delay$
$\mathcal{T}_1$	$\{t_a, t_8, t_e, t_{27}, t_f, t_{36}\}$	49	23	49	0	26	17
$\mathcal{T}_2$	$\{t_a, t_8, t_e, t_{27}, t_f, t_{36}\}$	49	23	49	0	26	17
$\mathcal{T}_3$	$\{t_a, t_8, t_e, t_{27}, t_f, t_{36}\}$	49	23	49	0	26	17
$\mathcal{T}_4$	$\{t_a, t_5, t_c, t_{17}, t_{18}, t_{27}, t_f, t_{36}\}$	45	23	45	0	22	22
$\mathcal{T}_5$	$\{t_a, t_5, t_c, t_{17}, t_{18}, t_{28}, t_g, t_{35}\}$	45	23	45	0	22	21
$\mathcal{T}_6$	$\{t_a, t_5, t_c, t_{17}, t_{18}, t_{28}, t_g, t_{35}\}$	45	23	45	0	22	21
$\mathcal{T}_7$	$\{t_a, t_6, t_b, t_{20}, t_{28}, t_g, t_{35}\}$	41	23	41	0	18	21
$\mathcal{T}_8$	$\{t_a, t_6, t_b, t_{20}, t_{28}, t_g, t_{35}\}$	41	23	41	0	18	21
$\mathcal{T}_9$	$\{t_a, t_6, t_b, t_{20}, t_{29}, t_h\}$	41	24	41	0	17	18
$\mathcal{T}_{10}$	$\{t_a, t_7, t_d, t_{16}, t_{19}, t_{29}, t_h\}$	57	24	57	0	33	31
$\mathcal{T}_{11}$	$\{t_a, t_7, t_d, t_{16}, t_{19}, t_{29}, t_h\}$	57	24	57	0	33	31
$\mathcal{T}_{12}$	$\{t_a, t_7, t_d, t_{16}, t_{19}, t_{29}, t_h\}$	57	24	57	0	33	31

Since there exists a synchronization transition in the abstract model, the cycle time can not be computed as a weighted addition of the transition's delay. Since no T-invariant can be executed in parallel, the cycle time of the global T-cover is equal to the addition of the cycle time of the T-semiflows.

Global cycle time = 576 time units

Total delay= 296 time units

The optimal schedule is such that it minimizes the delays in the system. The optimal cycle time can be reached by executing T-semiflow as soon as they become enabled.

4.5 Conclusions

The use of the mathematical formalism of the Petri nets allows to use and combine many different schedule techniques and heuristics; providing new scheduling methods. Petri nets facilitate a comprehensive graphical environment and reduce the complexity of the analysis of scheduling algorithms.

The problem of finding out an optimal schedule for a system whose model is represented by a free choice Petri net resides in finding out a T-cover whose T-invariant have P-semiflows with the same cycle time (in order to avoid looseness). If the complexity of a system represented by a free choice Petri net with an unique P-cover and multiple T-covers is NP-Complete, then the complexity of a Petri net that do not have a unique T-cover neither a unique P-cover become more complex. Therefore, the problem of finding out an optimal schedule of a system whose model is represented by a free choice Petri net with multiple P-covers and multiple T-covers can be interpreted as a minimax problem.

In some cases, the T-invariants analysis leads to the fact that the minimum cycle time of a free choice Petri net can be computed by the weighted addition of the T-invariants cycle times. The previous conclusion cannot be generalized for any free choice Petri net. If the optimal cycle time is computed as the weighted addition of the T-invariants execution times whose linear combination is the visit ratio v_r, then in each time instant τ a unique T-invariant $\mathcal{T}_i$ is being executed. The converse conclusion is not true, if a unique t-invariant is $\mathcal{T}_i$ been executed in each time instant τ, then the optimal cycle time is not necessarily reached.

Even in the case where there is no T-invariant working in parallel, decisions depends on maximum operations, causing a different cycle time for each T-cover. Therefore, even when the cycle time of a Petri net can be represented by a weighted addition of the T-invariants execution times, it cannot be known if the cycle time of a T-cover is the optimal cycle time since the cycle time of a T-invariant is equal to the cycle time of its slowest P-semiflow.

The understanding of the functionality of a complex system is so much easier as its description is analysed in a more structured and progressive manner. Therefore, in many complex applications, concurrent systems are described synchronizing the descriptions of different subsystems. The main advantage of models description is the complete freedom to analyse systems, considering a complex system as a composition of independent subsystems, where each subsystem can be modelled separately. Description methods facilitate or avoid the analysis of the complex model since the analysis is made over the subsystems.

In the general case, the proposed heuristic does not finds out the optimal T-cover, even when it does in most of the cases.

Chapter 5

CONCLUSIONS

5.1 Introduction

The scheduling problem is an optimisation problem for discrete event systems. To solve this problem, there exist different methods and tools. The two methods used in this work to deal with the scheduling problem are operations research and artificial intelligence.

Operations research methods can solve some specific scheduling problems by the system representation as a set of linear equation. In the practice is difficult to apply these methods on discrete event systems, either for the combinatory quantity of states or for the restriction to entire values of the system parameters.

Artificial intelligence based methods represent systems describing its behaviour and the inner relationships. These methods find out a good solution in a reasonable amount of time, considering that the solution fulfils the stated constraints. In most of the cases it is not possible to prove if the solution is the optimal one.

Independently of the used method, the difficulties to solve a scheduling problem depend on the accuracy of the model used for the analysis and on the reliability of the input data required. Therefore, Petri nets are chosen a as modelling tool since they capture DEDS features through a strong mathematical support allowing analysing properties and characterizing systems from a formal model.

The use of the mathematical formalism of the Petri nets provides the flexibility of using and combining different methods; providing new and better heuristics for specific applications. Petri nets facilitate a comprehensive graphical environment and reduce the algorithms complexity for scheduling analysis. This takes advantage of having a unique model; simplifying the analysing of systems properties, and simplifying the computation of the optimal schedule.

The scheduling problem for an arbitrary free choice Petri net is an NP-complete problem, but this subclass include more subclasses such that their schedule is not an NP-complete problem. Therefore, the study of the scheduling complexity on free choice Petri nets yields to develop heuristics and algorithms for specific subclasses of free choice Petri nets such as synchronised state machines.

The problem of finding out an optimal schedule for a system whose model is represented by a free choice Petri net resides in finding out a T-cover whose T-invariants have P-semiflows with the same cycle time (in order to avoid looseness). If the complexity of a system represented by a free choice Petri net with an unique P-cover and multiple T-covers is NP-Complete, then the complexity of a Petri net that do not have a unique T-cover neither a unique P-cover become more complex. Therefore, the problem of finding out an optimal schedule of a system whose model is represented by a free choice Petri net with multiple P-covers and multiple T-covers can be interpreted as a minimax problem.

In some cases, the T-invariants analysis leads to the fact that the minimum cycle time of a free choice Petri net can be computed by the weighted addition of the T-invariants cycle times. The previous conclusion cannot be generalized for any free choice Petri net. If the optimal cycle time is computed as the weighted addition of the T-invariants execution times whose linear combination is the visit ratio v_r, then in each time instant τ a unique T-invariant $\mathcal{T}_i$ is being executed. The converse conclusion is not true, if a unique T-invariant $\mathcal{T}_i$ is been executed in each time instant τ, then the optimal cycle time is not necessarily reached, since the topology.

Even in the case where there is no T-invariant working in parallel, decisions depends on maximum operations, causing a different cycle time for each T-cover. Therefore, even when the cycle time of a Petri net can be represented by a weighted addition of the T-invariants execution times, it cannot be known if the cycle time of a T-cover is the optimal cycle time since the cycle time of a T-invariant is equal to the cycle time of its slowest P-semiflow.

5.2 Advantages

The understanding of the functionality of a complex system is so much easier as its description is analysed in a more structured and progressive manner. Therefore, in many complex applications, concurrent systems are described synchronizing the descriptions of different subsystems. The main advantage of models description is the complete freedom to analyse systems, considering a complex system as a composition of independent subsystems, where each subsystem can be modelled separately. Description methods facilitate or avoid the analysis of the complex model since the analysis is made over the subsystems.

The heuristic proposed here in is applied to the subclass named synchronised state machines; this subclass of free choice Petri net is capable to model a large quantity of DEDS. This heuristic can be easily extended to complex systems whose subsystems are modelled not only by state machines. These complex systems may include subsystems modelled by different subclasses: marked graphs, state machines or other subsystems whose optimal schedule can by computed through polynomial algorithms.

The use of the A* artificial intelligence technique where the evaluation function is the simplex method allow to optimise more than one factor, increasing the heuristic optimisation power.

Previous results stated that optimizing an isolated subsystem can not yield to the optimization of the global system, since there exists a relationship between parallel subsystems. The results exposed here in prove that it is possible to obtain a global optimization by optimizing isolated subsystems through a statistic perspective.

5.3 Future Work

Scheduling in free choice Petri nets is a wide space of study. Due to the NP-completeness of the scheduling in free choice Petri nets, there exist no results for systems modelled by Petri nets capable to execute more than two elemental T-semiflows at the same time.

Parallelism in scheduling is a very difficult problem. The scheduling analysis made for systems modelled by free choice Petri nets have been developed for systems that do not have T-semiflows working in parallel, excluding free choice B* and C* Petri nets.

There exists shortage in results of parallel scheduling explaining the reasons, or trying to find heuristics or subclasses of Petri nets with parallel branches or subsystems whose schedule can be computed in a reasonable amount of time.

However, the analysis for Petri nets where there exist T-semiflows working in parallel becomes more complex due to the relationship between such T-semiflows working in parallel. Therefore, the parallel scheduling problem becomes a very difficult problem (in most of the cases combinatorial).

Bibliography

[ABC84] M. Ajmone Marsan, G. Balbo, G. Conte, A class of of generalized stochastic Petri nets for performance analysis of multiprocessor systems, *ACM Transactions on Computer Systems*, 2(1), 93-122, May 1984.

[Cam90] J. Campos. *Performance Bounds for Synchronized Queueing Networks.* Tesis Doctoral, Univesidad de Zaragoza, María de Luna 3 E-50015 Zaragoza, España, 1990.

[Car84] J. Carlier, Ph. Chretienne, and C. Girault. Modelling scheduling with timed Petri nets. En G. Rozenberg, H. Genrich, y G. Roucairol, edithors, *Advances in Petri Nets 1984*, volume 188 of *Lecture Notes in Computer Sciences*, pages 62-82. Springer-Verlag, Berlin, Germany, 1988.

[Car88] J. Carlier, and Ph. Chretienne. Timed Petri net schedules. En G. Rozenberg, edithor, *Advances in Petri Nets 1988*, volume 340 of *Lecture Notes in Computer Sciences*, pages 62-84. Springer-Verlag, Berlin, Germany, 1984.

[CR83] J. Coolahan and J. Roussopoulos, Timing requirements for time-driven systems using augmented Petri nets, *IEEE Transactions on Software Engineering*, SE-9(5), 1983

[D&E95] Jorg Desel & Javier Esparza, *Free Choice Petri Nets,* Cambridge University Press, 1995.

[Ram74] C. Ramchandani, Analysis of asynchronousconcurrent systems by Petri nets, Project MAC, TR-120, M.I.T., Cambridge, MA, 1974.

[GJ79] M. Garey and D. Johnson. *Computer and Intractability: A guide to the Theory of NP-Completeness.* W.H. Freeman and Company, USA, 1979.

[Mag84] J. Magott. Performance Evaluation of Conncurrent Systems using Petri nets. Information Processing Letters, (18):7-13, January 1984.

[Mag87] J. Magott. New np-complete problems in performace evaluation of concurrent systems using Petri nets. *IEEE Transactions on Software Engineering*, SE-13(5):578-581. May 1987.

[MF76] P. Merlin and D. Farber, Recoverability of communication protocols - Implication of a theoretical study, *IEEE Transactions on Communication,* 1036-1043, Sept. 1976.

[Mol81] M. Molloy, On the integration of delay and throughput measures in distributed processing models, Ph.D. Thesis, UCLA, Los Angeles, CA, 1981.

[Mur89] T. Murata, Petri nets: Properties, analysis and applications, *Proceedings of the IEEE,* 77(4):541-580, 1989.

[Nat80] S. Natkin, Les reseaux de Petri stochastiques et leur application a I'evaluation des systemes informatiques, These de Docteur Ingegneur, Cnam, Paris, France, 1980.

[Jen81] K. Jensen, Colored Petri nets and the invariant method, Theoretical Computer Science, 14,317-336, 1981.

[Ram93] A. Ramírez. *Scheduling en Redes de Petri.* Tesis Doctoral, Universidad de Zaragoza, España. 1993.

[Ram93a] A. Ramírez. Optimal and suboptimal scheduling ind DEDS using Petri nets. In Proceedings of the 1993 IEEE International conference on Systems, Man and Cybernetics, pages 289-294, Letouquet, France, October 1993.

[Ram93b] A. Ramírez, J. Campos, y M. Silva. On optimal scheduling in DEDS. In *Proceeding of the 1993 IEEE International Conference on Robotics and Automation,* pages 405-409, Chicago, USA, August 1993.

[RH80] C. Ramamoorthy and G. Ho, Performace evaluation of asynchronous concurrent systems usong Petri nets, IEEE Transaction on Software Engineering, SE-6(5),440-449, 1980.

[Sil85] *Las Redes de Petri: en la Automática y en la Informática.* AC, Madrid, España, 1985.

[TV84] P.S. Thiagarajan and K. Voss. A fresh look at the free choice Petri nets. *Information and Control,* 61(2):85-113, May 1984.

[Wan98] Wang, Jiacun, 1998: Timed Petri Nets Theory and Application

[ZD88] Zhu and Denton 1988

Appendix A

MATHEMATICAL PRELIMINARIES

Standard definitions on set, numbers, sequences, vectors and matrices are used. The purpose of this section is to fix some additional notations.

Notation A.1 *Sets, number, relations*

Let X and Y be sets.
$X \subseteq Y$ if X is a subset of Y, including the case $X = Y$.
$X \subset Y$ denotes that X is a proper subset of Y, i.e., $X \subseteq Y$ and $X \neq Y$.
$X \setminus Y$ denotes the set of elements of X that do not belong to Y
$|X|$ denotes the cardinality of X.

Notation A.2 $\mathbb{N}$ *denotes the set of natural numbers including* 0.
$\mathbb{R}$ *denotes the set of real numbers including* 0.
$\mathbb{R}^+$ *denotes the set of positive real numbers including* 0.
$\mathbb{Q}$ *denotes the set of rational numbers.*

Sequences play a particularly important role in DEDS. Finite sequences are mostly considered, which are isomorphic to strings over an alphabet, but also infinite ones. The concatenation of two sequences is defined only if the first sequence is finite.

Notation A.3 *Sequences*

Let A be a set. A finite sequence on A is a mapping $\{1, ..., n\} \to A$, including the mapping $\epsilon : \emptyset \to A$, called the empty sequence. A finite sequence $\sigma : \{1, ..., n\}$ is represented by the string $a_1\ a_2 \qquad a_n$ of elements of A, where $a_i = \sigma(i)$ for $1 \leq i \leq n$. The length of σ is denoted by $|\sigma|$, and the length of ϵ is 0.

A finite sequence is a mapping $\sigma : \{1, 2, 3...\} \to A$. Where σ is written as $\sigma = a_1\ a_2\ a_3. \qquad$ and $a_i = \sigma(i)$ for $1 \leq i$.

If $\sigma = a_1\ a_2 \qquad a_n$ and $\tau = b_1\ b_2 \qquad b_m$ are finite sequences then the concatenation of σ and τ, denoted by $\sigma\tau$, is the sequence $a_1\ a_2 \qquad a_n\ b_1\ b_2 \qquad b_m$ of length $n + m$.

If $\sigma = a_1\ a_2 \quad\quad a_n$ is a finite sequence and $\tau = b_1\ b_2\ b_3. \quad$ is an infinite sequences then the concatenation of σ and τ is the infinite sequence $a_1\ a_2 \quad\quad a_n\ b_1\ b_2\ b_3.$

A sequence τ is a prefix of a sequence σ if either $\tau = \sigma$ or $\tau\ \sigma' = \sigma$ for some sequence σ'. Define $Prefix(\sigma, n)$ as a function that maps to the prefix of the sequence σ from the first element of σ until the n-th element of σ.

The alphabet of a sequence σ is the set $\mathcal{A}(\sigma) = \{\ a \in A \mid a = \sigma(i)$ for some $i\ \}$, i.e., the set of elements that appear in the sequence σ.

Notation A.4 *Vectors, matrices*

Given a finite set $A = \{a_1, ..., a_k\}$, every mapping X from A to $\mathbb{Q}$ can be represented by the vector $(\ X(a_1), \quad , X(a_k)\ )$. The mapping X and the vector $(\ X(a_1), \quad , X(a_k)\ )$ are not distinguished.

$X \cdot Y$ denotes the dot product of two vectors. Similarly, if $\mathbf{N}$ is a matrix and X is a vector, $X \cdot \mathbf{N}$ and $\mathbf{N} \cdot X$ denote the left and right products of X and $\mathbf{N}$ respectively. Different symbols for row and column vector are not used. In the expression $\mathbf{N} \cdot X$, the vector X is a column vector, even though it is written as $X = (\ X(a_1), \ldots, X(a_k)\ )$.

$X \geq Y$ $(X > Y)$ if $X(a) \geq Y(a)$ $(X(a) > Y(a))$ for every element a of A. The mapping which maps every element to 0 is called the null vector and is denoted by $\mathbf{0}$. A matrix containing only zero entries is also denoted by $\mathbf{0}$.

Appendix B

PETRI NETS PROPERTIES

As a mathematical tool, Petri nets exhibit a set of properties. These properties allow to the system designer to identify the presence or absence of specific properties that are translated into system's properties. Two types of properties can be distinguished, behavioural and structural properties. The behavioural properties are those which depend on the initial state or marking of a Petri net. The structural properties, in the other hand, do not depend on the initial marking of a Petri net, they depend on the topology or net structure.

B.1 Behavioural Properties

Behavioural or dynamic properties of systems depend on the initial marking. The changing of the initial marking may easily change some properties. Some of the most important behavioural properties are reachability, boundedness, and liveness.

B.1.1 Reachability

An important issue in designing DEDS's is if a system can reach a specific state.

Definition B.1 *A marking M_i is said to be reachable from an initial marking M_0 if there exist a transition firing sequence leading from M_0 to M_i. The reachability set $R(N, M_0)$ is the set of all markings reachable from the initial marking. The set of all possible firing sequences from M_0 is denoted by $L(N, M_0)$. A marking M_1 is said to be immediately reachable from M_0 if firing an enabled transitions in M_0 results in M_1.*

B.1.2 Boundedness and Safeness

In a Petri net, places are often used to represent information storage areas in communication and computer systems, product and tool storage areas in manufacturing systems, etc. It is important be able to determine whether proposed control strategies prevent from the overflows of these storage areas. The Petri net property which helps to identify the existence of overflows in the modelled systems is the concept of boundedness.

Definition B.2 *Boundedness*

A place p, of a Petri net N, is said to be k-bounded if and only if the number of tokens in p is always less or equal to k (k is a nonnegative integer number) for every reachable marking $M \mid M \in R(N, M_0)$.

Definition B.3 *Safeness*

A place p is said to be safe if and only if p is 1-bounded.

Definition B.4 *A Petri net $N = (P, T, Pre, Post)$ with an initial marking M_0 is k-bounded if, each place in P is k-bounded*

Notice that every bounded system has a finite set of reachable markings.

Example B.1 *The Petri net shown in Figure 2.3 is 1-bounded (safe).*

B.1.3 Liveness

The concept of liveness is closely related to the deadlock situation, which has been situated extensively in the context of computer operating systems. Different levels of liveness are introduced.

Definition B.5 *A transition t in a Petri net $N = (P, T, Pre, Post)$ with an initial marking M_0 is said to be:*

1. $L0$-live (or dead) if there is no firing sequence $\sigma \mid \sigma \in L(N, M_0)$ which enables t.
2. $L1$-live (potentially firable) if t can be fired at least once in some firing sequence $\sigma \mid \sigma \in L(N, M_0)$.
3. $L2$-live if t can be fired at least k times in some firing sequence $\sigma \mid \sigma \in L(N, M_0)$ given any positive integer k.
4. $L3$-live if t can be fired infinitely often in some firing sequence in $\sigma \mid \sigma \in L(N, M_0)$.
5. $L4$-live (or live) if t is $L1$-live (potentially firable) for every reachable marking $M \mid M \in R(N, M_0)$.

Definition B.6 *A Petri net $N = (P, T, Pre, Post)$ is said to be Lk-live, for an initial marking M_0, if every transition in the net is at least Lk-live, where $k = 0, 1, 2, 3, 4$.*

Definition B.7 *A Petri net $N = (P, T, Pre, Post)$ is said to be partially live if for every reachable marking there exists at least a firable transition and other transition that is not firable.*

Example B.2 *The Petri net shown in Figure B.1 is strictly L1-live since each transition can be fired exactly once in order of t_2, t_4, t_5, t_1 and t_3. The transitions t_1, t_2, t_3 and t_4 in Figure B.2 are $L0$-live (dead), $L1$-live, $L2$-live, $L3$-live, respectively.*

If (N, M_0) is a live system, then it is also said that M_0 is a live marking of N. Notice that a *non-live transition* is not necessary dead, but then there exists a firing sequence that makes it dead.

The liveness property encapsulates the concept of a system which will be able to run continuously, the importance of this property resides on its capability to characterise system deadlocks. In fact, if a Petri Net is live, then the system does not deadlock; due to all transitions can be fired. The opposite proposition is not true; because there exist non-live nets such that are *deadlock free*.

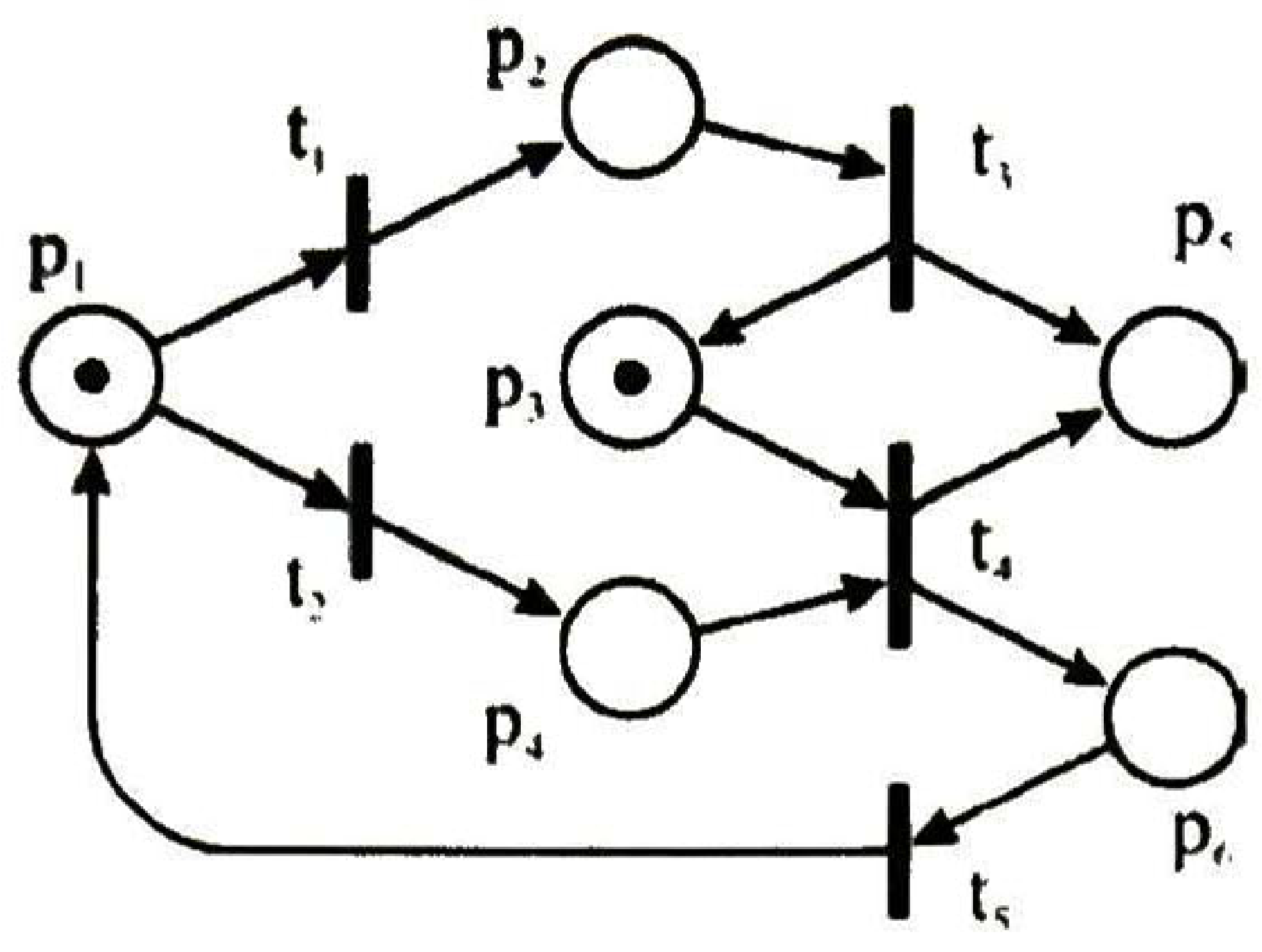

Figure B.1: A nonlive Petri net. But is strictly $L1$-live [Mur89]

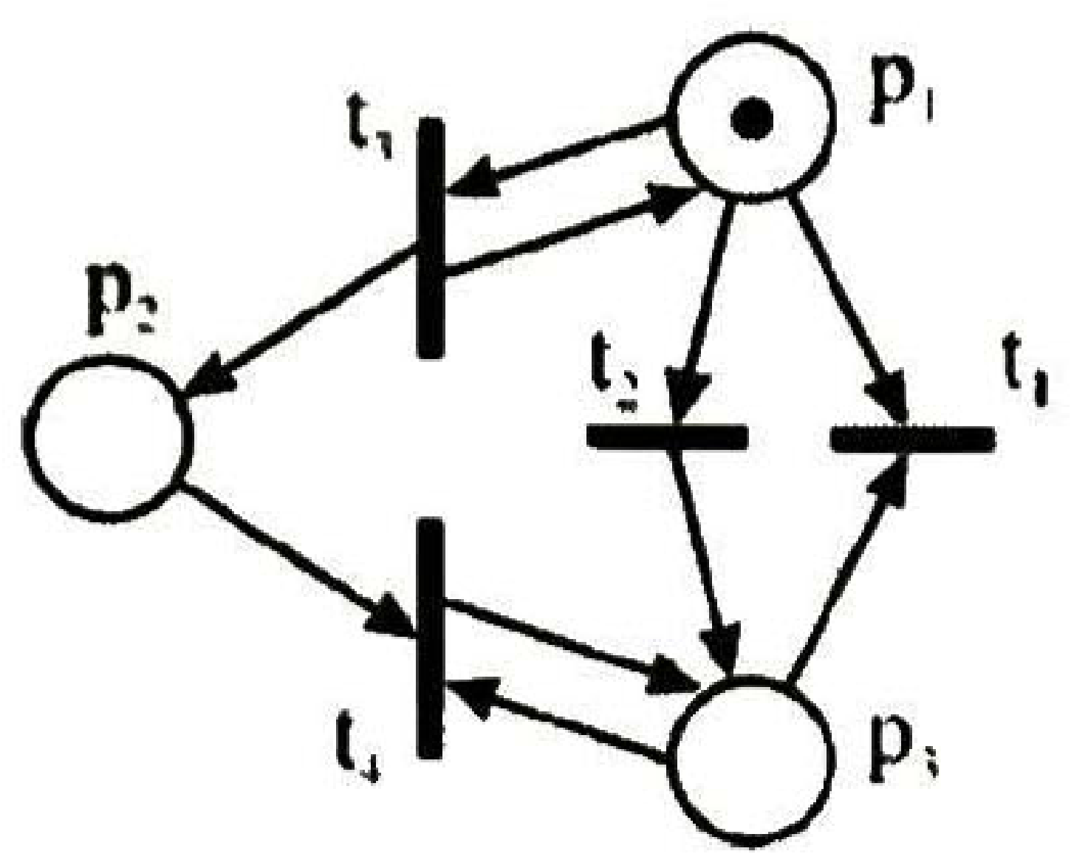

Figure B.2: Transitions t_1, t_2, t_3 and t_4 are dead ($L0$-live), $L1$-live, $L2$-live, $L3$-live, respectively [Mur89].

Definition B.8 *Deadlock-freedom*

A Petri net is deadlock-free if and only if there exist at least an enabled transition for every reachable marking $M \mid M \in R(N, M_0)$.

Observe that liveness is stronger than deadlock-freedom. Then, all live systems are deadlock-free.

B.1.4 Home Markings and Cyclicity

A home marking is set of markings where every marking is a reachable marking. A concurrent interactive system performs in some initial behaviour, when it settles in a ultimate cyclic mode of operation, it is said that the system reaches a home marking.

Definition B.9 *Home Marking*

Let (N, M_0) *be a system. A marking* M *of a net* N *is a home marking of* (N, M_0) *if it is reachable from every marking of* $M \mid M \in R(N, M_0)$.

It is said that (N, M_0) *has a home marking if some reachable marking is a home marking.*

The identification of home markings is an interesting issue in system analysis. A good system performance depends directly of the home marking selection.

Definition B.10 *Cyclicity*

A Petri net (N, M_0) *is cyclic for a initial marking* M_0 *if and only if* $\forall M \mid M \in R(N, M_0) : M_0 \in R(N, M)$.

Cyclicity is related with the fact that a model allows firing sequences leading to the initial marking from any initial marking.

B.2 Structural Properties

Structural properties of systems depend on the initial marking for safeness situations. Two of the most important structural properties are structural boundedness, and structural liveness.

B.2.1 Structural Liveness

Definition B.11 *A transition t is structurally live if and only if there exist an initial marking M_0 making it live.*

Definition B.12 *A Petri net $N = (P, T, Pre, Post)$ is structurally live if and only if each transition of T is structurally live.*

Definition B.13 *A Petri net $N = (P, T, Pre, Post)$ is well-formed if and only there exist a marking M_0 of N such that (N, M_0) is a live and bounded system.*

Observe that every well-formed Petri net is structurally live. Well-formed nets are strongly connected and have at least one transition and one place

B.2.2 Structural Boundedness

Definition B.14 *A place p is structurally bounded if and only if for any finite initial marking M_0 there exist a entire number k such that p will never contain more than k tokens for any reachable marking $M \mid M \in R(N, M_0)$.*

Definition B.15 *A Petri net $N = (P, T, Pre, Post)$ is structurally bounded if and only if each place is structurally bounded.*

Observe that structural boundedness is stronger than boundedness (behavioural property) due to structural boundedness holds for any initial marking. Then, every structural bounded system is bounded (behavioural property). The opposite proposition is not true; because there exist bounded nets such that are not structurally bounded.

Example B.3 *The Petri net shown in Figure B.3 is structurally bounded since each place will never contain more token than the sum of all tokens at the initial marking.*

Example B.4 *Figure B.4 shows that structural boundedness is not a necessary condition for boundedness. The Petri net of the Figure with an initial marking $M_0 = (0, 3, 0)$ is bounded, but changing the initial marking to $M_0' = (0, 4, 0)$ it becomes not bounded. The firing sequence $t_3t_1t_5t_1t_5t_5t_5t_5$ leads to a new marking $M = (1, 4, 0)$. If this sequence is repeated infinitely often, then the place p_1 becomes unbounded*

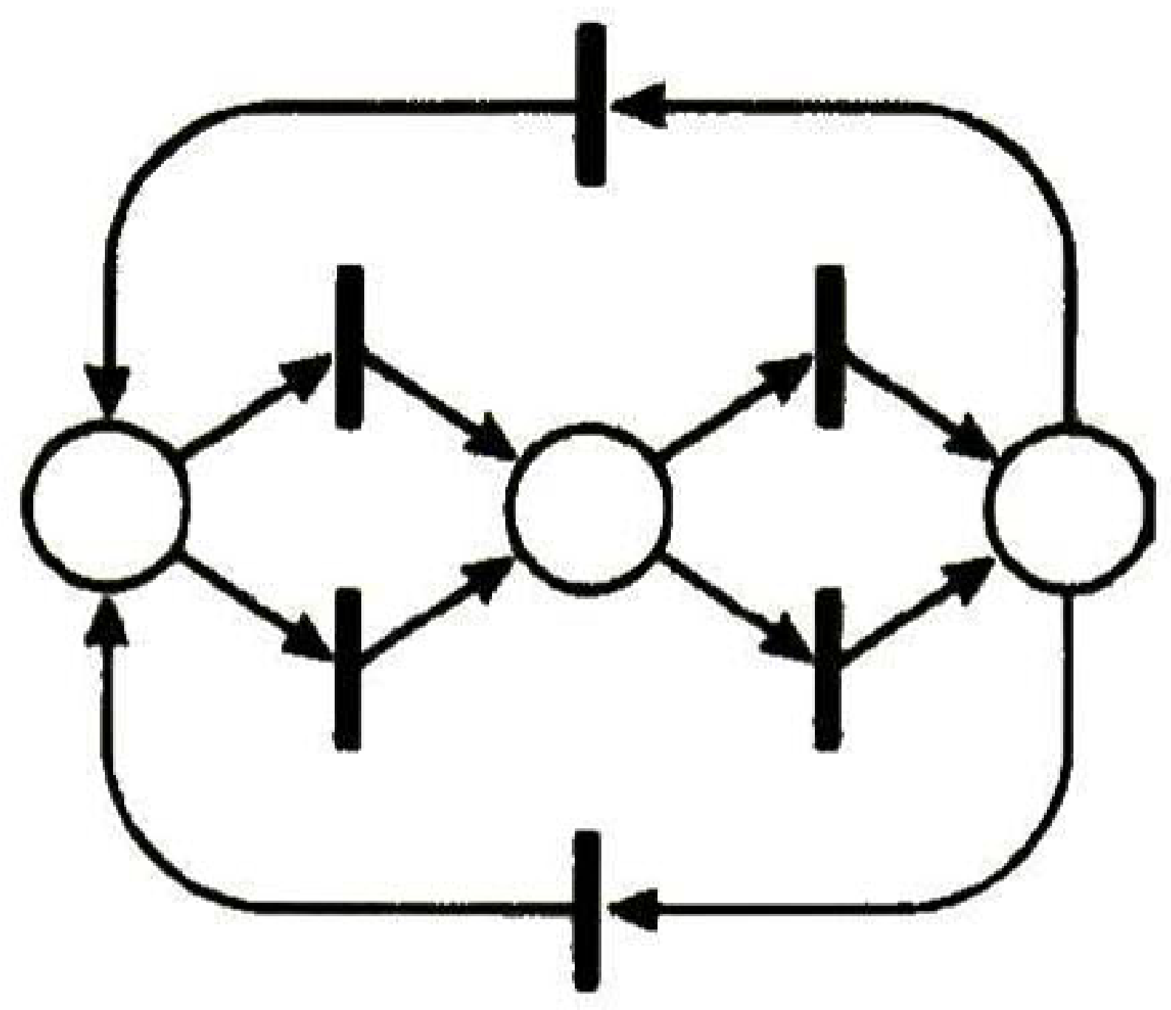
Figure B.3: A structurally bounded and structurally cyclic Petri net.

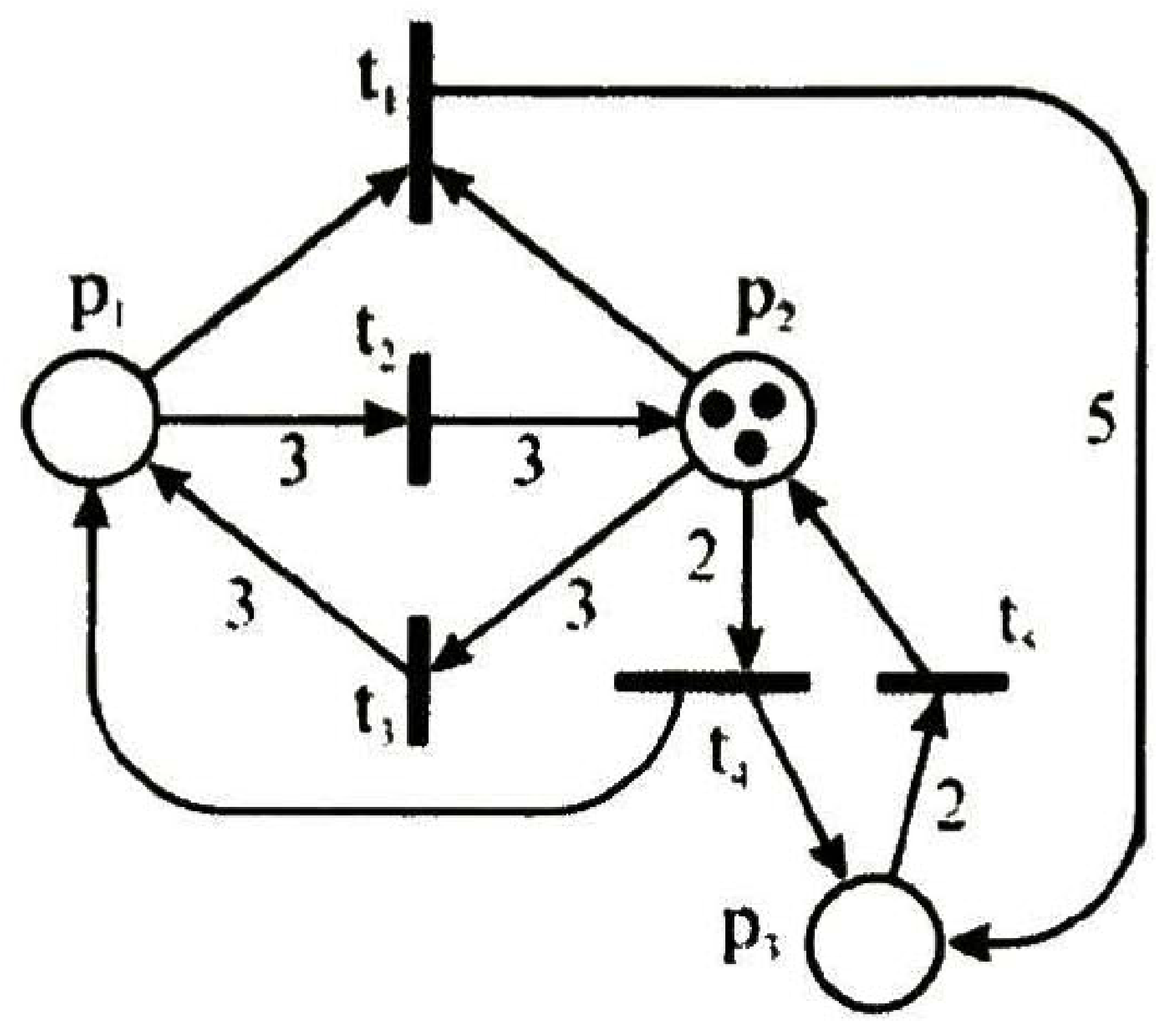

Figure B.4: A bounded and live marked Petri net.

B.2.3 Conservativeness

Tokens in a Petri net may represent resources. The number of tokens in a real system is typically fixed, then the number of tokens in a Petri net model of this system should remain unchanged when the marking changes. When Petri nets are used to represent resource allocation systems, conservation is an important property.

Definition B.16 *Strict Conservativeness*

A Petri net $N = (P, T, Pre, Post)$ with an initial marking M_0 is said to be strictly conservative if $\forall M \mid M \in R(N, M_0) : \sum_{p_i \in P} M(p_i) = constant$

Strict conservation is a strong relationship. It indicates that there exists exactly the same number of tokens in every reachable marking of a Petri net. However, in real systems resources are frequently combined together so that certain task can be executed. Then they are separated after the task is completed. In order to overcome this problem, weights may be associated with places allowing for the weighted addition of tokens in a net to be constant. This results in a broader definition of conservation:

Definition B.17 *Conservativeness*

A Petri net $N = (P, T, Pre, Post)$ with an initial marking M_0 is said to be conservative if there exists a vector $w = (w_1, w_2, ..., w_n)$ where n is the number of places, and $w_i > 0$ for each place $p_i \in P$, such that $\forall M \mid M \in R(N, M_0) : \sum_{i=1}^{n} w_i M(p_i) = constant$

Example B.5 *Figure B.5 shows the Petri net of a simple manufacturing system: a machine processes two types of pieces, namely, type A and type B. In this Petri net model, p_3 represents that the machine is available, p_1 and p_5 represent that pieces of type A and B are available, respectively. p_2 and p_4 represent that pieces of type A and B are under processing, respectively. This Petri net is not strict conservative, because there are three tokens in the initial marking, but in the marking followed by firing either t_1 or t_3 (the machine starts processing piece of type A or piece of type B), there are only two tokens (since the resources of machine and piece are combined into one). However, it is conservative with respect to* $w = (1, 2, 1, 2, 1)$

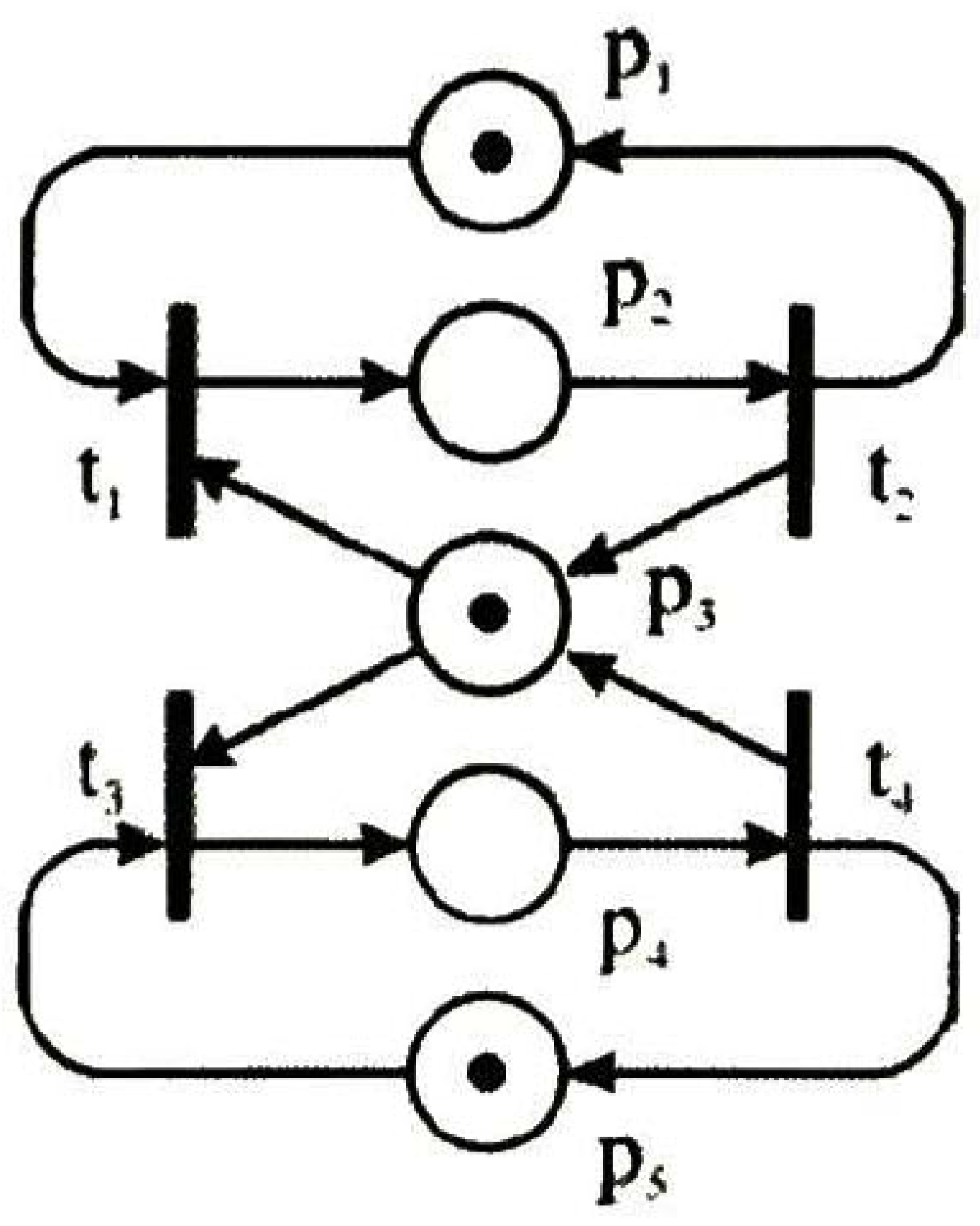

Figure B.5: A manufacturing system's Petri net model which is not conservative.

Appendix C

SYSTEM'S ANALYSIS

C.1 Introduction

The analysis of the modelled systems leads to important behaviour characteristics. Liveness, boundedness and cyclicity analysis through Petri nets can be achieved by different analysis methods. These methods are classified into different groups:

1. Analysis by enumeration
2. Analysis by transformation
3. Structural analysis
4. Analysis by simulation.

Methods belonging to the first four groups are denominated *static methods,* while simulation method is denominated a *dynamic method.* Analysis by enumeration essentially involves the enumeration of all reachable markings, and the use of this method is not suggested for timed Petri nets analysis because of the state explosion.

C.2 Transformation Methods

The interest on transformation methods is made over rules which preserve a set of properties Π. Analysis by transformation is based on the following idea: given a marked Petri net (N, M_0) with a set of properties Π to be analysed, (N, M_0) is then transformed into another Petri net (N', M_0') such that:

1. (N', M_0') satisfies the set of properties Π if and only if (N, M_0) also satisfies it.
2. It is easier to verify the set of properties Π on (N', M_0') than on (N, M_0).

Transformation methods are essentially graphic methods, they build a sequence of Petri nets preserving the studied properties. The objective of the Petri net transformation is to make the analysis easier on (N^{i+1}, M_0^{i+1}) than on the previous Petri nets (N^i, M_0^i) in the sequence. Transformation methods exhibit two group of techniques:

1. Synthesis techniques, and
2. Analysis techniques.

The name *synthesis techniques* is reserved for the set of rules which transforms a source Petri net into a target Petri net with *fewer* nodes. The set of rules which *increase* the number of nodes of the target Petri net is called *analysis techniques.*

A *transformation rule* ϕ is a binary relation on the class of all free-choice Petri nets. Given $(N, N') \in \phi$, the Petri net N is called the *source* Petri net and N' is called the *target* Petri net. $(N, N') \in \phi$ is read "the rule ϕ can transform N into N' "

A rule ϕ is *applicable* to a Petri net N if there exist another Petri net N' such that $(N, N') \in \phi$. A set Φ of rules is called a *kit.* A Petri net N can be transformed into N' by the kit Φ if the successive application of rules $\phi \mid \phi \in \Phi$ can transform N into N'

C.2.1 Synthesis techniques

Synthesis techniques allow eliminating or substituting places and/or transitions in such a manner that the properties to be analysed are not affected by the reduction. The use of these group of techniques is restricted by the existence of irreducible nets for some given reduction rules.

The classical synthesis process searches a total minimization of the description in an automatic form, but this total minimization impedes to minimize future realizations. Hence, optimization already does not search the minimal description; it searches just a simplification on the description capable to allow future minimizations. Synthesis techniques are centred on two fundamental ideas:

1. To reduce, "not to minimize", the number of places and/or transitions of the initial description.
2. To preserve "the physical meaning" of the initial description, attaining only local simplifications.

Synthesis techniques for Petri nets are classified into two well differentiated groups:

1. *Structural Simplifications.* These simplifications are independent of the associated Petri net interpretation, considering only the structure and the initial marking.
   The objective of this structural reduction is the elimination of structural redundancy (false parallel evolutions).
2. *Simplifications holding the Associated Interpretation.* These simplifications are made on and holding the semantic of the specification. These simplifications are not related with the Petri net structure. One of the most used techniques belonging to this group is: transitions fusion.

Transitions Fusion

Defined as $Fusion(t_m, t_n)[t_k]$, transitions fusion technique fusions two transitions t_m, t_n into a single one transition t_k.

Conditions on N:

1. $t_m \bullet = \{p_i\}$
2. $\bullet p_i = \{t_m\}$,
3. $p_i \bullet = \{t_n\}$
4. $\bullet t_n = \{p_i\}$

   Construction of N′:

5. $P' = P \setminus \{p_i\}$
6. $T' = T \setminus \{t_m, t_n\} \cup \{t_k\}$
7. $Pre' = Pre \setminus (p_i, t_n) \cup (\bullet t_m, t_k)$
8. $Post' = Post \setminus (t_m, p_i) \cup (t_k, t_n \bullet)$

This transformation rule is a transitive function, therefore is can be generalized for a path of places and transition as shown in Figure C.1. If this transformation is made over a DTTPN then it is redefined as $Fusion(t_m, t_n)[t_k, d_k]$ where d_k is represents the delay of transition t_k and it is computed as follows:

$$d_k = \sum_{i=1}^{n} d_i$$

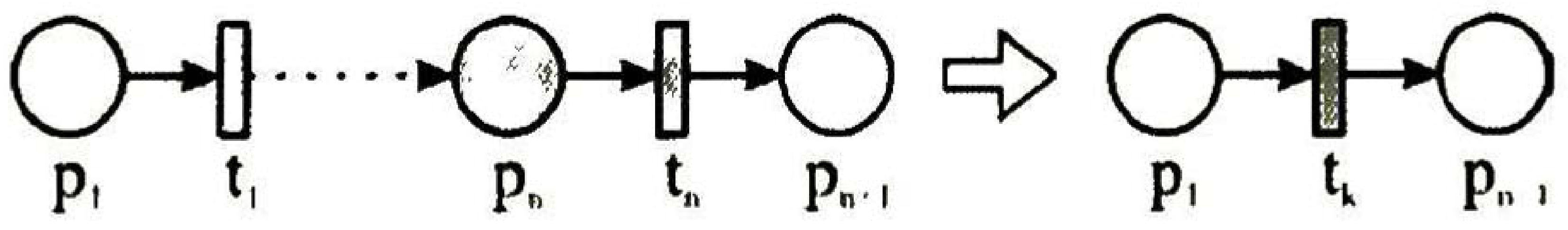

Figure C.1: Example of transitions fusion rule.

Macrotransitions

Macrotransition are subsystem exhibiting marked graphs properties, and can be represented by a unique transition with a firing delay.

Definition C.1 *Let the Petri net $N = (P, T, Pre, Post)$ represent the model of a system. A subnet $N' = (P', T', Pre', Post')$ where $P' \subseteq P$, $T' \subseteq T$, $Pre' \subseteq Pre$, and $Post' \subseteq Post$, is a macrotransition if and only if*

- $\bullet p$ and $p \bullet \in T'$, $\forall p \in P'$,
- $\exists t_i \in T'$ such that $\bullet t_i \subseteq P \setminus P'$ and $\bullet t_i \neq \emptyset$,

- $\exists t_o \in T'$ such that $T_o \bullet \subseteq T \setminus T'$ and $T_o \bullet \neq \emptyset$,
- If $x \in P' \cup T'$ then there exist a path from t_i that leads to x, and other path from x that leads to t_o, and
- $\forall p \in P' : |\bullet p| = |p \bullet| = 1$.

Note that a macrotransitions are non cyclic. Therefore, the analysis of a macrotransition can be achieved by inserting a place p_x such that $p_x \in t_o \bullet$, and $p_x \in \bullet t_i$.

Since macrotransitions exhibit marked graph properties, the firing delay and optimal schedule of a macrotransition can be computed as it is done for marked graphs.

The firing delay of a macrotransition is computed as the cycle time of the marked graph resulting from inserting the place p_x to the macrotransition.

The optimal schedule is to execute transitions $t \in T'$ as soon as they become enabled.

Macroconflict

Macroconflicts are subsystem exhibiting state machines properties, and can be represented by a conflict whose transitions represent the elemental T-invariants required to build an elemental T-cover on the subsystem. Elemental T-invariants and elemental T-cover are introduced in the next section

Definition C.2 *Let the Petri net $N = (P, T, Pre, Post)$ represent the model of a system. A subnet $N' = (P', T', Pre', Post')$ where $P' \subseteq P$, $T' \subseteq T$, $Pre' \subseteq Pre$, and $Post' \subseteq Post$, is a macroconflict if and only if*

- $\bullet t$ and $t \bullet \in P'$, $\forall t \in T'$,
- $\exists p_i \in P'$ such that $\bullet p_i \subseteq T \setminus T'$ and $\bullet p_i \neq \emptyset$,
- $\exists p_o \in P'$ such that $p_o \bullet \subseteq T \setminus T'$ and $p_o \bullet \neq \emptyset$,
- If $x \in P' \cup T'$ then there exist a path from p_i that leads to x, and other path from x that leads to p_o, and
- $\forall t \in T' : |\bullet t| = |t \bullet| = 1$.

Note that a macroconflicts are non cyclic. Therefore, the analysis of a macroconflict can be achieved by inserting a transition t_x such that $t_x \in p_o \bullet$, and $t_x \in \bullet p_i$.

Since macroconflicts exhibit state machines properties, transitions in a macroconflict represent the elemental T-invariants required to build an elemental T-cover on the subsystem. Since inserting the transition t_x into the macroconflict, the macroconflict becomes cyclic and its optimal schedule can be computed through T-invariants as it is done for state machines.

The firing delay of a transition belonging to a macroconflict is equal to the execution time of the elemental T-invariant that such transition represents.

The optimal schedule of a macroconflict is to fire the transition of the conflict as soon as they become enabled.

C.2.2 Analysis techniques

Increase techniques are suitable just for a reduced set of cases since it transforms a net into another with more nodes, and the objective of the transformation (to make the analysis easier) seems not to be accomplished. Therefore, increase techniques are most used to consequently better apply a reduction technique. One of the most used techniques belonging to this group is the meiosis technique.

Meiosis

Defined as $Meiosis(p_i, t_j)$, meiosis technique clones a place p_i and a transitions t_j as follows.
Conditions on N:

1. $\bullet p_i = \{t_g, t_h\}$,
2. $p_i \bullet = \{t_j\}$
3. $\bullet t_j = \{p_i\}$,
4. $t_j \bullet = \{p_k\}$

   Construction of N′:

5. $P' = P \cup \{p'_i\}$
6. $T' = T \cup \{t'_j\}$
7. $Pre' = Pre \cup (p'_i, t'_j)$
8. $Post' = (Post \setminus (t_g, p_i)) \cup (t_g, p'_i) \cup (t'_j, p_k)$

This transformation rule is also a transitive function, therefore is can be generalized for a path of places and transition. Figure C.2 shows an example of this rule. If this transformation is made over a DTTPN then define the delay d'_j of transition t'_j as follows:

$$d'_j = d_j$$

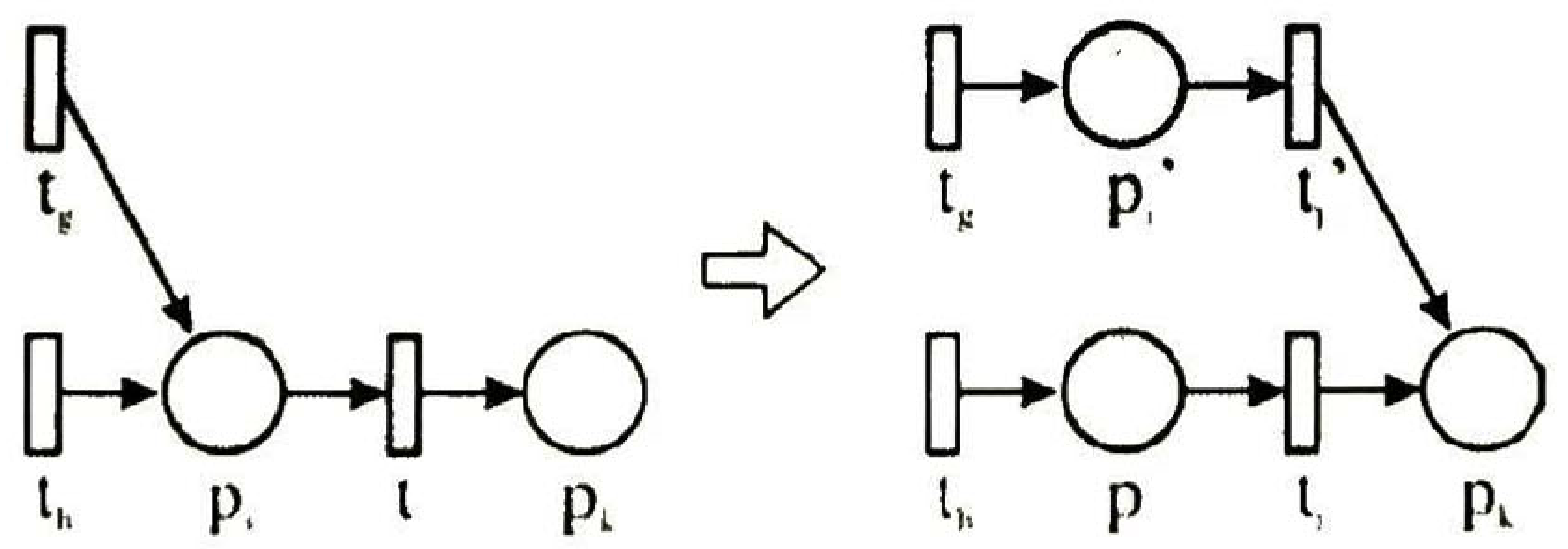

Figure C.2: Example of a meiosis transformation.

C.3 Description Methods

Description methods are also considered as transformation method. Description methods are divided into two clasess Bottom-Up Methods and Top-Down Methods.

C.3.1 Bottom-Up Methods

In bottom-up methods, the strategy consists on elaborating complex models starting from simple models corresponding to the resulting parts of a decomposition of the complete system, in order to later bind them forming a model corresponding to the complete system. Every subsystem is modelled for separate, either ignoring the interactions with the other subsystems. The model is built relatively quick and its analysis is easier.

The different subsystems models can have common places or transitions representing common resources or activities, they constitute the interaction among the subsystems. Once, subsystems models have been defined, the combination of the models is achieved by fusing the common parts (places, transition, or paths).

Transitions Merging

On each step, two subnets are bind in such a manner that two transitions, one of each subnet, they become a single transition. The procedure is described at once:

Definition C.3 *A merging function $M(N_1, t_i, N_2, t_j)$ is a function that synchronises two nets (N_1 and N_2) by merging two transitions ($t_i \in T_1$ and $t_j \in T_2$), one of each subnet, into a single one ($t_k \notin T_1 \cup T_2$), generating a more complex net ($N = N_1 \cup N_2$).*

Let $N_1 = (P_1, T_1, Pre_1, Post_1)$ and $N_2 = (P_2, T_2, Pre_2, Post_2)$ be two submodels to be bind through the merging of $t_i \in T_1$ and $t_j \in T_2$ into a single transition $t_k \notin T_1 \cup T_2$. The resulting net $N = (P, T, Pre, Post)$ is built as follow:
$P = P_1 \cup P_2$
$T = (T_1 \backslash t_i) \cup (T_2 \backslash t_j) \cup \{t_k\}$
$Pre = \{Pre_1 \backslash (\bullet t_i, t_i)\} \cup \{Pre_2 \backslash (\bullet t_j, t_j)\} \cup \{(\bullet t_i \cup \bullet t_j, t_k)\}$
$Post = \{Post_1 \backslash (t_i, t_i \bullet)\} \cup \{Post_2 \backslash (t_j, t_j \bullet)\} \cup \{(t_k, t_i \bullet \cup t_j \bullet)\}$

Example C.1 *Figure C.3 a) shows the model of a warehouse where t_1 represents the input of raw material, p_1 represents the store, and t_2 represents the output of raw material. Figure C.3 b) shows the model of a container producer machine where t_3 represents the input of the raw material to the machine, t_4 represents the injection of the preform and the lid, t_5 represents the blowing of the preform, and t_6 represents the output of the container pieces from the machine These models are combined by the merging of transitions t_2 and t_3 in order to obtain a more complex model. Figure C.4 shows the obtained model.*

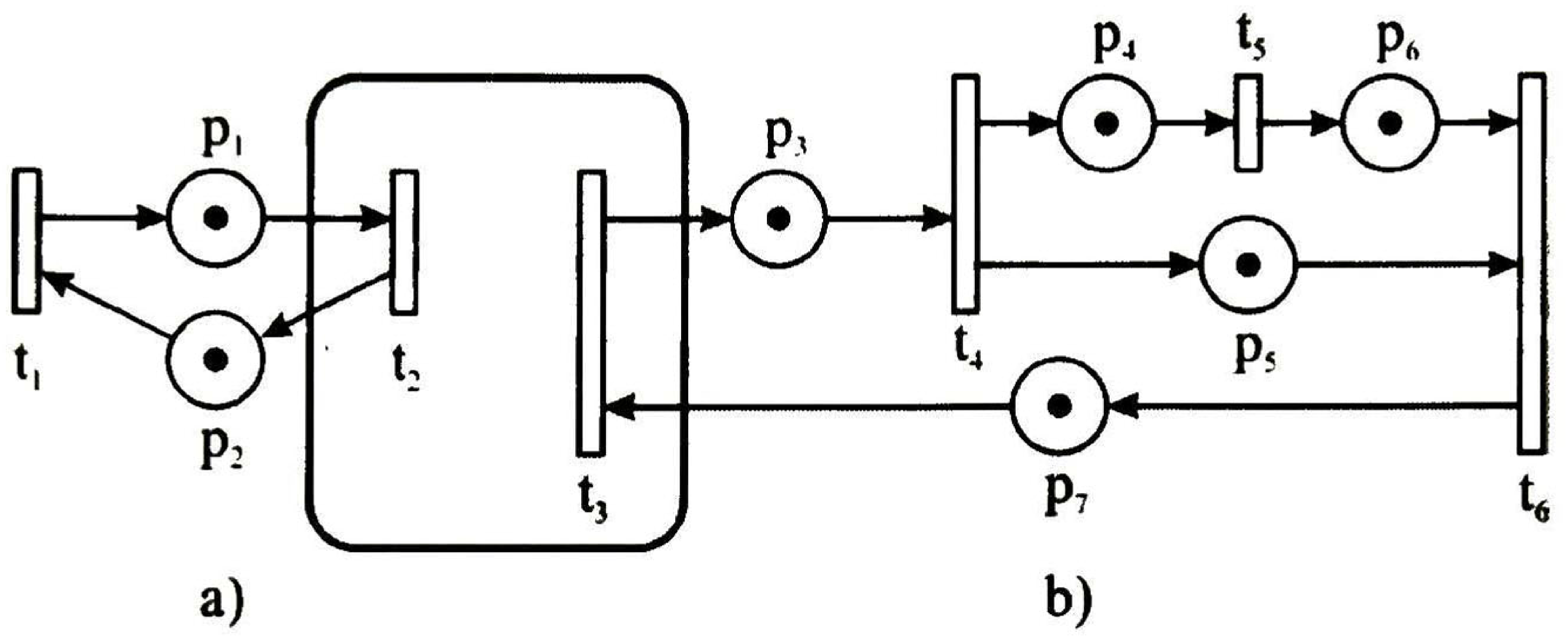

Figure C.3: Two simple models.

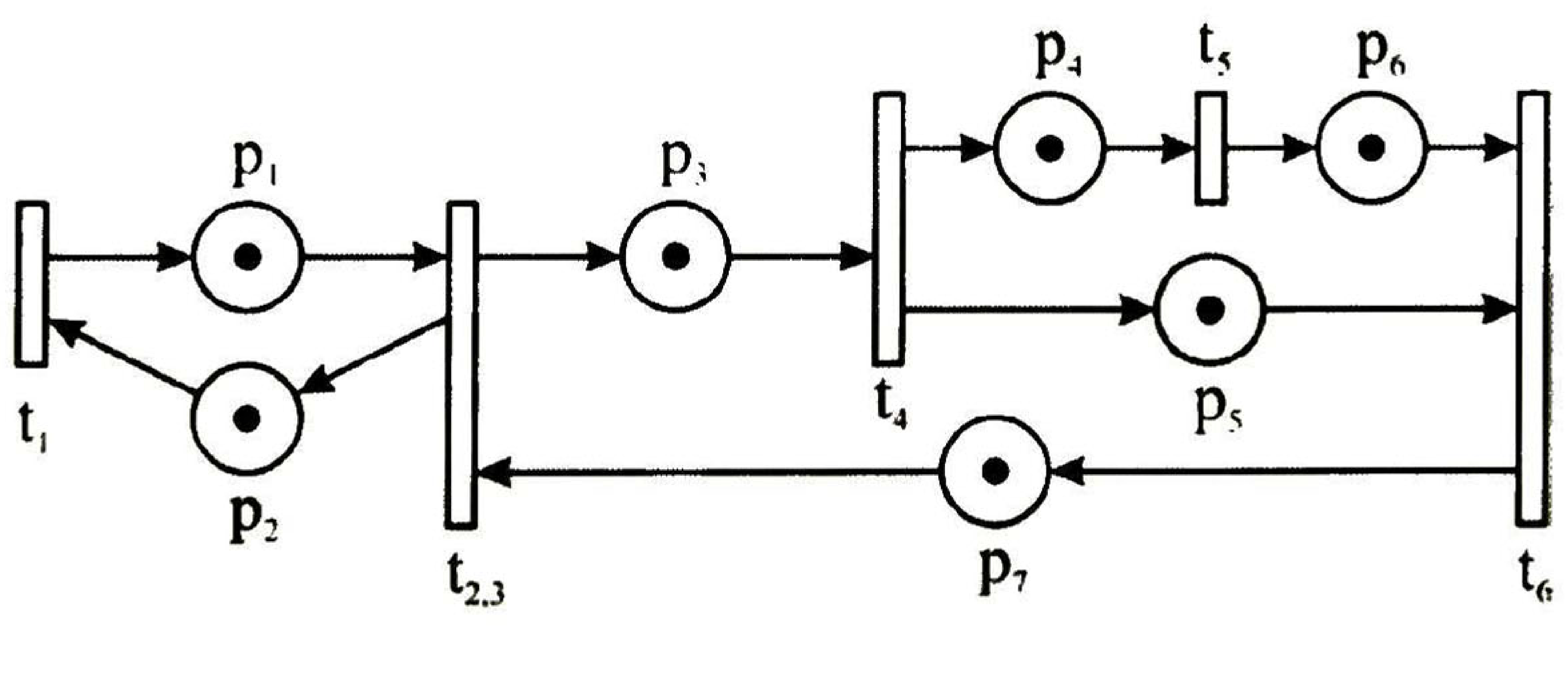

Figure C.4: A complex model built from two simple models.

C.3.2 Top-Down Methods

In the top-down methods, there exists a global vision of the system from the beginning of the model construction. The strategy consists on defining initially, a model of high level of abstraction without caring the details. Afterwards, it is achieved a step-wise refinement, in other words, places and transitions expansion in order to give more detail to the model. The refinement is made until the level of detail of the model satisfies the system specifications.

These methods impose conditions in the expansion of places or transitions with the purpose that at each step the properties of the original net are preserved.

C.4 Structural Analysis

C.4.1 Invariants Method

This group of analysis methods differ from the two previous (enumeration and transformation) in the sense that it is not an exhaustive method. The analysis is concentrated on the net structure, resumed on the incidence matrix, using linear algebra tools.

Conservativeness and cyclicity are two structural properties easy to analyse through invariants method, they are related with some dynamical-structural properties as structural boundedness and structural liveness respectively. The analysis process is reduced to solve a set of linear equations, obtained from the incidence matrix of a given Petri net model. Solutions of these linear equations (restricted to non negative integers) lead to a classification of the Petri net: conservative and repetitive components. This classification allows studying properties independently of the initial marking.

An invariant of a dynamic system is an assertion that holds for every reachable state. For DEDS's modelled by Petri nets, it is possible to compute certain vectors of rational numbers (directly from the structure) which induce invariants. Below are shown two techniques of invariants analysis based on the determination of valid relationships independent of the net evolution.

Place Invariants

Given an arbitrary Petri net, it is difficult to characterize all the vectors Y such that $Y \cdot M$ remains constant for every reachable marking M. However, it is easy to derive a sufficient condition from the marking equation.

Consider the Petri net of Figure C.5. It is easy to see that for every reachable marking M the equation $M(p_2) + M(p_3) = 1$ holds, i.e., it is an invariant of the system. This equation can be rewritten as

$$\begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} M(p_1) \\ M(p_2) \\ M(p_3) \end{bmatrix} = 1 \text{ or just } \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} M = 1$$

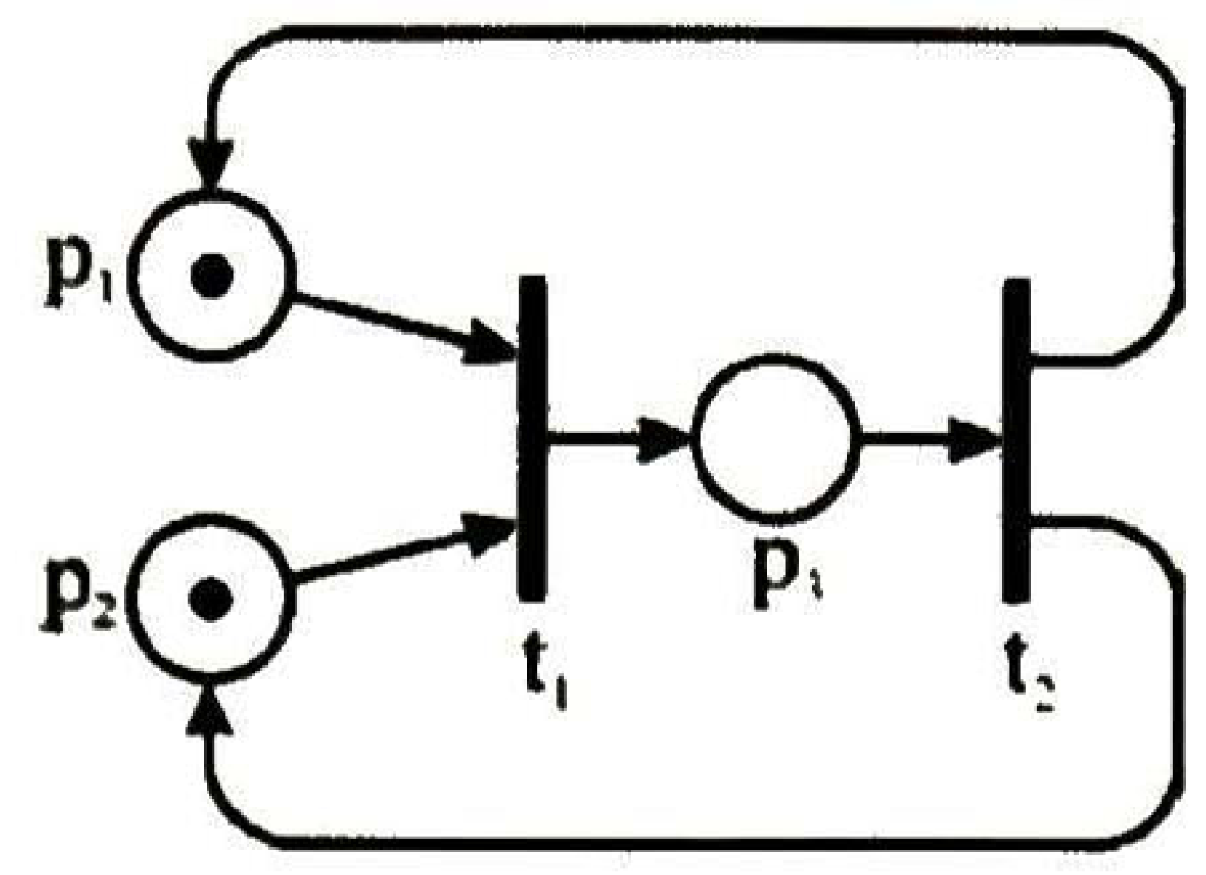

Figure C.5: The vector $Y = (0, 1, 1)$ is an P-semiflow.

Definition C.4 *A P-semiflow Y of a net N is a nonnegative rational-valued solution Y of the equation*

$$Y \cdot \mathbf{N} = 0 \tag{C.1}$$

Definition C.5 *A P-semiflow Y is called semi-positive if and only if $Y \geqslant \mathbf{0}$ and $Y \neq \mathbf{0}$. A P-semiflow Y is called positive if and only if $Y > \mathbf{0}$, i.e. $Y(p) > 0$ for every place p. The support of a P-semiflow Y, denoted by $\langle Y \rangle$, is the set of places satisfying $Y(p) > 0$. Note that $\langle Y \rangle$ cannot be the empty set because $Y \neq \mathbf{0}$.*

The non-zero entries in a P-semiflow represent weights associated to the corresponding places so that the weighted addition of tokens in these places is constant for every reachable marking.

By definition, a set of P-semiflows of a net constitutes a vector space over the field of rational numbers. The set $\{(1, 0, 1), (0, 1, 1)\}$ is a basis of the space vector of P-semiflows of the Petri net in Figure 2.7.

P-semiflows allow analysing liveness and boundedness in Petri nets. Esparza [D&E95] exposes these two properties and their relationship with P-semiflows. Let (N, M_0) be a system

- If (N, M_0) is a live system, then every semi-positive P-semiflow Y of N satisfies $Y \cdot M_0 > 0$.
- If N has a positive P-semiflow Y, then (N, M_0) is bounded.

Definition C.6 *Let $N = (P, T, Pre, Post)$ be a Petri net, a P-invariant of N is a sub Petri net $\tilde{N} = (\tilde{P}, \tilde{T}, \tilde{Pre}, \tilde{Post})$ generated by a P-semiflow Y where:*

1. $\tilde{P} = \langle Y \rangle$
2. $\tilde{T} = T \cap \{\bullet \langle Y \rangle\} \cap \{\langle Y \rangle \bullet\}$
3. $\tilde{Pre} = Pre \cap (\tilde{P} \times \tilde{T})$
4. $\tilde{Post} = Post \cap (\tilde{T} \times \tilde{P})$

Transition Invariants

T-semiflows of a Petri net N are vectors X satisfying $\mathbf{N} \ X = 0$. It seems natural to study if T-semiflows also have interesting properties. It will be shown that T-semiflows are related to the occurrence of sequences which reproduce a marking (cyclicity property), i.e., those that lead from a marking to itself.

Definition C.7 *A T-semiflow X of a Petri net N is a nonnegative rational-valued solution of the equation*

$$\mathbf{N} \cdot X = 0 \tag{C.2}$$

The set of T-semiflows of a Petri net constitutes again a vector space over the field of rational numbers.

Example C.2 *The dimension of the T-semiflow space vector of the Petri net of Figure C.6, is 2: the set $\{(1,0,1), (0,1,1)\}$ is a basis of space of T-semiflows.*

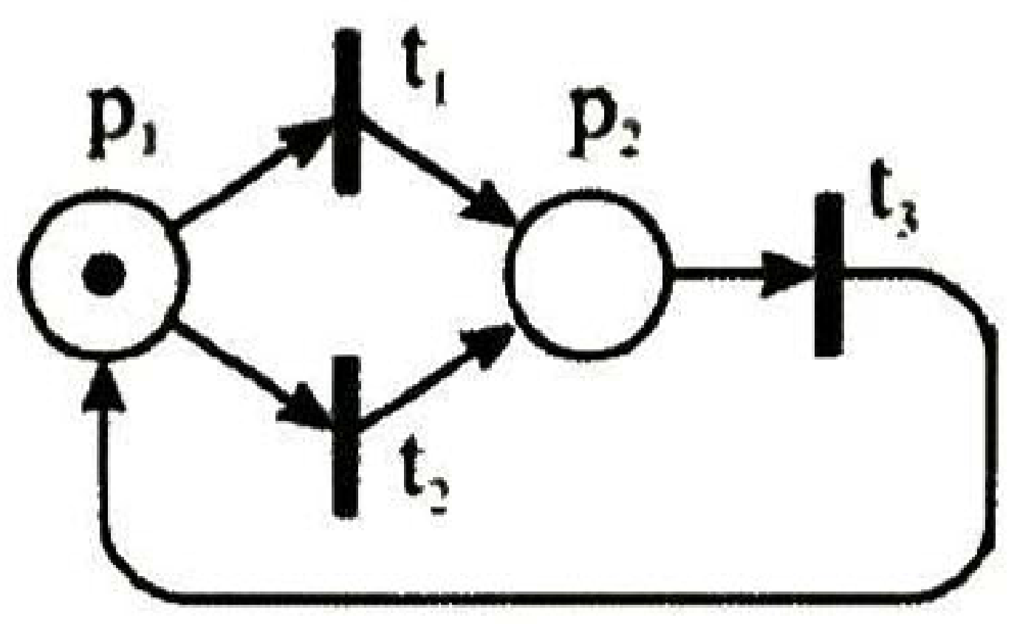

Figure C.6: The vector $(1,0,1)$ is a T-invariant

Proposition C.1 *Fundamental Property of T-semiflows*

Let σ be a finite sequence of transitions of a Petri net N which is enabled in a marking M. Then the Parikh vector $\overrightarrow{\sigma}$ is a T-semiflow if and only if $M \xrightarrow{\sigma} M$ (i.e., if and only if the occurrence of σ reproduced the marking M).

Proof. ($\Rightarrow$) Since σ is enabled at M, then $M \xrightarrow{\sigma} M_1$ for some marking M_1. By the Marking Equation $M_1 = M + \mathbf{N} \cdot \overrightarrow{\sigma}$. Since $\overrightarrow{\sigma}$ is a T-semiflow then $\mathbf{N} \cdot \overrightarrow{\sigma} = \mathbf{0}$. Therefore, $M_1 = M$.
($\Leftarrow$) If $M \xrightarrow{\sigma} M$ then, by the Marking Equation, $\mathbf{N} \cdot \overrightarrow{\sigma} = \mathbf{0}$. Therefore, $\overrightarrow{\sigma}$ is a T-semiflow. ■

The definition of semi-positive T-semiflow, positive T-semiflow and support are defined as for P-invariants.

Definition C.8 *Let $N = (P, T, Pre, Post)$ be a Petri net, X be a T-semiflow of N, M be a marking such that $M \in R(N, M_0)$, σ be a firable sequence such that $\sigma = t_1, t_2, ..., t_n$ and $\overrightarrow{\sigma} = X$. The firing of the sequence σ is an execution of X if and only if $M \xrightarrow{\sigma} M$, and σ is a sequence with at least one transition.*

For a given T-semiflow X there might be more that one firable sequence σ that executes X. The firable sequence σ representing the execution of X depends on the marking.

Example C.3 *For the Petri net in Figure C.6 there exist a T-semiflow $X = (1,0,1)$. For this T-semiflow there exist two firable sequences $\sigma_1 = t_1, t_3$ and $\sigma_2 = t_3, t_1$. If the place p_1 is marked, then σ_1 shall be the sequence σ representing the execution of X. If the place p_2 is marked, then σ_2 shall be the sequence σ representing the execution of X. Note that $\overrightarrow{\sigma_1} = \overrightarrow{\sigma_2} = X$.*

Definition C.9 *Let X be a T-semiflow of a Petri net N, and let σ be a firable sequence representing one of those possible executions of X. The T-semiflow X is said to be enabled if and only if σ is enabled.*

Definition C.10 *Let $N = (P, T, Pre, Post)$ be a Petri net, a T-invariant of N is a sub Petri net $\tilde{N} = (\tilde{P}, \tilde{T}, \tilde{Pre}, \tilde{Post})$ generated by a T-semiflow X where:*

1. $\tilde{P} = P \cap \{\bullet\langle X\rangle\} \cap \{\langle X\rangle\bullet\}$

2. $\tilde{T} = \langle X\rangle$

3. $\tilde{Pre} = Pre \cap (\tilde{P} \times \tilde{T})$

4. $\tilde{Post} = Post \cap (\tilde{T} \times \tilde{P})$

The definition of invariants is based on the incidence matrix $\mathbf{N}$ of a Petri net. For invariant analysis, usually, the Petri net requires to be 'pure'. The analysis by invariants considers the Petri net as a whole, and allows studying global properties of the net. Esparza [D&E95] exposes some Petri nets properties and their relationship with T-semiflows: Let (N, M_0) be a system

- Every well-formed Petri net has a positive T-semiflow.

- Every connected Petri net with a positive P-semiflow and a positive T-semiflow is strongly connected.

Minimal Semiflows and Covers

Definition C.11 *A semi-positive semiflow Y_i is minimal if no other semi-positive semiflow Y_j satisfies $\langle Y_j\rangle \subset \langle Y_i\rangle$.*

Observe that by this definition each nonzero multiple of a minimal semiflow is again minimal, because the minimalism of an invariant depends solely on its support.

Theorem C.1 *Every semi-positive semiflow is a linear combination of minimal semiflows.*

Proof. Let $Y : X \to Q$ be a semi-positive semiflows. The proof is done by induction on $|\langle Y\rangle|$.
Base. $|\langle Y\rangle| = 1$. Then Y is minimal because every semi-positive semiflows has at least one non-zero entry.
Step. $|\langle Y\rangle| > 1$. If Y is minimal, then it is done. So assume that Y is not minimal. Then, there exist a minimal semiflows Y_i such that $\langle Y_i\rangle \subset \langle Y\rangle$. Let x be an element of $\langle Y_i\rangle$, and $Y_j = Y - Y_i$ be a minimal semiflow such that $x \notin \langle Y_j\rangle$.
By the choice of x, $Y_j(x) = 0$ and $Y_j \geq \mathbf{0}$. Moreover, $\langle Y_j\rangle \subset \langle Y\rangle$. Since $\langle Y_i\rangle$ is a proper subset of $\langle Y\rangle$, $\langle Y_j\rangle \neq \emptyset$. Therefore Y is a linear combination of T-semiflows, because $Y = Y_i + Y_j$.
In the case that Y_j is not a minimal T-semiflow. The induction hypothesis can be applied to Y_j, conclude that Y_j is linear combination of minimal semiflows, and consequently Y is also linear combination of minimal semiflows. ■

Theorem C.2 *If a Petri net has a positive semiflow, then every semiflow is a linear combination of minimal semiflows.*

Proof. Let N be a Petri net having a positive semiflow Y, and let Y_i be an arbitrary semiflow of N. There exist an integer k such that $kY + Y_i$ is a semi-positive semiflow. By the previous theorem, both Y and $kY + Y_i$ are sums of minimal semiflows. Since $Y_i = (kY + Y_i) - kY$, the semiflow Y_i is a linear combination of minimal semiflows. ■
Note that Y_i is neither positive nor semi-positive semiflow, since some of its entries are negative.

Definition C.12 *A semi-positive semiflow Y_i is canonical if the greatest common divisor of its non-zero entries is the unit.*

According with the previous definition:

- $Y = (12, 20, 0, 0, 4)$ is not a canonical semiflow due to $\gcd(Y_1, Y_2, Y_5) = 4$.
- $Y = (2, 0, 3, 0, 4)$ is a canonical semiflow due to $\gcd(Y_1, Y_3, Y_5) = 1$.

Definition C.13 *A semi-positive semiflow Y_i is called elemental if it is minimal and canonical.*

The number of elemental semiflows of a net does not depend of the rank of the incidence matrix [Sil85], since this number of T-semiflow can be less, equal or greater than the dimension of any of the infinite basis of the kernel. Moreover, a Petri net (even a simple one) might have several elemental semiflows.

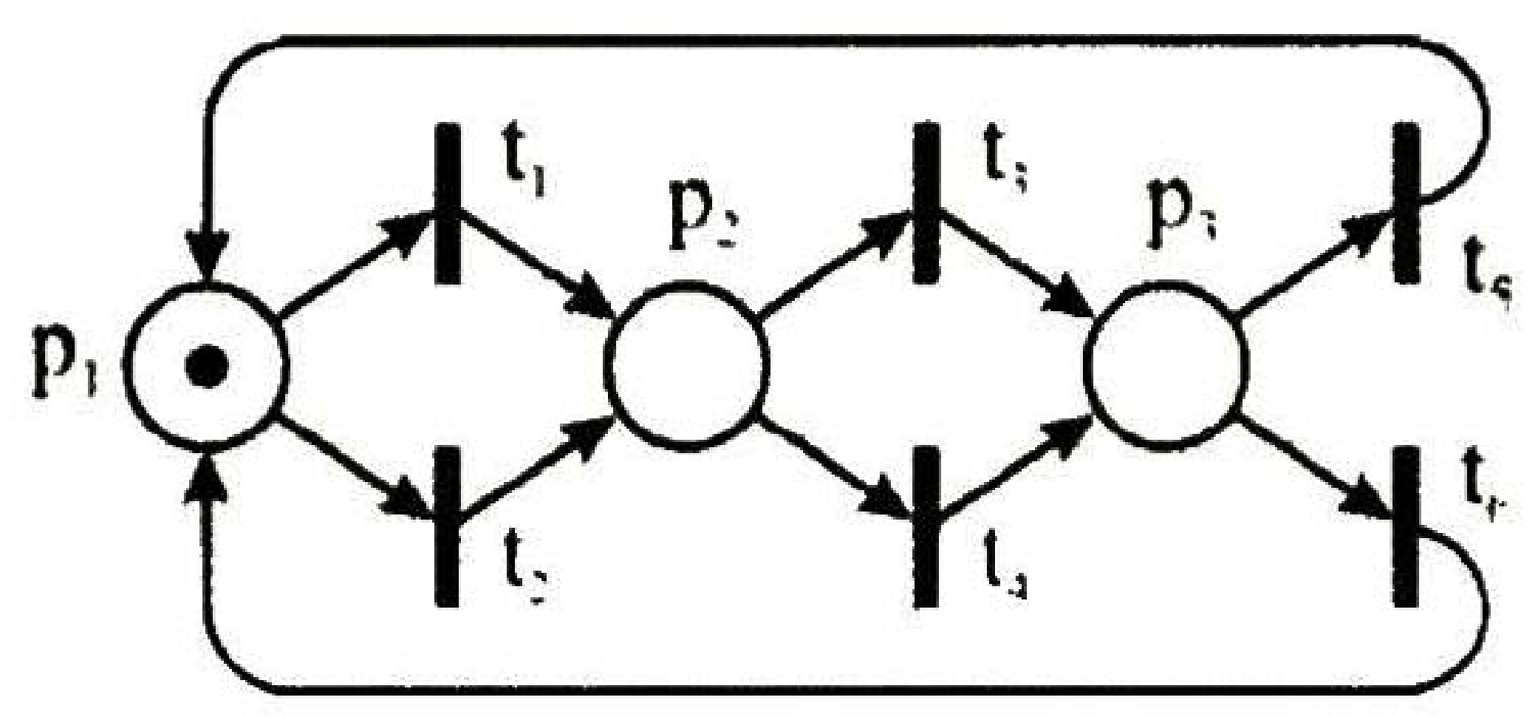

Figure C.7: A Petri net with different elemental T-covers.

Definition C.14 *Let $C = \{Y_1, Y_2, ..., Y_q\}$ be a set of semi-positive elemental semiflows and define X as*

$$X = \sum \alpha_i Y_i, \quad \textit{where } Y_i \in C \textit{ and } \alpha_i \in \mathbb{Z}^+$$

C is said to be a cover if X is a positive vector.

Proposition C.2 *The following algorithm computes in a polynomial number of steps if a set C of semi-positive semiflows is a cover or not.*

Algorithm C.1 *Algorithm to verify if a set C of semi-positive elemental semiflows is a cover or not.*

1. Build a matrix **C**, where each column of **C** represents a different semiflow of those contained in C.
2. Multiply the matrix **C** by a vector of dimension $|C|$, whose entries are all 1's.
3. If the resulting vector is a semi-positive vector, then C is not a cover.
4. If the resulting vector is a positive vector, then C is a cover.

Proof. Step 1 can be achieved in n steps, where n is the cardinality of C.
Step 2 is a matrix multiplication; there exist polynomial algorithms for this purpose that can be achieved with at most $m \cdot n^2$ steps, where m is the number of entries of the semiflows.
Steps 3 and 4 also can be achieved in at most n steps.
Therefore, the algorithm is executed in at most $(m \cdot n^2) + 3n$ steps ■

Definition C.15 *A cover $\mathcal{C}$ is said to be an elemental cover if $X = \sum \alpha_i Y_i$ becomes a positive semiflow only when $\alpha_i > 0$ for every $Y_i \in \mathcal{C}$.*

Proposition C.3 *The following algorithm computes in a polynomial number of steps if the set of semiflows $\mathcal{C}$ is an elemental cover or not.*

Algorithm C.2 *Algorithm to verify if a set $\mathcal{C}$ of elemental semiflows is an elemental cover or not.*

1. Prove if $\mathcal{C}$ is a cover.

2. If $\mathcal{C}$ is not a cover, then the algorithm finishes

3. If $\mathcal{C}$ is a cover, then compute a vector X by the following linear programming problem:

$$\min \sum_{i=1}^{n} X(i)$$

$$\mathbf{C} \cdot X \geq \mathbf{1}$$

$$X \geq 0$$

   Where $\mathbf{C}$ is the matrix representing the set of T-semiflows $\mathcal{C}$.

4. If the resulting vector X is a semi-positive vector, then $\mathcal{C}$ is a non elemental cover.

5. If the resulting vector X is a positive vector, then $\mathcal{C}$ is an elemental cover.

Proof. Step 1 can be achieved in polynomial time by proposition C.2.
Step 2 is just a decision on the on the outcome of step 1.
Step 3 is linear programming problem, and every linear programming problem can be solved in polynomial time.
Steps 4 and 5 are again decisions on the on the outcome of step 3. Therefore, the algorithm is executed in a polynomial number of steps. ■

Note that a Petri net might have more than one elemental cover; it depends of the structure of such Petri net.

Example C.4 *From the Petri net of Figure C.7 the following elemental T-covers can be found*

$$\mathcal{C}_1 = \left\{ \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \right\} \quad \mathcal{C}_2 = \left\{ \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right\}$$

$$\mathcal{C}_3 = \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\} \quad \mathcal{C}_4 = \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right\}$$

ANIVERSARIO Cinvestav

Centro de Investigación y de Estudios Avanzados del IPN

Unidad Guadalajara

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó la tesis: SCHEDULING DE SISTEMAS DE EVENTOS DISCRETOS MODELADOS POR MÁQUINAS DE ESTADOS SINCRONIZADAS del(a) C. José Luis CÓRDOVA BARBA el día 24 de Enero de 2003

[illegible]
DR. OFELIA BEGOVICH MENDOZA
INVESTIGADOR CINVESTAV 3A
CINVESTAV GDL
GUADALAJARA

[illegible]
DR. LUIS ERNESTO LÓPEZ MELLADO
INVESTIGADOR CINVESTAV 3A
CINVESTAV GDL
GUADALAJARA

[illegible]
DR. ANTONIO RAMIREZ TREVIÑO
INVESTIGADOR CINVESTAV 2A
CINVESTAV GDL
GUADALAJARA

[illegible]
DR. LUIS ISIDRO AGUIRRE SALAS
PROFESOR
DEPARTAMENTO DE INGENIERIAS
CENTRO UNIVERSITARIO DE LA COSTA SUR DE LA UNIVERSIDAD DE GUADALAJARA
GUADALAJARA