Towards Verifying Petri Nets: A Model Checking Approach

Proyecto Fin de Máster en Programación y Tecnología Software

Máster en Investigación en Informática, Facultad de Informática, Universidad Complutense de Madrid

María Rosa Martos Salgado
Curso académico 2009-2010

Trabajo dirigido por el Doctor David de Frutos Escrig.
Colaborador externo: Doctor Fernando Rosa Velardo.

La abajo firmante, matriculada en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Towards verifying Petri nets: a model checking approach", realizado durante el curso académico 2009-2010 bajo la dirección de David de Frutos Escrig y con la colaboración externa de dirección de Fernando Rosa Velardo en el Departamento de Sistemas Informáticos y Computación, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Agradecimientos

Este trabajo representa para mí un primer acercamiento a un mundo que espero seguir descubriendo poco a poco: el mundo de la investigación. Quiero expresar mi más sincero agradecimiento a todos los que me han apoyado y acompañado en esta primera incursión que ha hecho que quiera continuar este camino.

En primer lugar, quiero agradecer a Fernando Rosa haber guiado mis primeros pequeños pasitos en este mundo. Gracias por su paciencia y por haberme transmitido la inquietud y las ganas de saber cada día un poquito más.

Gracias también a David de Frutos, por sus correcciones que han hecho que este trabajo mejore y sobre todo por haberme mostrado parte del camino a recorrer.

No puedo olvidarme de mis compañeros de matemáticas y en especial agradecer a David las clases, conversaciones, ejercicios, presentaciones... que hemos tenido en común este curso.

Quiero darle las gracias a No es Culpa Nuestra, y en especial a los habitantes de cierto barrio: Gracias Steward, Irene, José, Julio, Rodrigo, Beatriz, Karol, Alvarito... Gracias también a Antón, por todo. Él, que le gustan tanto las palabras, hace que una palabra tan malgastada como "amistad" recobre todo su significado.

Muchas gracias a la música, que hace mejores mis peores momentos. Gracias a los jonsuitas: Alba, Flexer, Elena, Susana, Jesús... por aguantar los meses que llevo sin poder dejar de hablar de este proyecto cada vez hablo con ellos y porque siempre, siempre, están ahí cuando los necesito. Gracias a Alberto, por ser mi mejor antídoto contra la soledad del agosto madrileño, y por enseñarme el verdadero significado de cierta función matemática.

Gracias a mi familia, por haberme soportado en estos días en los que más tiempo he pasado en casa, pero a la vez más lejos de ellos he estado. Gracias a mi madre, por apoyarme siempre.

Contents

List of Figures

Abstract

Petri nets are an important formalism to specify concurrent and distributed systems. When Petri nets are used to model systems, we would like to be able to verify certain properties about the modelled systems automatically, as is done when other formalisms, as finite automata, are applied. This problem is called the "model checking" problem. In this work, we present a compilation of the main model checking results for plain Petri nets and two orthogonal monotonic extensions: reset Petri nets and ν-Petri nets. We first introduce Petri nets and its extensions, and summarize the main decidability results for them, which are useful in proving decidability issues about model checking. An important part of the work has been looking for model checking results about other formalisms (in particular lossy VASS with inhibitor arcs) which are applicable to reset nets and ν-Petri nets. Despite the fact that this work is eminently bibliographic, some new results are proved here. In particular, a proof for the undecidability of the repeated coverability problem for ν-Petri nets is given, and we use this result to prove the undecidability of model checking certain temporal logic for ν-Petri nets. Finally, as all the considered temporal logics for reset nets and ν-Petri nets are undecidable, we discuss how to approach the model checking problem in the undecidable case with two different techniques: unfolding and cover computation.

Keywords: Petri nets, reset nets, ν-PN, formal verification, model checking, temporal logics, decidability.

Resumen

Las redes de Petri son un importante formalismo para la especificación de sistemas concurrentes y distribuídos. Tal y como se hace con otros formalismos, como los autómatas finitos, nos gustaría poder verificar formal y automáticamente ciertas propiedades sobre los sistemas representados mediante redes de Petri. Este problema se denomina "model checking" (o comprobación de modelos). En este trabajo hemos hecho una recopilación de los principales resultados sobre model checking para redes de Petri y dos de sus principales extensiones monótonas que son ortogonales: las redes reset y las ν-PNs. Comenzamos con una pequeña introducción a las redes de Petri, sus extensiones y los principales resultados de decidibilidad sobre estas, que sirven para demostrar cuestiones sobre model checking. Una de las principales tareas realizadas en este trabajo ha sido identificar resultados sobre model checking para otros formalismos y adaptarlos a las extensiones de redes de Petri anteriormente citadas. A pesar de la naturaleza eminentemente bibliográfica del trabajo, presentamos también algunos resultados originales. En particular, se da una demostración de la indecidibilidad del recubrimiento repetido para ν-PNs y de aquí se deduce la indecidibilidad del problema de model-checking para cierta lógica temporal para ν-Petri nets. Finalmente, al encontrarnos con que todas las lógicas temporales consideradas son indecidibles para las redes reset y las ν-PNs, discutimos cómo enfrentarnos al problema del model checking en el caso indecidible, mediante dos técnicas diferentes: el unfolding y el cómputo del cover.

Palabras clave: Redes de Petri, redes reset, ν-PN, verificación formal, comprobación de modelos, lógicas temporales, decidibilidad.

Chapter 1

Introduction

Nowadays software is fundamental to our way of life. We need to trust in programs very often: in transports, security, banks... That is why we need tools to verify the properties of these systems we are interested in, and to find why our specifications do not satisfy them, when that is the case. There are several available techniques for this purpose, such as testing, which try to find errors by applying some test cases to the programs, static analysis, or model checking, an automatic formal verification procedure.

In this work we focus on model checking. Model checking is a set of techniques for verifying correctness properties about systems. A model checking algorithm provides a formal and automatic verification of the systems on study. Nowadays model checking has been already used in some real systems. For example, in [56] Lowe study how using a model checking algorithm, an error was discovered in a cryptographic protocol designed by Needham and Schroeder [66], which was thought to be correct during 17 years, until the error was discovered.

In this work we summarize the decidability results for model checking a specific formalism: Petri nets, presenting also some new (modest) results in this area.

1.1 Concurrent and distributed systems

Now it is usual to find systems where several different tasks are performed simultaneously. For example, if you want to listen to music in your PC, you do not need to stop typing your master project or surfing the web. These systems which perform several tasks at the same time or perform several tasks for a common objective, are called concurrent or distributed systems.

Concurrent systems are those systems in which several (maybe interacting) processes may be executed "simultaneously" or in parallel. Concurrent programming is very frequent nowadays, and that is why many programming languages admit concurrency, as Java, C++ or Ada. Unfortunately, to use concurrent programming, we have to cope with several new and important difficulties, such as blocking problems (deadlocks) or the use of shared resources. These difficulties make the verification of concurrent systems a challenging problem.

A distributed system is a system in which several autonomous processes run and interact with each others with a common goal. Distributed computing was originally based on nets of computers which were separated geographically, but nowadays distributed systems can have all their processes running in the same computer. There are several important applications of distributed computing. One of the most important applications is parallel computing. As the problems to be computed have become larger and larger, it is usual to decompose them in smaller parts whose solutions are somehow combined to obtain the solution of the original problem. That is the fundamental concept of parallel computing. Nowadays parallel computing is the dominant technique in computer architectures, and multi-core processors (like dual-core processors or quad-core processors) are becoming very common. Other applications of distributed computing are telephone networks, computer networks as internet, peer-to-peer networks and distributed databases.

These systems have become a very important part of the software in present-day. That is why there are a lot of formalisms to specify these systems. Some of the most important formalisms to specify concurrent and distributed systems are:

- Process algebra like CCS [63], CSP [43] or π-calculus[64], which model the communication and interaction between independent processes.

- Vector addition systems [47], or VASS for short, that are a formalism which consists on vectors of natural numbers which represent the states of the system. VASS can be used to model concurrent systems.

- Petri nets [68], which are equivalent to VASS, but have an intuitive graphic representation.

With concurrent and distributed systems, the complexity of systems is increasing, and the modelers have to cope with a problem: they would like to use

formalisms with a very high expressive power and with good decidability properties, but these two requirements are definitely contradictory. For example, on the one hand, classical finite automata have very good decidability properties, but they cannot model infinite state systems. On the other hand, Turing machines are very expressive, but most of the interesting properties we would like to check are undecidable for them.

In this work we focus on Petri nets, which is a formalism which can express concurrency and infinite states, but has better decidability properties than Turing machines.

1.2 Petri Nets

As we said in the previous section, Petri nets (PN for short) are a formalism to model concurrent and distributed systems. The main differences between Petri nets and the very well-known finite automata are that PNs can cope with an infinite number of states and that the states of a PN are established in a distributed way.

First of all, let us explain informally what a Petri net is. The main components of a Petri net are the places, which contain tokens, and the transitions, which connect them. States of Petri nets are usually called markings. A marking is simply a function which determines how many tokens each place contains at the current state. Graphically, places are represented as circles, and tokens are inside them. The presence of tokens may enable some transitions, in that case, if any of them is fired, then the marking of a Petri net changes by removing tokens from some given places and adding tokens to some other. Each transition has given sets of places from which to remove and to put tokens. If a transition cannot remove the tokens from the appropriate places, then it cannot be fired. That is why we call preconditions of a transition to the set of places that need to have a token (at least) to fire the transition, while we call postconditions of a transition to the set of places in which the transition puts a token when it is fired. Graphically, a transition is represented by a rectangle, and its preconditions and postconditions are identified by means of arrows. An arrow from a place to a transition represents that the place is a precondition of the transition. Analogously, an arrow from a transition to a place represents that the place is a postcondition of the transition.

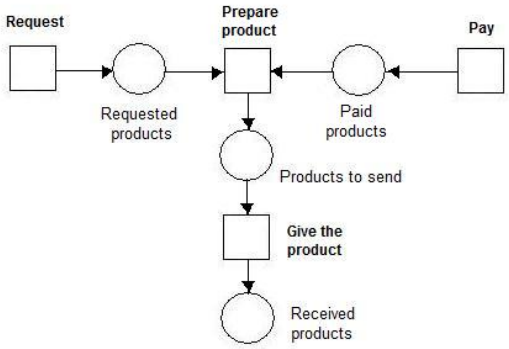

Figure 1.1: A Petri net modelling an online shop

Let us show a toy example to illustrate Petri nets. In figure 1.1 we can see a Petri net which represents how products are sold in an online shop (of course, this is a toy example without any practical use). To buy a product, a client has to request it and to pay for it, and then the seller will prepare the product to send it to the client, and send it.

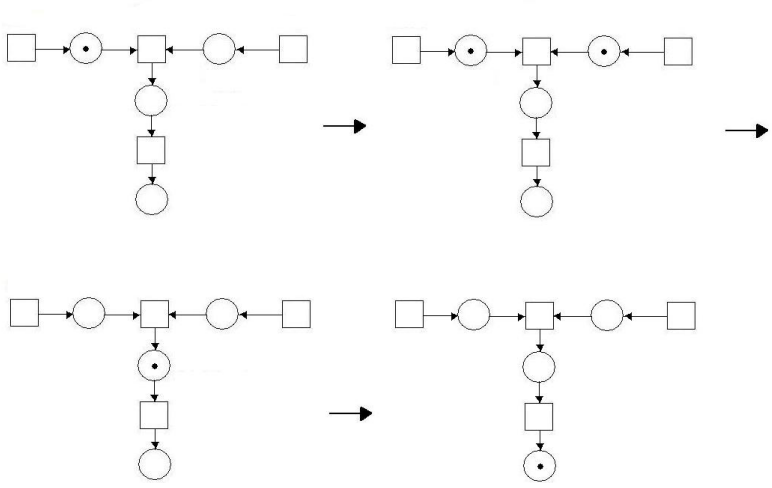

Figure 1.2: Firing transitions

Tokens can represent paid products, requested products, products ready to send to the customer, and sent products, depending on the place in which they lie. At the initial marking, every place is empty because no products have been paid, requested, prepared or sent. The transitions represent the actions of paying a product, requesting a product, preparing a product and sending a product. To know the preconditions and postconditions of the transitions, you only have

to focus on the arrows. For example, to prepare a product, the product must have been paid and requested, so there must be a token in the place "Requested products" and a token in the place "paid products"; and to send a product, it is necessary that the shop has a prepared product, so there must be a token in the place "products to send".

Even though this is a very simple example, it already shows how Petri nets can model concurrency. The actions "Pay" and "Request" can be performed concurrently. It does not matter which of them is fired before, they are independent, though both of them must have been fired at least one time in order to fire "Prepare product" later.

In figure 1.2 we show how a sequence of transitions is fired: The client requests a product, he pays for it and then the seller prepares a product and sends it to the client.

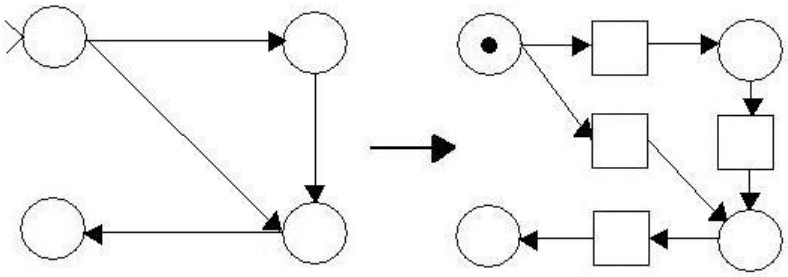

Figure 1.3: An automaton and its corresponding Petri net

Petri nets can be seen as a distributed generalization of finite state automata. Suppose we would like to model a set of finite automata which are running concurrently, and some of their transitions must synchronize in order to occur. Let us explain how that set of automata could be modelled by a Petri net. The Petri net we want to build has a place for each state of each automaton, and a token in a place means that an automaton is in the corresponding state. The transitions would represent the changes of states in automata, as in figure 1.3. When there is a synchronization between two automata (or more), then they change their states "at the same time". With Petri nets we can model this synchronization too, by using only one transition of the Petri net to model both changes of states, like in figure 1.4 , where the changes of states labelled by t synchronize.

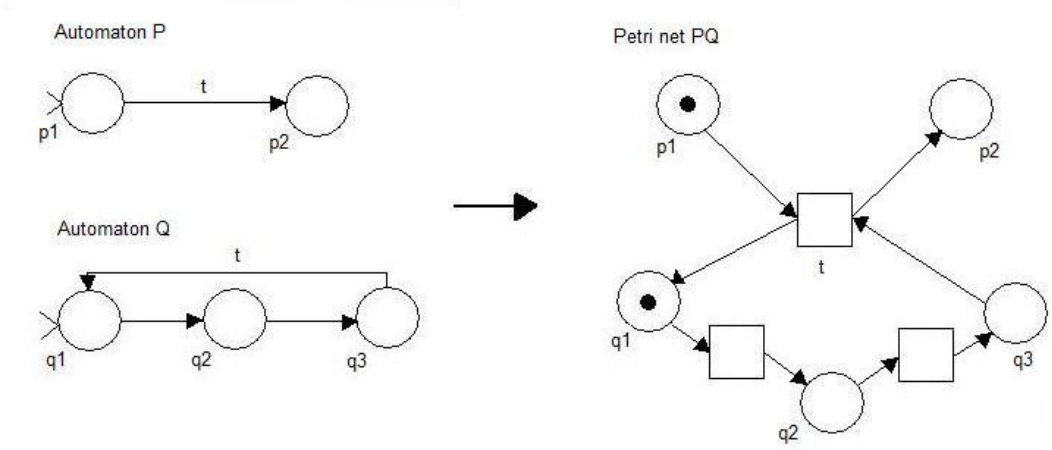

Figure 1.4: Synchronization of two automata

Petri nets were first introduced by Carl Adam Petri[1] in his PhD thesis, in 1962 [68]. Independently, since by the way the importance of the early works by Petri was not recognized at all when it was developed, Karp and Miller introduced vector addition systems in "Parallel program schemata", in 1969 [47]. Later, it was shown that both formalism are equivalent, so that the results obtained for vector addition systems by Karp and Miller, can be translated to Petri nets, too. Since then, Petri nets have been widely studied.

Since Petri nets are a formalism to specify concurrent systems, we would like to be able to verify Petri nets, that is, to automatically guarantee that our systems have the expected behavior. For example, in the previous example, we could be interested in knowing if we can reach a marking in which there is a sent product, or if we can pay infinitely many products without receiving any of them. As a first step, we can ask ourselves whether there are algorithms which decide this kind of questions, that is, whether certain problems are decidable or not. Some decidability problems have been important for the development of the field, as the decidability of the reachability problem [59], which was opened for a long time, or the decidability of the coverability problem [47].

Although Petri nets are a good first approximation to concurrent systems modeling, they do not really have all the expressive power we need to model some systems. Let us show some examples about behaviors of the shop of figure 1.1 that we cannot model with plain Petri nets:

[1]Unfortunately Petri recently died when I was close to finish this work.

- We could like to model how the seller gives the client his money back. Then, the place "Payed products" should be emptied, but we cannot model that behavior with a Petri net, because we cannot recognize when a place has been fully emptied.

- Analogously, we could like to model how the shop sends all the remaining products to the client at a time. All tokens in "Products to send" should be transfered to "Received products", but we cannot transfer all tokens from a place to another with Petri nets.

- We could also like to model how several different clients ask for products. Then we would like to have distinguishable tokens which would represent the money paid, requested products, prepared products and sent products for the different clients.

With these examples it is shown that there are many behaviors of concurrent systems that cannot be modeled with plain Petri nets, so sometimes we need more expressive formalisms nets in order to model the systems we consider. There are numerous extensions of Petri nets in the literature. Some extensions are Turing-complete, but fortunately there are some formalisms which extend Petri nets but are not yet Turing-complete.

A problem that we find when we work with formalisms to model concurrent systems, and in particular with Petri nets, is the fact that there exist a lot of very diverse modeling formalisms, and some of them are equivalent (or equivalent in some sense) to other formalisms which seem to be very different. For example, Petri nets are equivalent to vector addition systems. Part of our work has been to look for results on formalisms which can be applied to Petri nets or Petri net extensions. Some results are obtained as consequences of analogous results on a big variety of formalisms, such as vector addition systems, lossy vector addition systems with inhibitor arcs or counter machines.

Finkel generalized Petri nets to well-structured transition systems [30] in order to transfer some Petri nets decidability results to other formalisms, which are more expressive than Petri nets, but preserve some common properties like monotonicity. Finkel mainly studied the decidability of boundedness, termination and coverability-set problems. Abdulla et al. gave another definition of well-structured transition systems [1], based in their studies about lossy-channels systems and other formalisms. They studied the decidability of covering, inevitability and

simulation problems. Since then, well-structured transition systems have been widely studied in the literature [34], and they subsume important formalisms like reset Petri nets or transfer Petri nets.

The extensions we consider in this work extend Petri nets in two different ways: adding special arcs or extending the nature of tokens. Some of the main Petri net extensions are:

- Affine well-structured nets [36] are nets which admit whole-place operations such as emptying a place or transferring all tokens from one place to another, in a homogeneous way. Reset and transfer nets, which are well structured transition systems, are subsumed by this formalism.

- Nets within nets [78] is a paradigm which contains object nets [79] and nested nets [55]. In this paradigm tokens themselves are Petri nets. Valk introduced this idea in the context of task/flow systems, in which tokens represent tasks which are formalized by Petri nets, and the whole system is again a Petri net.

- Another important formalism is recursive nets [41], which was introduced in order to manage the dynamical creation of objects. It is more expressive than Petri nets, and still preserve good decidability properties.

- Coloured nets [46] are a Turing complete extension which combine Petri nets with other languages. In coloured nets indistinguishable tokens are replaced by data objects, and transitions can modify the data of these tokens. Tokens can be of many different kinds of data so with Coloured Petri nets we can model a lot of structures, as lists or even Turing machines. Data nets [52] (nets in which an order holds between tokens and capable of performing whole-place operations) and ν-Petri nets [73] (which have distinguishable tokens and are capable of creating new names) are subsumed by coloured nets.

As we said before, when we extend formalisms, and in particular when we extend Petri nets, we gain more expressive power, but we lose decidability properties. For example, reachability is decidable for Petri nets, but not for reset, transfer or ν-Petri nets. However, when we consider well-structured transition systems some good decidability results, as the decidability of the coverability problem, are ensured. That is why in this work we focus on well-structured transition systems, and in particular on reset nets and ν-Petri nets.

In this project we are going to focus on how to check some given properties about a Petri net. In particular, we will mainly focus on model checking Petri nets, and what to do if this is not possible in order to obtain some information about the properties we want to check. In the following section we describe what model checking is.

1.3 Model checking

As we have already mentioned, nowadays we have to trust in very complex systems everyday. In many contexts, such as aeronautics or medical software, we even trust human lifes to programs, so we need to be sure that these programs work correctly. In other words, we would like to be sure of certain properties of programs. For example, in the software of an aeroplane we would like to verify that the system does not deadlock unexpectedly.

Model checking is a collection of techniques for automatic formal verification of systems. It was first introduced by Clarke and Emerson in 1981 [11, 21],who recently shared the 2007 Turing Award with J. Sifakis for their work on model checking. Given a formal description $\mathcal{A}$ of the system we want to check, and a property P to check, a model checking algorithm tells us if P holds at the system specification or not. Model checking was first studied for finite state systems and there are algorithms which solve the model checking problem for finite state automata since a long time ago. Intuitively, the classic algorithms consist on building an automaton $\neg\mathcal{P}$ which admits the language of all sequences of transitions which do not satisfy the desired property. Then, the algorithms compute the product automaton $\mathcal{A} \times \neg\mathcal{P}$ between $\neg\mathcal{P}$ and the automaton to check. If the language which $\mathcal{A} \times \neg\mathcal{P}$ admits is empty, then there is not any word of our automaton $\mathcal{A}$ which satisfies the negation of P, that is, every running of the system specification satisfies P.

To perform model checking we need to have a formal model to express the specification of the system to check, but also a language to express the properties that we want to check. In the online shop example, we could be interested in checking properties such as “if a client pays for the product and requests a product then in the future he will receive the product” or “a product is not received until the client pays for it”. These properties are “temporal properties” in some sense. We talk about events which will happen in the future, which will always happen, which never happen, etc. In order to express this kind of properties temporal

logics were introduced by Pnueli[2].

Temporal logics consider atomic predicates which express atomic (not temporal) properties, and several temporal operators and path quantifiers. The main operators are: **X** (which means"at the next state"), **F** (at a future state) and **G**(always) and the classic boolean operators. The main paths quantifiers are **E** (which means that there exist a running from the current state in which a property holds) and **A** (in all runnings from the current state a property holds). Using combinations of the previous operators and path quantifiers one can express many temporal properties. The way in which different temporal logics are obtained is by considering different ways of combining these operators.

There are mainly two kinds of temporal logics: branching time logics and linear time logics. While properties expressed in branching time logics are properties about the different paths that are possible from an initial state, linear time logics express properties about a single path of states (or a single running). The most representative linear time logic is LTL [69, 70, 57, 58] and the most representative branching time logic is CTL [5]. Model checking techniques for verifying these logic properties for finite state systems have been widely studied.

Model checking finite state automata is decidable for all temporal logics as we said before. From this point, a great deal of effort has been devoted to the efficiency of the algorithms, since when we run naive model checking algorithms usually the number of visited states grows exponentially with the size of the system. This problem is called the state explosion problem, and is the main problem we have to face when we check finite state systems. Several techniques have been studied in order to palliate this problem, such as symbolic model checking (which finds a better representation of the system graph), bounded model checking (which checks the formula in a finite number of states), abstractions (which firstly simplify the system) or unfoldings (which unroll the runnings of concurrent systems).

Classic algorithms for model checking finite state systems visit the "interesting" states to check the formula, which could even be all the states of the automaton. As the number of states of the system is finite, the algorithms always terminate. However, we cannot apply these algorithms to model check infinite state systems, because in general their termination is not guaranteed. In particular, Petri nets are a formalism which admits infinite states, so other kind of

[2]Pnueli also died at the beginning of this year

algorithms are needed to model check them. But the battle is not lost: decidability issues for model checking Petri nets are usually reduced to classic problems about Petri nets, as the reachability problem.

In this work we discuss the decidability of model checking for Petri nets and some of its monotonic extensions in the literature, we give a few new results about it and explain what can be done when the decidability results are negative. This problem has been widely studied for Petri nets, but it has not been so thoroughly studied for the extensions we consider here. Of course, the undecidability results for Petri nets hold for extensions too, because the extensions we consider can model a Petri net. But, what about the decidability results? There is not much literature about it, and the literature that there exists is "hidden" in results for other equivalent formalisms as vector addition systems. Unfortunately, we will see that in most of the cases Petri nets decidability results for model checking cannot be extended to Petri net extensions.

So, what can we do with the properties we cannot check because they are undecidable for the formalism we are considering? We can consider semialgorithms, which cannot solve completely the problem, but can give us a partial answer, or search for an answer finitely and perhaps find it. In particular, unfoldings are used for these purposes. Unfoldings are a representation of the different possible runnings of a concurrent system. So with unfoldings we can explore part of the possible runnings of a system, and try to find the answer to our questions. In [29] unfoldings for Petri nets are studied.

1.4 Contributions and outline of the work

Now we detail the outline of the work, which is basically a discussion about model checking Petri nets:

The second chapter is an introduction to Petri nets and Petri nets extensions. The fundamental concepts to understand the rest of the work are explained there. In particular, we will see the definitions and intuitive examples of Petri nets, well structured transition systems, reset nets, transfer nets and ν-Petri nets. The main decidability results are summarized in the second chapter too.

In the third chapter we discuss the model checking problem for Petri nets. Firstly we explain what model checking and temporal logics are. In a second section we summarize the most important classic results about model checking

Petri nets. Then, in the third section of this chapter we summarize the results for Petri net extensions. Finally we explain a pair of techniques which can be used in case of undecidability: unfolding and computation of the cover.

This work is mainly bibliographic, and the biggest part of it is a recompilation of results in the literature. In some cases we have found that some results for several formalisms (mainly lossy vector addition systems), are applicable to Petri nets (mainly reset nets). Despite the fact that most part of the project is bibliographic, there are some original and non trivial contributions:

- In chapter two we give a construction which proofs the undecidability of repeated coverability for ν-Petri nets. In this construction we reduce the problem of repeated coverability for reset nets, which is undecidable, to the same problem for ν-Petri nets.
- In the third chapter, we give a construction which proofs the undecidability of model checking certain temporal logics for ν and reset Petri nets.

Chapter 2

Petri nets and some monotonic extensions

Petri nets are a mathematical formalism to model concurrent and distributed systems, which was introduced by Carl Adam Petri in his doctoral dissertation, in 1962 [68]. It is a very intuitive and simple formalism, because there are only two basic elements: places which may contain tokens and transitions which consume or produce tokens from the places. Petri nets have a very intuitive and easy graphic representation which makes them easily-understandable.

Petri nets have several properties which make them a great formalism to model concurrent and distributed real systems. The way Petri nets change from one state to other (when a transition is fired) is very intuitive. In Petri nets, to change the state some resources are consumed and other resources are produced. This is a very intuitive representation because in many computational contexts changes occur in this way, as in production processes, chemical systems or natural processes. In addition, transitions are fired locally, which means that when a Petri net changes from a state to another, only the part of the net which is related to the transition which is fired is affected. That is why Petri nets are such a good formalism to model concurrency.

Petri nets can also be formalised in other algebraic way using vectors which represent the number of tokens which each place currently carries. Transitions can also be represented as vectors. Then, matrix multiplication represents the firing of transitions, and several algebraic techniques can be (and have been) applied to study Petri nets. In the literature, the term "vector addition system" was used for Petri nets for a while.

Petri nets have become such an important model that most of the techniques for the analysis of Petri nets are available in computer tools. There are tools for modeling, tools for verifying properties about a Petri net, and even tools applicable at specific fields, as business process management (workflow).

When we study a system, we would like to verify some properties about it, but as we said before not all the problems are decidable for a certain formalism. This is another good characteristic of Petri nets: Many interesting problems are decidable for them, such as the reachability problem [59], the boundedness problem and the coverability problem [47]. An important fact to study the model checking problem for Petri nets is that they have an infinite number of states, so model checking algorithms for finite state systems cannot be used for Petri nets. Fortunately, the nice properties we summarized before give us ways to obtain algorithms for model checking Petri nets.

The formalism of Petri nets can be extended for a better modeling of the systems we want to model. That is why in the literature Petri nets are extended by adding special firing rules or considering different types of tokens. In this project we consider extensions of Petri nets of both kinds. Some of the extensions are more expressive than Petri nets, and are not as expressive as Turing machines, which means that they have intermediate properties and expressivity. Mainly, we consider reset Petri nets [3] and ν-Petri nets [73]. When we extend a formalism, some of the decidability properties that it has, are usually lost. For example, the reachability problem is undecidable for Petri nets extended with Reset arcs [18] or for ν-Petri nets [73].

Classic decidability problems, such as the problems mentioned before are important for the main purpose of this work: model checking Petri nets. So in this chapter we explain the fundamental concepts to understand Petri nets and some of its extensions, and we summarize some of the well known decidability results.

2.1 Place/Transition nets

In this section we present the definition of Place/Transition nets, and explain how they work. From now on we will call place/transition nets (P/T for short) to Petri nets without extensions, and the term Petri net will be used for the general case of Petri nets, with or without extensions.

Definition 1 (Place/Transition nets) *A Place/Transition net is a tuple* $\mathcal{N} = (P, T, F)$*, where:*

- P *is a finite set of places,*
- T *is a finite set of transitions,*
- $P \cap T = \emptyset$*,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is the flow relation. The elements of* F *are called arcs.*

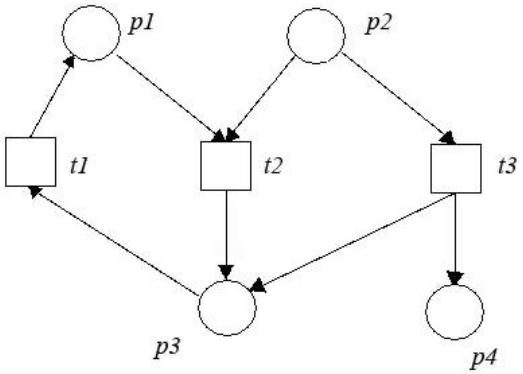

Figure 2.1: A place/transition net

Example 1 *Figure 2.1 shows the graphic representation of a place/transition net , with a set of places* $\{p1, p2, p3, p4\}$*, a set of transitions* $\{t1, t2, t3\}$ *and arcs* $\{(t1, p1), (p1, t2), (p2, t2), (p2, t3), (p3, t1), (t2, p3), (t3, p3), (t3, p4)\}$*. The places are represented by circles, the transitions by squares and the arcs by arrows.*

States in the field of place/transition nets are usually called markings. A marking of a P/T is a mapping $\mathcal{M} : P \to \mathbb{N}$, that is, a finite multiset of places. If $p \in P$ and $\mathcal{M}(p) = n$, then we say that there are n tokens in the place p, so we can consider a marking as a function that sets the number of tokens that there are currently in each place of the net. A marking $\mathcal{M}$ enables a transition $t \in T$ (the transition t is enabled) if for every $p \in {}^\bullet t$, $\mathcal{M}(p) > 1$, ie, in p there is at least one token, where:

$${}^\bullet t = \{p \in P \mid (p, t) \in F\}.$$

If a transition is enabled, then it can occur and we say that the transition can be fired. As we said before, a marking is a multiset of places. Let us consider + and - the addition and the subtraction of multisets. That is, if A and B are multisets of places, p is a place and $A(p)$ and $B(p)$ denote the number of appearances of p in A and B, then $A+B$ is the multiset of places where for every place p, $(A+B)(p) = A(p) + B(p)$ and if $B \subseteq A$, then $A - B$ is the multiset of places where for every place p, $(A-B)(p) = A(p) - B(p)$. The occurrence of a transition t leads to a new marking $\mathcal{M}'$ (we denote $\mathcal{M} \xrightarrow{t} \mathcal{M}'$) such that:

$$\mathcal{M}' = (\mathcal{M} -^{\bullet} t) + t^{\bullet}$$

where $t^{\bullet} = \{p \in P \mid (t,p) \in F\}$. $\mathcal{M} -^{\bullet} t$ can be performed because to fire the transition it have to be enabled, so ${}^{\bullet}t \subseteq \mathcal{M}$. This means that when a transition t is fired, a token is removed from each place in ${}^{\bullet}t$, and a token is added to each place in $t^{\bullet}$.

Let us show an example of how a transition is fired:

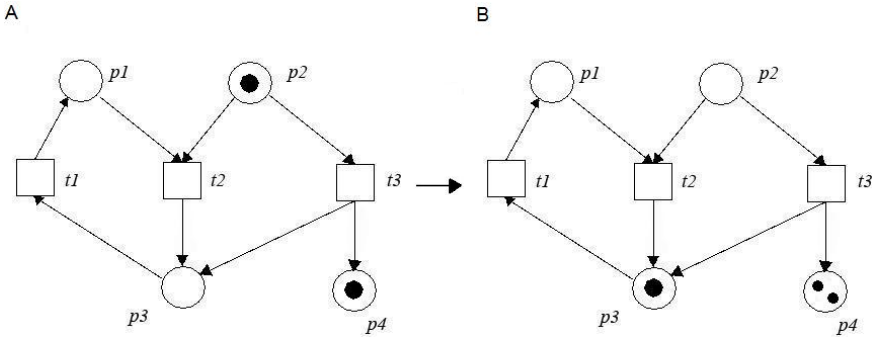

Figure 2.2: Firing a transition

Example 2 *Figure 2.2 (left) shows the P/T of Example 1, with a token in p2 and p4 that is, the P/T marked by a marking $\mathcal{M}$ in which $\mathcal{M}(pi) = 0$ for $i = 1, 3$, $\mathcal{M}(p2) = 1$ and $\mathcal{M}(p4) = 1$.*

$\mathcal{M}$ enables t3, but does not enable t1 nor t2. If t3 is fired, then a token is removed from p2 and a token is added to p3 and p4. In this way, we obtain the marking represented by Figure 1.2 (right).

If we have a sequence of occurrences $\mathcal{M}_1 \xrightarrow{t_1} \mathcal{M}_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} \mathcal{M}_n$ then we write $\mathcal{M}_1 \xrightarrow{t_1 \ldots t_{n-1}} \mathcal{M}_n$.

Given a marking $\mathcal{M}_0$, we say that another marking $\mathcal{M}$ is reachable from $\mathcal{M}_0$ if there exists a sequence $t_1...t_n$ such that $\mathcal{M}_0 \xrightarrow{t_1...t_n} \mathcal{M}$.

Given a place/transition net $\mathcal{N} = (P, T, F)$, an initial marking $\mathcal{M}_0$, and a final marking $\mathcal{M}$, we define the free language of $\mathcal{N}$ with respect to $\mathcal{M}_0$ and $\mathcal{M}$ as:

$$\mathcal{L}(\mathcal{N}, \mathcal{M}_0, \mathcal{M}) = \{\omega \mid \mathcal{M}_0 \xrightarrow{\omega} \mathcal{M}\}$$

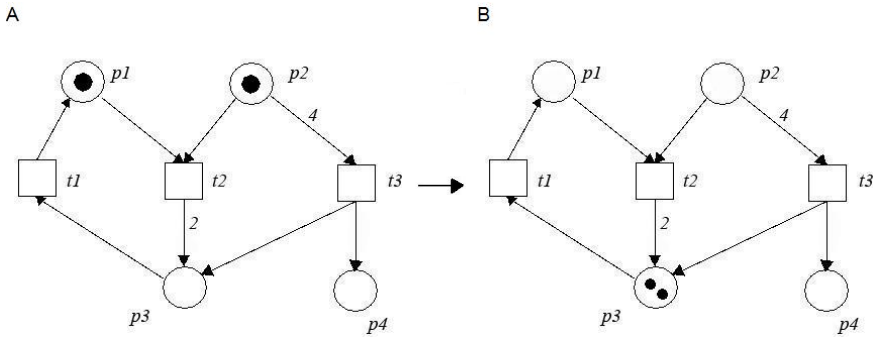

Figure 2.3: Place/transition nets with weight arcs

Sometimes we will assume that transitions are labelled by a labelling function $L : T \to P$. Then, to build the language of a Petri net we will consider the labels of the fired transitions instead of the transitions themselves.

In the literature several similar formalisms are considered, like P/Ts with arc weights. A place/transition net with arc weights is a P/T with a function which assigns a weight (in $\mathbb{N}$) to each arc. The weight of an arc (p, t) (or (t, p)) specifies the number of tokens to be removed from (or added to) p when firing the transition.

Example 3 *Figure 2.3A shows a Petri net with weight arcs. The labels of the arrows represent the weights of the corresponding arcs. If an arrow is not labelled, it means that the weight of the corresponding arc is 1. Transition t2 is enabled, but transition t3 is not, because there are not four or more tokens in p2. If t2 is fired, then the marking in figure 2.3B is reached.*

Place/transition nets with arc weights do not increase the expressive power of place/transition nets. Indeed, given a P/T with arc weights $\mathcal{N}$, you can build a P/T $\mathcal{N}'$ (probably with much more places and transitions), such that $\mathcal{N}$ and $\mathcal{N}'$, have the same behavior. For more detail on this, see [71]. In the following, we will use the most convenient formalism, P/Ts with or without arc weights, at each time.

Another feature which does not increment the expressive power of P/T nets is place capacities. The capacity of a place is the maximum number of tokens that there can be at the place. That means that when a transition is fired, it has to be enabled in the sense of P/T, but there must also be enough "space" to put the created tokens at the postcondition places of the transition, so that if there is not enough space, then the transition cannot be fired. If all the places of a P/T net with capacities have a finite associated bound, then the system we are specifying is finite-state.

P/T nets can simulate P/T nets with place capacities with an easy construction which consist on adding a place p' for each place p (called complement places) which will represent the number of tokens that we can still add to p. Then, for every place p and for every reachable marking $\mathcal{M}$, if p' is the complement place of p, then $\mathcal{M}(p) + \mathcal{M}(p') = c(p)$, where $c(p)$ is the capacity of p. Let us show an example:

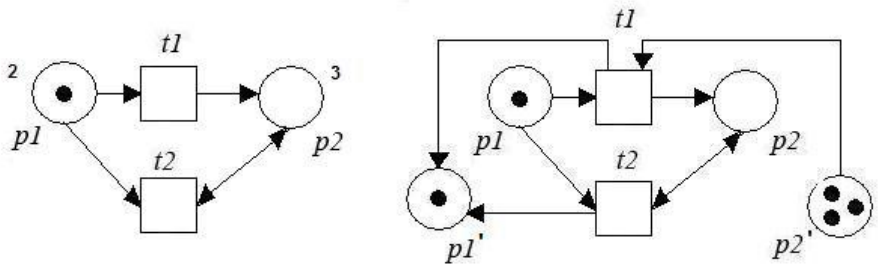

Figure 2.4: Petri nets with weight arcs

Example 4 *In figure 2.4 a P/T with place capacities is shown. It has two places named* p1 *and* p2 *with capacities 2 and 3, respectively. If the place* p2 *were marked with three tokens and the place* p1 *with a token, the transition* t1 *could not be fired,*

though it would be enabled in the sense of P/T, because there would not be space in the place $p2$ for another token.

Next to the P/T with place capacities, there is a plain P/T. These two systems have equivalent behaviors. To build this P/T, two places $p1'$ and $p2'$ have been added in order to control the number of tokens of the places $p1$ and $p2$. Accordingly, the flow relation is modified so that in order to put a token in a place p, one must be able to remove a token from the complement place p'.

2.1.1 Decidability results for place/transition nets

As we said, place/transition nets have good decidability properties, which have been useful to prove some facts about model checking. Let us summarize some of them:

The reachability problem consists on deciding if, given a Petri net $\mathcal{N}$, an initial marking $\mathcal{M}_0$ and a final marking $\mathcal{M}$, $\mathcal{M}$ can be reached from $\mathcal{M}_0$. This problem was open for a long time [40], and it was not until 1981 when Mayr demonstrated the decidability of the reachability problem [59]. Kosaraju and Lambert simplified the proof in 1982 and 1992, respectively [49, 51]. In July 2010 Leroux gave an easier proof [53]. The exact complexity of reachability remains open. While all the previous algorithms have non primitive recursive complexity the best lower bound known is EXPSPACE [54]. Other decidability problems, such as the liveness problem and the deadlock-freedom problem, are recursively equivalent to reachability, so that they are also decidable [40]. Some of the model checking problems we will discuss later have been reduced to the reachability problem in the literature, too. That is why the decidability of reachability is important for our purposes.

Let $\mathcal{M}$ and $\mathcal{M}'$ be two markings. We say that $\mathcal{M}' \geq \mathcal{M}$ if for every place p in the considered Petri net, $\mathcal{M}'(p) \geq \mathcal{M}(p)$. The coverability problem consists on, given a PN, an initial marking $\mathcal{M}_0$ and a final marking $\mathcal{M}$, deciding if there exists another marking $\mathcal{M}'$ such that $\mathcal{M}' \geq \mathcal{M}$ and $\mathcal{M}'$ is reached from $\mathcal{M}_0$. One of the reasons why coverability is important is because it can be used to specify safety properties, such as "it is impossible to do *something bad*", because we can specify that the preconditions to fire a *bad* transition can not be covered. That makes coverability a crucial problem for model checking P/T nets.

The boundedness problem consists on deciding if the set of reachable markings is finite. In fact, for every place p there is a finite bound $b \in \mathbb{N}$ such that $\mathcal{M}(p) \leq b$

for every reachable marking $\mathcal{M}$, then the set of reachable markings is finite. That is, the number of tokens in each place is bounded on the reachable markings if and only if the net is bounded. That is so because if we have such a bound b_i for each place p_i in a net with n places, then the highest number of possible reachable markings there may be is $\prod_{1 \leq i \leq n} b_i$. Conversely, if we have a place p without a bound for the number of tokens in it, that means that for every reachable marking $\mathcal{M}$, there is a greater marking $\mathcal{M}'$ such that $\mathcal{M}(p) > \mathcal{M}'(p)$ for some place p. By an inductive reasoning, that lead us to the fact that there are infinite many reachable markings. The importance of the boundedness problem to model checking lies in the fact that if there is a finite number of reachable markings, then the system we want to check is a finite states system, so we could apply the classic techniques.

A problem closely related to the boundedness problem is the place-boundedness problem. The place-boundedness problem consists on deciding, given a Petri net $\mathcal{N}$ and a place p, if there is a bound for the number of tokens in p on for all reachable markings. One could think that this problem is equivalent to the boundedness problem, but this is not the case. In fact, when place-boundedness is decidable, boundedness is decidable too, because of the boundedness problem characterization we explained before; but the fact that boundedness is decidable does not mean place-boundedness is decidable too, by the way for several interesting Petri net extensions the first problem is decidable, but not the second one.

Karp and Miller introduced in [47] a new construction: the coverability tree. The coverability tree is an abstraction of the reachability tree which is precise enough to decide some important problems: with this construction the coverability, boundedness and place boundedness problems for P/T nets were shown to be decidable.

Let us explain how to build the coverability tree: The nodes of the coverability tree are vectors which represent sets of markings. The vectors have one component for each place which represents the number of tokens in the place. The value of the components may be a natural number or the symbol ω, which represents the possibility to have an unbounded number of tokens in the place. The coverability tree is built by applying the following rules:

- The root of the coverability tree is the initial marking.
- The sons of a node n are the markings which can be reached by firing a transition from n.

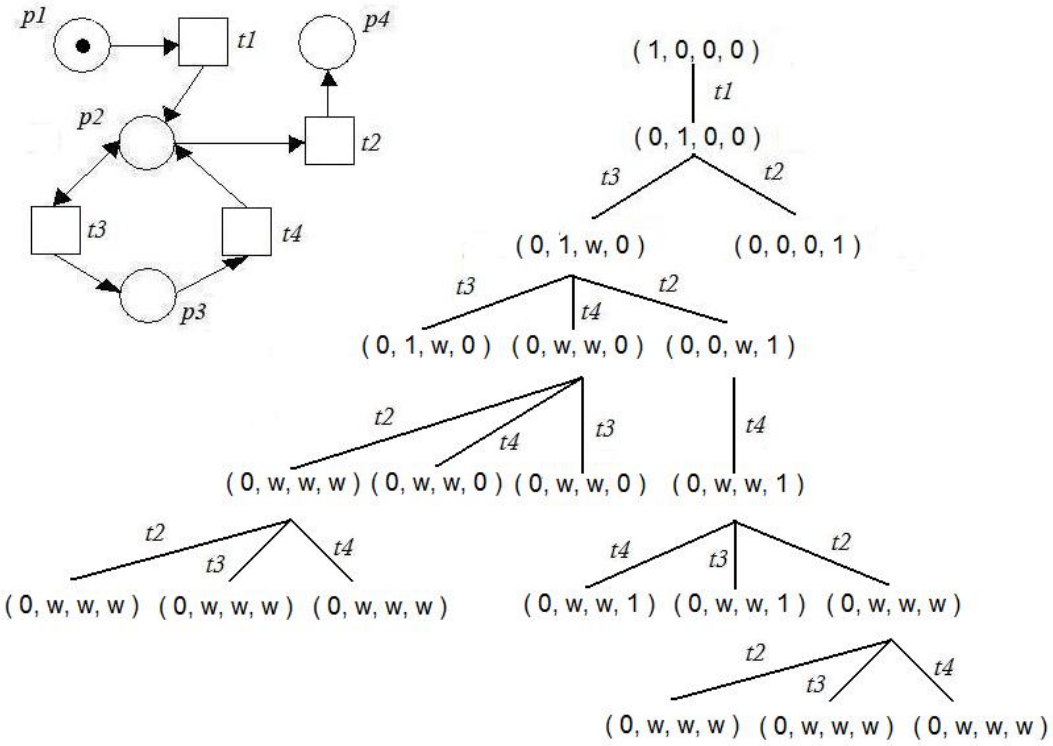

Figure 2.5: The coverability tree of a P/T net

- If a new node n repeats a marking of the path from the root to n, then n is a terminal node (it does not have any child).
- If a new node n is greater than another node n' of the path from the root to n, then the values of n which are greater than the corresponding values of n' are replaced by ω.
- If a node n has an ω in some of its positions then its children will have an ω at the same position, too.

Example 5 *In figure 2.5 a P/T net and its corresponding coverability tree are shown. Let us focus on the second and the third levels of the tree. When we fire* $t3$ *from* $(0,1,0,0)$*, we reach the marking* $(0,1,1,0)$*. As* $(0,1,1,0) \geq (0,1,0,0)$*, in the third level the child is* $(0,1,\omega,0)$ *instead of* $(0,1,1,0)$*. Then, the* ω *value is carried until the terminal nodes.*

We said before that a ω in a vector means that the corresponding place possibly has an unbounded amount of tokens in this following branch. That is because

when we reach a marking $\mathcal{M}$ from another marking $\mathcal{M}'$ by firing a transition sequence σ and $\mathcal{M}' \subseteq \mathcal{M}$, then the same sequence is enabled at $\mathcal{M}'$, and we can fire it again, reaching a marking which is greater than $\mathcal{M}$. That is an important property: monotonicity. In the next section we will explain why this is such an important property.

With the coverability tree it is easy to know if a P/T net is unbounded or not: the net is unbounded iff ω appears in the tree. A place is unbounded iff ω appears in the corresponding component of a node of the tree. We can also check the coverability of a marking $\mathcal{M}$ just by searching a node which covers $\mathcal{M}$.

Example 6 *The P/T of figure 2.5 is unbounded. In fact, only the place p*1 *is bounded. That is because there is not any node in the tree with an ω in the first component, and there is a node $(0, w, w, w)$ in the tree.*

Unfortunately, not every problem is decidable for P/T nets. Given two Petri nets $\mathcal{N}_1 = (P_1, T_1, F_1)$ and $\mathcal{N}_2 = (P_2, T_2, F_2)$ with the same number of places, and a bijection $f : P_1 \to P_2$, then we have a natural bijection F between markings of $\mathcal{N}_1$ and markings of $\mathcal{N}_2$. The containment problem consists on deciding if for every reachable marking $\mathcal{M}$ of $\mathcal{N}_1$, $F(\mathcal{M})$ is a reachable marking of $\mathcal{N}_2$. Hack proved in [40] that the containment problem is undecidable for P/Ts.

2.1.2 Well-structured transition systems

Some of the decidability results of the previous subsection are due to the monotonicity of place/transition nets and Dickson's lemma. Finkel was the first one who gave a definition of well-structured transition system (WSTS for short) [30], which are essentially systems with the two previous properties (or analogous ones). Then, some of the decidability results for P/Ts were generalized to WSTS [32], for example, to reset nets. Let us see the fundamental concepts needed to understand what is a WSTS:

A quasi ordering $\leq$ on a set X is a reflexive and transitive relation on X. If $\leq$ is also antisymmetric, then it is a partial ordering.

Given a quasi ordering $\leq$ on a set A, and a subset X of A, we define $\uparrow X = \{x' \mid \exists x \in X, x' \geq x\}$. In particular, if $x \in A$, $\uparrow x = \{x' \mid x' \geq x\}$. An upward-closed set $U \in X$ is a set such that, for every $u \in U$, if there is an $x \in X$ such

that $x \geq u$ then $x \in U$. If U is an upward-closed set, then $\uparrow U = U$. For every $x \in X$, $\uparrow x$ is an upward-closed set. A basis of an upward-closed set U is a set B such that $U = \cup_{x \in B} \uparrow x = \uparrow B$.

Definition 2 *A well quasi ordering (we will denote wqo for short) is a quasi ordering such that for any infinite sequence $x_1, x_2, ... \in X$, there are $i, j \in \mathbb{N}$ with $i < j$ and $x_i \leq x_j$.*

Next we recall the definition of transition system, in order to define monotone transition systems and, finally, well-structured transition systems:

Definition 3 *A (labelled) transition system is tuple $\mathcal{T} = (S, L, \rightarrow, s_0)$ where S is a possibly infinite set of states, L is a set of transition labels, $\rightarrow$ is a transition relation $\rightarrow \subseteq S \times L \times S$ and $s_0 \in S$ is the initial state.*

We will write $s_1 \xrightarrow{l} s_2$ instead of $(s_1, l, s_2) \in \rightarrow$. If there exist $s_1, s_2, ..., s_n \in S$ and $l_1, l_2, ..., l_{n-1} \in L$ such that $s_1 \xrightarrow{l_1} s_2 \xrightarrow{l_2} s_3 \xrightarrow{l_3} ... \xrightarrow{ln-1} s_n$, and $w = l_1 l_2 l_3 ... l_{n-1}$ then we write $s_1 \xrightarrow{w} s_n$

The immediate predecessors (resp. successors) of a state s of a transition system is the set $Pre(s) = \{s' \in S \mid s' \rightarrow s\}$ (resp. $Post(s) = \{s' \in S \mid s \rightarrow s'\}$). A transition system $\mathcal{T} = (S, L, \rightarrow, s_0)$ has effective successor if there exists an algorithm such that, given an state $s \in S$, calculates Post(s). A transition system has effective-Pred basis if there is an algorithm that given a state s of the system, calculates a finite basis for $\uparrow Pre(\uparrow s)$.

Given a state s of a transition system S, we call $Post^*(s)$ (resp. $Pre^*(s)$ to the set of states s' of this transition system such that there exists a chain of transitions α such that $s \xrightarrow{\alpha} s'$ (resp. $s' \xrightarrow{\alpha} s$).

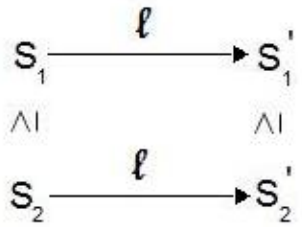

Figure 2.6: Monotonicity

A labelled transition system $\mathcal{T} = (S, L, \rightarrow, s_0)$ is monotone with respect to $\leq$ if for every $s_1, s_2, s_1' \in S$ such that $s_1 \xrightarrow{l} s_2$ and $s_1 \leq s_1'$, there exists an $s_2' \in S$ such that $s_1' \xrightarrow{l} s_2'$ and $s_2 \leq s_2'$ (see Figure 2.6). An ordering $<$ is a quasi ordering such that, given to elements s_1 and s_2 of a set, then $s_1 < s_2 \Leftrightarrow s_1 \leq s_2$ and $s_2 \not\leq s_1$. If the quasi ordering $\leq$ is replaced by the (partial) ordering $<$, then the transition system is strictly monotone.

Definition 4 (Well-structured transition system) *A transition system $\mathcal{T} = (S, L, \rightarrow, s_0)$ is a well-structured transition system with respect to an ordering $\leq$ on S, if:*

- *$\mathcal{T}$ is monotone with respect to $\leq$.*
- *$\leq$ is a well-quasi order.*

A WSTS is strict if it is a well-structured transition system and is strictly monotone.

The finite reachability tree [32] of a WSTS $\mathcal{T} = (S, L, \rightarrow, s_0)$ (FRT for short) is un unordered tree which has states of the system as nodes. There are live and dead nodes. The root of the tree is the state s_0, and it is live. A dead node does not have any child. A live node has one child for each of its successors. If along the path from s_0 to a node s there exists another node s' such that $s \geq s'$, then we say that the node s' is subsumed by the node s, and the node s is dead. Otherwise, the node is live.

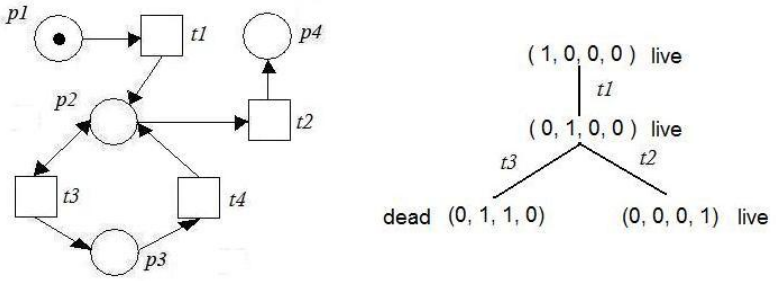

Figure 2.7: The reachability tree of a PN

Example 7 *Figure 2.7 shows a place/transition net and its corresponding reachability tree. The nodes of the tree are vectors which have four elements, representing*

the number of tokens at each place in a marking. The initial marking is the marking with one token in $p1$ *and no token at the rest of the places. The corresponding node is the root of the tree and it is live.*

At this marking, only $t1$ *can be fired, so the root only has a child:* $(0, 1, 0, 0)$. *The child is live and has two childs:* $(0, 1, 1, 0)$ *(by firing* $t3$*) and* $(0, 0, 0, 1)$ *(by firing t2). The first child is dead, because his parent* $(0, 1, 0, 0)$ *is subsumed by it; so it does not have any child.* $(0, 0, 0, 1)$ *is live, but does not have any child. Now there is not any other node to visit, so the tree is completed.*

The reachability tree of a WSTS has been shown to be always finite. That is because, applying König's lemma it suffices to prove that there cannot be an infinite branch and the quasi order in a WSTS is a well-quasi order, so given an infinite sequence of markings (the nodes of the branches of our tree), there are two markings such that one subsumes the other. So considering the way the trees are constructed, there cannot be any infinite branch.

Finkel and Schnoebelen showed that a WSTS has a non-terminating computation iff its FRT contains a subsumed node, so termination has been shown to be decidable for any WSTS with a decidable ordering and an effective Succ [34]. Therefore, the termination problem for WSTS is equivalent to ask if there is a dead node in the corresponding reachability tree. That is because if there is not any dead node, then all the branches of the tree end at a live node. That is, all firing sequences end at a state from which any transition cannot be fired, so the system terminates for any running.

Conversely, if there is a dead node n, then this node subsumes another node n' which is its ancestor. That means that $n \geq n'$ and there exists a firing sequence σ such that $n' \xrightarrow{\sigma} n$. WSTS are monotone, so the firing sequence σ can be fired from n, reaching a state n_1 such that $n \leq n_1$, so the same sequence can be fired from n_1. By an inductive reasoning we can conclude that there is a infinite transition sequence and the system does not terminate.

There is another problem we can solve with the finite reachability tree: the boundedness problem for WSTS with strict monotonicity. A WSTS with strict monotonicity is unbounded if and only if there exists a dead node n such that its subsumed node n' is strictly smaller than itself. That is because if we have a dead node n, and its subsumed node n' such that $n' < n$ and $n' \xrightarrow{\sigma} n$ then, because of the strict monotonicity, σ can be fired from n reaching an state n_1 such that $n < n_1$, ans then by a trivial inductive reasoning we obtain that σ can be fired

infinitely many times from n_1 reaching bigger states every time. That means that the system is unbounded.

Conversely, if there is not any subsumed node with the previous property then the branches of the tree may end in a live node without childs (so the sequence of transitions ends and is finite), or in a dead node n such that its subsumed node n' is $n' = n$. In that case, the corresponding sequence of transitions can be fired again from n, but in this case no new markings are added. Therefore, the reachable states are those which appear as nodes of the reachability tree, that is a finite amount of nodes, so the number of states of the system is finite, and the system is bounded.

The coverability problem was shown to be decidable for P/T nets by Karp and Miller in [47]. The algorithm constructs the Karp Miller (K-M for short) tree, which is a finite representation of $\downarrow Post^*(\downarrow s)$, where s is a state of the system. These ideas can be used in the general case [38, 39]. For any WSTS we call cover of the state s to the downward closure of the reachability set of a Petri net with s as initial state. If we want to solve the coverability problem for a given state s, and we can compute the *cover* of the initial state of a net, then it is easy to solve the problem just by checking if s is in the *cover*. We say that this method works forward, because the successors of a state are computed. Unfortunately, the *cover* of a system is not computable in general, so this method of solving the coverability problem cannot be applied in all the cases.

A method which works backwards (computing the predecessors of the state we want to check if it is coverable) was proposed later. Although the previous forward algorithm did not always work, coverability is shown to be decidable for WSTS under very general effectiveness hypothesis. More precisely, whenever there are an effective Pred-basis and a decidable ordering a new algorithm is proposed [1, 33]. Specifically, given a state s and an initial state s_0, the algorithm that decides whether s can be covered from s_0 computes $\uparrow Pred^*(\uparrow s)$, the set of states from which s can be covered. This set is computed as the limit of the sequence $I_0 \subseteq I_1 \subseteq I_2 ...$ where $I_0 = \uparrow s$ and $I_{n+1} = I_n \cup Pred(I_n)$. This sequence is computable because the considered systems have effective Pred-basis, and it is finite because $\leq$ being a well-quasi ordering implies that infinite increasing sequences of upward-closed sets eventually stabilize, and indeed for all $i \in \mathbb{N}$, I_i is upward-closed because of monotonicity. To solve the problem of coverability we only have to check if s_0 is in $\uparrow Pred^*(\uparrow s)$ or not.

Let us explain why Place/transition nets are WSTSs. To do that we have to explain why the already defined quasi ordering for P/T nets is a well-quasi ordering and why Petri nets are monotone with respect to this order.

First we are going to explain why the quasi ordering "$\leq$" is a well-quasi ordering, that is, why for any infinite sequence of markings $\mathcal{M}_1, \mathcal{M}_2...$, there are $i, j \in \mathbb{N}$ such that $i < j$ and $\mathcal{M}_i \leq \mathcal{M}_j$. The markings of a net with n places can be seen as vectors of natural numbers with n components, that is, elements of $\mathbb{N}^n$. Consider the following quasi ordering in $\mathbb{N}^k$: given two vectors of the same size $n \in \mathbb{N}$, then "$u \leq v \Leftrightarrow u(i) \leq v(i)$" for all $i \leq n$ where $u(i)$ and $v(i)$ are the i-th components of the vectors. As the previous quasi order is a well-quasi ordering for $\mathbb{N}^n$ (Dickson's lemma), and markings and their defined order can be seen as vectors of $\mathbb{N}^n$ with the previous notion of ordering, the ordering of markings is a well-quasi ordering.

Now, we explain why P/T are monotone with respect to $\leq$. Let $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}'_1$ be three markings of a P/T such that $\mathcal{M}_1 \xrightarrow{t} \mathcal{M}_2$ and $\mathcal{M}_1 \leq \mathcal{M}'_1$. $\mathcal{M}'_1$ has in each place, as less, the same number of tokens as $\mathcal{M}_1$ (because of how the ordering is defined), so if t is enabled at $\mathcal{M}_1$, then t is enabled at $\mathcal{M}'_1$ too, that is, for every $p \in {}^\bullet t$, $\mathcal{M}'_1(p) > 1$, because $\mathcal{M}_1(p) > 1$ and $\mathcal{M}_1(p) \leq \mathcal{M}'_1(p)$. When we fire the transition t from $\mathcal{M}_1$, we reach the marking $\mathcal{M}_2 = (\mathcal{M}_1 -^\bullet t) + t^\bullet$, and when we fire it from $\mathcal{M}'_1$, we reach the marking $\mathcal{M}'_2 = (\mathcal{M}'_1 -^\bullet t) + t^\bullet$. Then, as $\mathcal{M}_1 \leq \mathcal{M}'_1$,

$$\mathcal{M}_2 = (\mathcal{M}_1 -^\bullet t) + t^\bullet \leq (\mathcal{M}'_1 -^\bullet t) + t^\bullet = \mathcal{M}'_2$$

Therefore, $\mathcal{M}_2 \leq \mathcal{M}'_2$, and the monotonicity of P/T nets is proved.

In the following section we explain some of the extensions of place/transition nets that have been defined in literature. Some of these extensions are WSTSs too, so they satisfy the properties above, which are very useful for model checking.

In the next two sections some extensions of place/transition nets, which will increment the expressive power of Petri nets, are presented. We focus on significant extensions with a simple definition which preserve monotonicity.

2.2 Special firing rules

In the literature there are several ways of extending place/transition nets. The first way we are going to consider is by adding special firing rules. Special arcs are

added and these arcs, intuitively, will empty places (reset arcs) or transfer tokens from one place to another (transfer arcs). There are more ways of extending place/transition nets by adding special rules, but we consider reset and transfer arcs because they preserve monotonicity, which brings good properties as we said before, for example the decidability of the coverability problem.

A non monotone extension is Petri nets with inhibitor arcs. Intuitively, what an inhibitor arc pointing to a place does is to check if in the place there is some token. If there is a token, the transition cannot be fired. If the place is empty and the transition is enabled, then it can be fired. With inhibitor arcs the expressive power of P/T nets is greatly increased. In fact, Petri nets with two inhibitor arcs have the same expressive power as Turing machine, because they can simulate Minsky machines with two counters. [65].

Reset nets were first introduced by T. Araki and T. Kasami in [3], and their properties have been extensively studied since then [3, 18, 6, 10, 48]. What makes reset nets interesting is that their transitions can empty places. That increments the expressive power of P/T nets, so they are more suitable to model several systems which cannot be modelled by P/T nets. The formal definition of a reset Petri net (or Petri net with reset arcs) is the following:

Definition 5 (Petri net with reset arcs) *A Petri net with reset arcs is a tuple* (P, T, F, R)*, where* (P, T, F) *is a P/T net and* $R \subseteq P \times T$ *is a relation containing the so called reset arcs.*

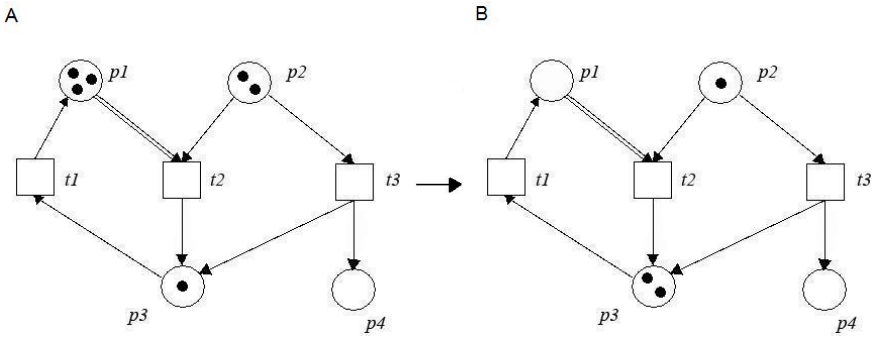

Figure 2.8: Firing a transition with a reset arc

Reset arcs do not affect the notion of enabled transition, but when a transition

t is fired, and there is a reset arc $(p,t) \in R$, from p to t, then the new marking $\mathcal{M}$ that is reached, is such that:

- $\mathcal{M}(p) = 1$, if $(t,p) \in F$.
- $\mathcal{M}(p) = 0$, otherwise.

The new marking is set as for plain nets at the rest of the places. That means that a reset arc (p,t) empties the place p when the transition t is fired.

When one sees the definition of reset nets for the first time, one could ask himself in what way they have more expressive power than P/T. In fact, to empty a place it seems it is enough to have a transition which removes one token at a time from the place we want to empty. The problem of this approach is that we do not know when the place we are emptying is indeed empty, so we do not know when to stop removing tokens. We would need to be able to check if a place is empty to apply the previous reasoning to empty this place, and this is not possible in a P/T. That is why we cannot simulate a reset net with a P/N.

Example 8 *Figure 2.8A shows a Petri net with a reset arc* $(p1, t2)$*, represented by* $\Rightarrow$*. If* $t2$ *is fired, then* $p1$ *is emptied, and the marking showed in figure 2.8B is reached.*

Next we present Petri nets with transfer arcs, which is another important extension of Petri nets which have been studied in the literature [73, 18] and it is quite useful for the modelling of broadcast [26].

We explained in our introduction how Petri nets can be seen as a generalization of finite state automata. We showed how to model nets of automata running concurrently, which synchronized in some transitions. Now we want to model a new kind of communication between the different parts of the net: broadcasting. Broadcasting is a way of message-passing in which one component (a process) of the system sends information to all the rest of components, simultaneously. Intuitively, Petri nets are a good approach to model this behavior, because of how transitions works (it is easy to model the sending of information from one place to as many places as we want). The problem is that when we want to model broadcast, we need to be able to perform whole-place operations, and in particular to transfer all the information of one token to some others. That is why P/T nets are not enough. We need a more expressive formalism: Transfer nets.

Definition 6 (Petri net with transfer arcs) *A Petri net with transfer arcs is a tuple (P, T, F, Tr), where (P, T, F) is a P/T and $Tr \subseteq P \times T \times P$ is a relation defining the transfer arcs.*

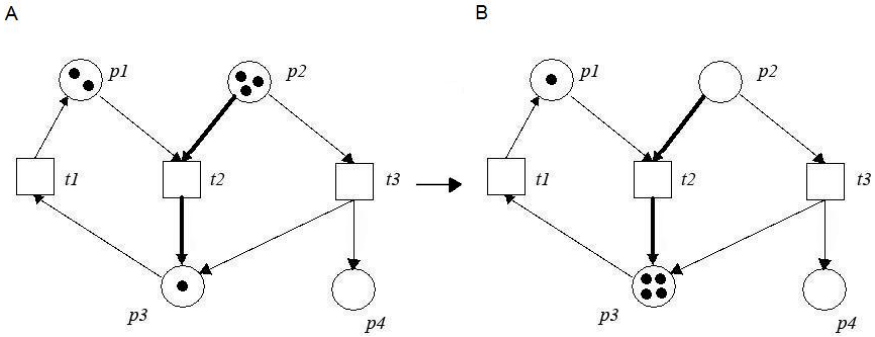

Figure 2.9: Firing a transition with a transfer arc

Let us suppose that we have a Petri net with transfer arcs $\mathcal{N} = (P, T, F, Tr)$, and a marking $\mathcal{M}$. Here the enabled transitions are the same as for the place/transition net (P, T, F), but when we fire a transition $t \in T$, then the new reached marking $\mathcal{M}'$ is such that, if (p, t, p') is a transfer arc, then:

- $\mathcal{M}'(p) = 1$, if $(t, p) \in F$.
- $\mathcal{M}'(p) = 0$, otherwise.

and

- $\mathcal{M}'(p') = \mathcal{M}(p') + \mathcal{M}(p) + 1$, if $p' \in t^\bullet$ and $p' \notin {}^\bullet t$.
- $\mathcal{M}'(p') = \mathcal{M}(p') + \mathcal{M}(p) - 1$, if $p' \notin t^\bullet$ and $p' \in {}^\bullet t$.
- $\mathcal{M}'(p') = \mathcal{M}(p') + \mathcal{M}(p)$, otherwise.

The markings of the rest of the places are modified as for plain Petri nets. The intuitive meaning of a transfer arc is that they are able to transfer all the tokens at some place to another.

Example 9 *Figure 2.9A shows a Petri net with a transfer arc (represented by a black arrow). If transition t2 is fired, then p2 is emptied, and its three tokens are added to p3. Then, the marking showed in figure 2.9B is reached.*

The reason P/T cannot transfer all tokens from a place to another in a correct way is similar to the reason they cannot empty places as reset arcs need. We could think that having a transition which translates a token from one place to another is enough to transfer all tokens from the first place to the second one, but this is not true. As in the case of reset nets, we would need to check when the first place is empty, but we cannot do that in a P/T. Therefore, we do not know when we could stop firing the transition.

Reset and transfer Petri nets are subsumed by another formalism: affine well-structured nets. Affine well-structured nets (AWNs) were first introduced by Finkel, McKenzie and Picaronny in [36]. Basically, an AWN is a generalization of P/Ts, reset and transfer nets in which the firing of a transition consists on three steps (the first and the last steps are the same as in a P/Ts): In the first step tokens are removed from the proper places; in the second one, whole-place operations, such as reseting a place or transferring all tokens of a place to another, take place; and in the last one, tokens are added to the proper places.

As P/Ts can be defined as vectors which represent the markings and the transitions consist on the addition of a vector to the marking, AWNs are frequently defined algebraically too: affine nets can be seen as a set of affine functions (that is why they are called affine!). In [36] affine nets are defined as follows:

Definition 7 *An affine net $\mathcal{A}$ with p places is a tuple (T, P, M, H), where T is a set of transition labels, C and H are vectors of size p, labelled by elements of T, and M is a matrix of size $p \times p$, labelled by elements of T, too. Given two markings $\mathcal{M}$ and $\mathcal{M}'$, the transition relation is defined as follows:*

$$\mathcal{M} \xrightarrow{t} \mathcal{M}' \Leftrightarrow (\mathcal{M} \geq P_t) \wedge (\mathcal{M}' = M_t * (\mathcal{M} - P_t) + H_t).$$

where markings are seen as vectors and P_t, M_t and H_t are the components of P, M and H labelled by t.

In fact, P_t can be seen as the vector which gives us the preconditions of t, and H_t as the vector which gives us the postconditions of t. The matrix M determines the whole-place operations of the net. In fact, if we add certain constraints to M we obtain different extensions. For example, if $M = \{Id\}$, then the affine net is a P/T. If the matrices are diagonal with only zero and one values the net is a reset net (a zero in a position of the diagonal of the matrix of a transition correspond to a place which will be emptied if the transition is fired). And if the matrices contain only zero and ones, then the net is a transfer net.

In [36] it is proved that affine nets are effective well-structured transition systems, so coverability and termination are decidable. In the next section the main decidability results for reset and transfer nets are summarized.

Decidability results for Petri nets with special firing rules

It is useful to know some of the decidability results of transfer and reset PN, in order to face our main purpose: model checking Petri nets. So let us summarize some of these decidability (or undecidability) properties: coverability, termination, boundedness, place boundedness and reachability.

First of all, let us explain why transfer and reset Petri nets are WSTS: We consider the same ordering on the markings "$\geq$" as for P/T systems, which is a well-quasi order. Reset and transfer nets are monotone with respect to "$\geq$", because if we have two markings $\mathcal{M}_1$ and $\mathcal{M}_2$ such that $\mathcal{M}_2 \geq \mathcal{M}_1$, then if we fire a transition t from $\mathcal{M}_1$, reaching a marking $\mathcal{M}'_1$, then we can fire t from $\mathcal{M}_2$ because $\mathcal{M}_2 \geq \mathcal{M}_1$. When we fire t from $\mathcal{M}_2$ we reach a marking $\mathcal{M}'_2$ which satisfies $\mathcal{M}'_2 \geq \mathcal{M}'_1$. That is because:

- If t does not have reset or transfer arcs, then when we fire it every thing works like in P/Ts, which are monotone.
- If t has a reset arc then all the tokens of the corresponding place p are removed when we fire t from $\mathcal{M}_1$ or $\mathcal{M}_2$. The normal arcs have the same effect as in P/Ts. Therefore, $\mathcal{M}'_1(p) = \mathcal{M}'_2(p)$ and $\mathcal{M}'_2(p') \geq \mathcal{M}'_1(p')$ for all $p' \neq p$.
- If t has a transfer arc then when we fire t from $\mathcal{M}_1$ or $\mathcal{M}_2$ all the tokens of the corresponding place p_1 are moved to the proper place p_2. The normal arcs have the same effect as in P/Ts. So $\mathcal{M}'_1(p_1) = \mathcal{M}'_2(p_1)$, $\mathcal{M}'_1(p_2) = \mathcal{M}_1(p_1) \geq \mathcal{M}_2(p_1) = \mathcal{M}'_2(p_2)$ and $\mathcal{M}'_2(p') \geq \mathcal{M}'_1(p')$ for all $p' \neq p$.

So reset and transfer nets preserve monotonicity, and are WSTSs. As we said before, coverability has been shown to be decidable for WSTS, so coverability is decidable for transfer and reset nets. In [32] Finkel gave a proof of the decidability of termination for certain kind of WSTSs, as we recalled in a previous section, and this result is also applicable to transfer and reset nets.

Let us focus in another problem, which we will use later to prove some model checking undecidability results: the repeated coverability problem. Given a Petri

net $\mathcal{N}$ with an initial marking $\mathcal{M}$ and a marking $\mathcal{M}'$, the problem of deciding if $\mathcal{M}'$ can be repeatedly covered from $\mathcal{M}$, that is, if there exists an infinite sequence of fireable transitions from $\mathcal{M}$ such that $\mathcal{M}'$ is covered infinitely often, is called the repeated coverability problem. This problem is decidable for P/T nets, but it has been shown to be undecidable for transfer nets, by reducing to it the halting problem for counter machines [28]. Repeated coverability is undecidable for reset nets too [20].

The problem of boundedness is a tricky problem. Actually, it was erroneously proved to be decidable for reset nets in [48]. Boundedness had always been linked to the problem of coverability until undecidability of boundedness was proved in [18]. Reset nets are not strictly monotonic, so we cannot use any algorithm based in this property to solve the boundedness problem. This result is surprising, since boundedness is decidable for transfer-nets, which are supposed to be more powerful than reset nets [33]. In [19] it was shown that boundedness is decidable for reset nets with two reseteable places, but it is undecidable for reset nets with three reset arcs.

The problem of place-boundedness is undecidable for reset nets, because if it was decidable, boundedness would be decidable for reset nets, too. It was shown in [18] that place-boundedness is undecidable for transfer nets too, by reducing boundedness for reset nets to place-boundedness for transfer nets. In fact, it is easy to simulate a reset net by a transfer net, only by adding to the transfer net we want to build a dummy place to transfer there the tokens of a place when this place is supposed to be reseted. This was an important result because it was the first time that a formalism had decidable boundedness and undecidable place-boundedness.

The reachability problem is undecidable for transfer and reset Petri nets having two extended arcs. In [18] Dufourd , Finkel and Schnoebelen reduce reachability for nets with inhibitor arcs to the reachability problem for reset or transfer nets. Reachability is undecidable for nets with inhibitor arcs with two or more inhibitor arcs. Using the construction in [18], undecidability of reachability is shown for transfer and reset Petri nets which have two or more extended arcs.

2.3 Extended tokens

Another way in which P/Ts are extended in the literature is by considering distinguishable tokens. One of the most important cases of Petri nets with extended tokens is coloured Petri nets [46]. Coloured Petri nets combine Petri nets with the functional programming language Standard ML. The result is a very powerful language which is, in fact, Turing complete. In coloured nets indistinguishable tokens are replaced by data objects, and transitions can modify the data of these tokens. Tokens can be of many different kinds of data, so with Coloured Petri nets we can model many structures, as lists or even Turing machines.

Two of the main extensions of P/T nets with distinguishable tokens that are subsumed by coloured Petri nets are ν-PN [73] and Data nets [52]. Data nets are nets where tokens carry a datum taken from a linearly ordered domain and capable of performing whole-place operations and broadcasting. In ν-PNs tokens are not ordered, but they are distinguishable. In this formalism there is a mechanism to create fresh pure names [67]. ν-PNs have been used in the literature for different purposes [17, 42, 72, 74]. The nets with extended tokens we are going to consider in this project are ν-PN nets, which are not Turing complete (unlike coloured PN) and preserve monotonicity. We do not consider Data nets for simplicity, since they have an intricate definition.

Definition 8 (ν-Petri net) *Let Var be a set of variables, including ν. A ν-Petri net (ν-PN for short) is a tuple $\mathcal{N} = (P, T, F)$, where P is a set of places, T is a set of transitions such that $P \cap T = \emptyset$ and $F : (P \times T) \cup (T \times P) \to Var$, is a partial function such that for all $t \in T$, $\{F(t,p) \mid p \in t^\bullet\} \backslash \{\nu\} \subseteq \{F(p,t) \mid p \in {}^\bullet t\}$ and $\nu \notin \{F(p,t) \mid p \in {}^\bullet t\}$, where $t^\bullet = \{p \in P \mid F(t,p)$ is defined$\}$ and ${}^\bullet t = \{p \in P \mid F(p,t)$ is defined$\}$*

Here, F is a partial function that labels each arc with a of variable. An arc of the form (p,t), with $p \in P$ and $t \in T$, cannot be labelled by ν, because ν will be reserved to create fresh names. Given $t \in T$, we denote by $Var(t)$ the set of variables that label the arcs of the form (p,t) or (t,p), with $p \in P$.

Example 10 *In Figure 2.10 we can see a ν-PN. The arcs are labelled with variables (x, y or ν).*

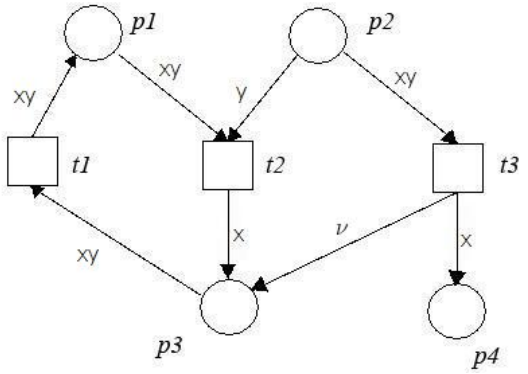

Figure 2.10: A ν-PN

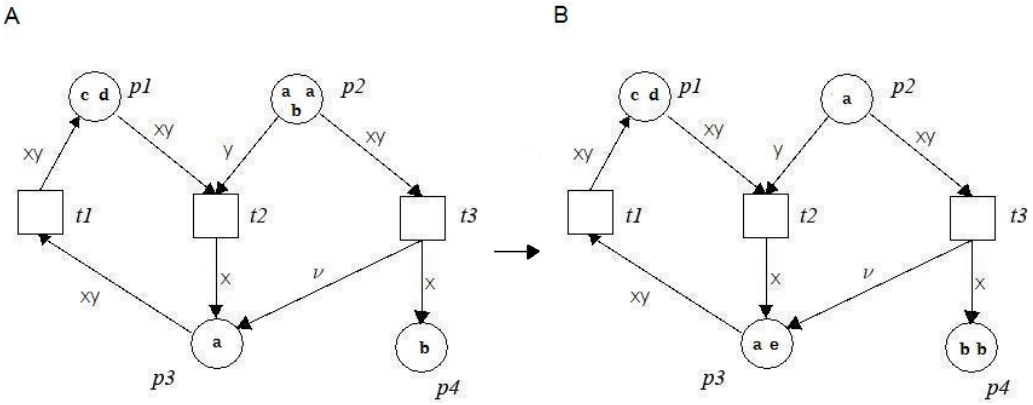

Figure 2.11: Firing a transition in a ν-PN

Let us consider Id a set of names, and SId the set of multisets of Id. In a ν-PN a marking $\mathcal{M} : P \to SId$ is a function which assigns a multiset of Id to each place.

Given a transition $t \in T$ of a ν-PN, a mode $\sigma : Var(t) \to Id$ is an injection that instantiates each variable of $Var(t)$ to a name of Id.

We say that a transition t is enabled with a mode σ for a marking $\mathcal{M}$, if for all $p \in P$, $\sigma(F(p,t)) \subseteq \mathcal{M}(p)$ and $\sigma(\nu) \notin S(\mathcal{M})$, where $S(\mathcal{M})$ is the set of names in the marking $\mathcal{M}$, that is $S(\mathcal{M}) = \{n \in Id \mid \exists p \in P \text{ such that } n \in \mathcal{M}(p)\}$. Then, t can be fired, and a new marking $\mathcal{M}'$ is reached, where $\mathcal{M}'$ is such that, for all $p \in P$:

$$\mathcal{M}'(p) = (\mathcal{M}(p) - \sigma(F(p,t))) + \sigma(F(t,p)).$$

In ν-Petri nets names are distinguishable, but can only be compared in terms of equality or inequality, not as in Data nets, in which tokens are ordered. Modes make the comparison of tokens possible in ν-Petri nets. Intuitively, when we are modelling a system with a ν-Petri net and we want that two arcs of a transition add or remove tokens of the same name, that is, to ensure the equality of these names, then we have to label both arcs by the same variable. Then, if the transition is fired at a mode, the tokens of both arcs will be tokens of the same name, which is the name that the mode assigns to the variable which labels both arcs. Reciprocally, if we want that two arcs of the same transition add or remove tokens of different names, then we have to label them with different variables. Then, if the transition is fired at a mode, the tokens which the two arcs add or remove will be of different names. That is because modes are injections, so the names assigned by the mode to each of the two different variables are different. Reciprocally, if we want that two arcs of the same transition add or remove tokens of different names, then we have to label them with different variables. Then, if the transition is fired at a mode, the tokens which the two arcs add or remove will be of different names because modes are injections.

Example 11 *In Figure 2.11A* $t3$ *is enabled with a mode* σ*, such that* $\sigma(x) = b$ *and* $\sigma(y) = a$*. It is also enabled with a mode* σ' *such that* $\sigma'(x) = a$ *and* $\sigma'(y) = b$*. Transitions* $t1$ *and* $t2$ *are not enabled with any mode. We consider mode* σ*. If we fire* $t3$*, we reach the new marking in figure 2.11. Firing* $t3$*, the arc* $(t3, p3)$*, labelled by* ν *creates the new name* e *in* $p3$*.*

Before explaining the most important decidability issues about ν-Petri nets, let us focus on the fact that they are WSTSs. First of all, we have to consider an order on the markings that we have not explained yet. When one starts to imagine a quasi order on the marking of a ν-Petri net, one tends to think of the following notion of order:

$$\mathcal{M}_1 \leq \mathcal{M}_2 \Rightarrow M_1(p) \subseteq M_2(p) \text{ for all } p \in P.$$

The problem of this intuitive approach is that, in fact, this is not a well-quasi order, that is, there exists an infinite sequence of incomparable markings. For example, consider a chain of markings $\mathcal{M}_i$ such that for all place $p \in P$ and for all $i \in \mathbb{N}$, $\mathcal{M}_i(p) = \{a_i\}$, with $a_i \in Id$ and $a_i \neq a_j$ for all $i \neq j$. This is an infinite chain of incomparable markings, so the quasi order we are considering is not a well-quasi order.

The previous quasi order is in fact too restrictive, because it does not take into account that the names of a ν-Petri net are pure, so they should mean the same if we rename them. In [73] a notion of equivalence of markings is explained in order to define the proper quasi order. They define the relation $\equiv_\alpha$ as the smaller equivalence relation between markings such that $\mathcal{M} \equiv_\alpha \mathcal{M}\,[\eta/\eta']$ with $\eta \in S(M)$ and $\eta \notin S(M)$, that is, the smaller equivalence relation in which a marking $\mathcal{M}$ is equivalent to all markings such that are $\mathcal{M}$ renamed. They also proof that the behavior of a ν-Petri net does not change when it is correctly renamed. That is because in a ν-Petri net tokens are distinguishable only in terms of equality and inequality, as we said before.

Finally, the notion of order they consider is the following. We say that $\mathcal{M} \sqsubseteq \mathcal{M}' \Leftrightarrow \mathcal{M}(p) \subseteq \mathcal{M}'(p)$ for all $p \in P$. Then:

$$\mathcal{M}_1 \sqsubseteq_\alpha \mathcal{M}_2 \Leftrightarrow \text{there exists a marking } \mathcal{M}'_1 \text{ such that } \mathcal{M}'_1 \equiv_\alpha \mathcal{M}_1 \text{ and } \mathcal{M}'_1 \sqsubseteq_\alpha \mathcal{M}_2.$$

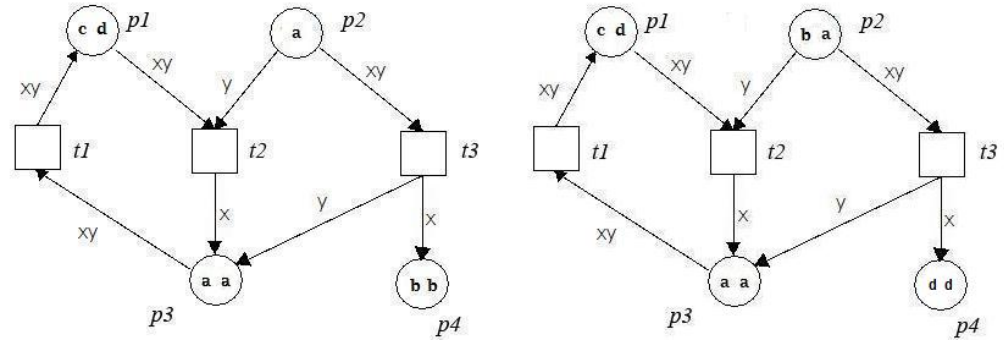

Figure 2.12: Order of ν-PN

Example 12 *Figure 2.12 shows the same ν-PN with two different markings. The second marking is greater than the first one. That is because is we rename the first marking, replacing b by d, then we obtain a marking which is contained in the second marking in the sense of $\sqsubseteq$.*

In [73] it is proved that $\sqsubseteq_\alpha$ is a well quasi order, and that the transition relation in ν-PNs is strictly monotone. Therefore, ν-PNs are WSTSs. Decidability of properties of ν-Petri nets have been studied in the literature. Let us explain some of the most important results.

- In [73] it is proved the undecidability of the reachability problem. The way this result is proved is by giving a construction in which the reachability of a counter machine with two counters is reduced to the reachability of a marking of a ν-Petri net. The reachability problem is undecidable for counter machines with two counters, so it is undecidable for ν-Petri nets too.

- As we said before, ν-PN are strictly well structured, so that coverability, termination and boundedness are decidable. Again, the fact that boundedness is decidable is important for model checking purposes, because if the system is bounded, then we can apply standard model checking results for finite systems. The coverability problem is the same as for P/T nets but taking the new defined order (if we did not consider this order, then we could not deduce the decidability of the problem from the fact that the formalism is a WSTS).

- Repeated coverability is undecidable. This is a new result, so we are going to prove it in detail.

In order to prove the undecidability of the repeated coverability for ν-Petri nets, we are going to simulate a reset Petri net by means of a ν-PN, strongly enough to reduce repeated coverability for reset nets to the same problem for ν Petri nets. Since this problem is undecidable for reset nets, the construction will prove that the problem is undecidable for ν-Petri nets, too.

Intuitively, the construction will divide each transition of the reset net into three transitions: in the first and the third transitions tokens will be added to and removed from the proper places, like in P/Ts, and in the second transition the places which have to be reseted will be emptied. For each transition there will be two places associated with each place, and one global place, representing in what phase of the transition the net is, in order to be sure that a different transition does not start until the current transition has ended. We model emptying a place as follows. Intuitively, each place p will have another related place $c(p)$ in which there is always one single token. This name will represent the "real tokens" of the place p. That is, p will have tokens of several names, but only the tokens that are of the same name as the token in $c(p)$ will be considered "real". Therefore, to empty a place p, we only have to remove the token in $c(p)$ and create a new name, adding it to $c(p)$. As the name is new, there is no token with this name in p, so this place does not have any "real" token.

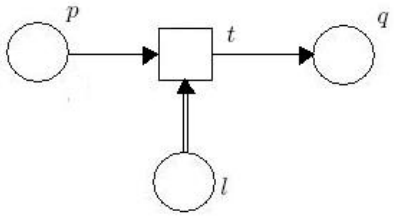

Figure 2.13: A reset net

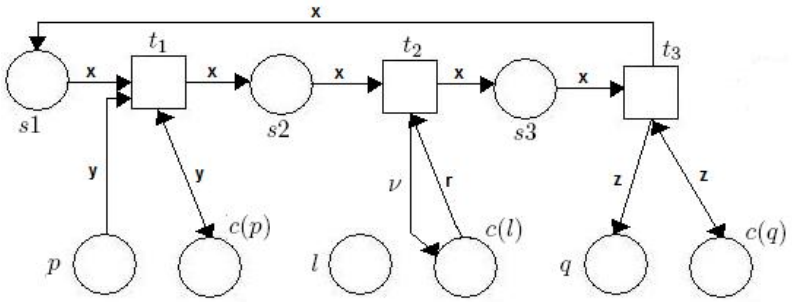

Figure 2.14: The corresponding ν-PN

Let us show how the construction proceeds:

Definition 9 *Given a reset net $\mathcal{N} = (P, T, F, R)$ we build a ν-Petri net $\mathcal{F}(\mathcal{N}) = (P', T', F')$ such that:*

- $P' = P \cup \{c(p) \mid p \in P\} \cup \{s1\} \cup \{s2_t \mid t \in T\} \cup \{s3_t \mid t \in T\}$.
- $T' = \{t_1, t_2, t_3 \mid t \in T\}$.
- – $F'(s1, t_1) = F'(s2_t, t_2) = F'(s3_t, t_3) = F'(t_1, s2_t) = F'(t_2, s3_t) = F'(t_3, s1) = x$, *for all* $t \in T$.
  – $F'(p, t_1) = F'(c(p), t_1) = F'(t_1, c(p)) = y_p$, *for all* $p \in P$ *and for all* $t \in T$ *such that* $(p, t) \in F$.
  – $F'(t_3, p) = F'(c(p), t_3) = F'(t_3, c(p)) = y_p$, *for all* $p \in P$ *and for all* $t \in T$ *such that* $(t, p) \in F$.
  – $F'(c(p), t_2) = r$, *for all* $p \in P$ *and for all* $t \in T$ *such that* $(p, t) \in R$.
  – $F'(t_2, c(p)) = \nu$, *for all* $p \in P$ *and for all* $t \in T$ *such that* $(p, t) \in R$.
  – F' *is undefined in the rest of the cases.*

We assume that x is different to any other variable.

Given a place p of a ν-Petri net and $\eta \in Id$, let $\mathcal{M}(p)(\eta)$ denote the number of tokens carrying name η in p. Given a marking $\mathcal{M}$ of the reset net we want to simulate, we have to define the corresponding marking of the ν-Petri net that we build, that will represent $\mathcal{M}$. In fact, we are going to define a corresponding set of markings instead of a single marking, because we do not need to make precise the trash which is created when a place is reset. We define $\mathcal{M}^*$ as the set of markings of the net $\mathcal{F}(\mathcal{N})$ such that, if for all $p \in P$, $c_p \in Id$ are pairwise distinct, and $\bullet \in Id$, then, if $\mathcal{M}' \in \mathcal{M}^*$ then:

- $\mathcal{M}'(s_1) = \{\bullet\}$.
- $\mathcal{M}'(s_{2t}) = \mathcal{M}'(s_{3t}) = \emptyset$ for all $t \in T$.
- $\mathcal{M}'(c(p)) = \{c_p\}$.
- $\mathcal{M}'(p)(c_p) = \mathcal{M}(p)$

At $\mathcal{M}'$ there could be some tokens with other names at each place p, which would correspond to the created trash. The concrete value of c_p does not really matter, since names are abstract, so renaming the net does not change its behavior.

Example 13 *Figure 2.13 shows a reset net with a transition with three arcs. The arc (l, t) is a reset arc. Figure 2.14 shows the ν-Petri net obtained by applying the previous construction to simulate that reset net.*

We have explained intuitively how the construction works, but does it really works? Let us show that if $\mathcal{M}_1$ and $\mathcal{M}_2$ are markings of the considered reset net, then if $\mathcal{M}_1 \to^* \mathcal{M}_2$ then for all $\mathcal{M}'_1 \in \mathcal{M}_1^*$, there exists a marking $\mathcal{M}'_2 \in \mathcal{M}_2^*$ such that $\mathcal{M}'_1 \to^* \mathcal{M}'_2$; and for all $\mathcal{M}'_1 \in \mathcal{M}_1^*$ and $\mathcal{M}'_2 \in \mathcal{M}_2^*$, $\mathcal{M}_1 \to^* \mathcal{M}_2 \Rightarrow \mathcal{M}_1 \to^* \mathcal{M}_2$.

Proposition 10 *Given a reset net $\mathcal{N} = (P, T, F, R)$ and the corresponding ν-Petri net $\mathcal{F}(\mathcal{N}) = (P', T', F')$ obtained by the application of the construction detailed above, then for all $t \in T$, we have*

- *If $\mathcal{M}_1 \xrightarrow{t}_{\mathcal{N}} \mathcal{M}_2$ then, for all $\mathcal{M}'_1 \in \mathcal{M}_1^*$ there exists a marking $\mathcal{M}'_2 \in \mathcal{M}_2^*$ such that $\mathcal{M}'_1 \xrightarrow{t_1 t_2 t_3}_{\mathcal{F}(\mathcal{N})} \mathcal{M}'_2$*

- *Given two markings $\mathcal{M}_1$ and $\mathcal{M}_2$ of $\mathcal{N}$, if there exist $\mathcal{M}'_1 \in \mathcal{M}_1^*$ and $\mathcal{M}'_2 \in \mathcal{M}_2^*$ such that $\mathcal{M}'_1 \xrightarrow{t_1 t_2 t_3}_{\mathcal{F}(\mathcal{N})} \mathcal{M}'_2$, then $\mathcal{M}_1 \xrightarrow{t}_{\mathcal{N}} \mathcal{M}_2$.*

Proof:

- First, let us suppose that $\mathcal{M}_1 \xrightarrow{t}_{\mathcal{N}} \mathcal{M}_2$, and let us prove that given a marking $\mathcal{M}'_1 \in \mathcal{M}_1^*$, there exists $\mathcal{M}'_2 \in \mathcal{M}_2^*$ such that $\mathcal{M}'_1 \xrightarrow{t_1 t_2 t_3}_{\mathcal{F}(\mathcal{N})} \mathcal{M}'_2$.

  At the marking $\mathcal{M}'_1$ the transition t_1 is enabled. That is because $s1$ is marked with one token at $\mathcal{M}'_1$, $c(p)$ is marked with one token too, and all the places p which are preconditions of t are marked in $\mathcal{M}'_1$ with a token (or more) with the same name as the token in $c(p)$. That is because t is enabled in the reset net, so the places which are preconditions of t are marked, so the corresponding places has to be marked too, taking into account how the construction was defined.

  When t_1 is fired from $\mathcal{M}'_1$, the net $\mathcal{F}(\mathcal{N})$ reaches a marking $\mathcal{M}''$ such that:

  - $\mathcal{M}''(s2_t) = \{\bullet\}$.
  - $\mathcal{M}''(s1) = \mathcal{M}''(s3_t) = \emptyset$.
  - $\mathcal{M}''(p)(c_p) = \mathcal{M}'(p)(c_p) - 1$, where $p \in {}^\bullet t$.
  - The rest of the places remain the same as in $\mathcal{M}'_1$.

  At this new marking, the only fireable transition is t_2, because the rest of the transitions are not enabled, since the "s" places that are their preconditions are not marked. The only preconditions for t_2 are $s2_t$ and $c(l)$, for each place l which is a place with a reset arc reaching t. At $\mathcal{M}''$ $c(p)$ is marked for all $p \in P$, and $s2_t$ is marked too, so t_2 is fireable. When we fire t_2 we reach a marking $\mathcal{M}'''$ such that:

  - $\mathcal{M}'''(s3_t) = \{\bullet\}$.
  - $\mathcal{M}'''(s1) = \mathcal{M}'''(s2_t) = \emptyset$.
  - $\mathcal{M}'''(c(l))(c_l) = 0$ and $\mathcal{M}'''(c(l))(c_{l1}) = 1$, where $c_{l1} \in Id$ is a new name, for all l such that $(l, t) \in R$.
  - The rest of the places remain the same as in $\mathcal{M}''_1$.

  Again, at this new marking only a transition is fireable: t_3. The reason is analogous to the previous case. When we fire this transition from $\mathcal{M}'''$ we reach a new marking $\mathcal{M}''''$ such that:

- $\mathcal{M}''''(s1) = \{\bullet\}$.
- $\mathcal{M}''''(s3_t) = \mathcal{M}''''(s2_t) = \emptyset$.
- $\mathcal{M}'''(p)(c_p) = \mathcal{M}'''(p)(c_p) + 1$, for all $p \in t^\bullet$, where c_p is the colour of the token in $c(p)$.
- The rest of the places remain the same as in $\mathcal{M}_1'''$.

We have reached a marking $\mathcal{M}''''$. Let us check explain why $\mathcal{M}'''' \in \mathcal{M}_2^*$. The marking $\mathcal{M}_2$ is such that:

- If $(p,t) \in R$ and $(t,p) \in F$ then $\mathcal{M}_2(p) = 1$, so for every $\mathcal{M}_{2*} \in \mathcal{M}_2^*$, $\mathcal{M}_{2*}(p)(c_p) = 1$, where c_p is the colour in $c(p)$.
- If $(p,t) \in R$ and $(t,p) \notin F$ then $\mathcal{M}_2(p) = 0$, so for every $\mathcal{M}_{2*} \in \mathcal{M}_2^*$, $\mathcal{M}_{2*}(p)(c_p) = 0$, where c_p is the colour in $c(p)$.
- If $(p,t) \notin R$ and $(p,t) \in F$ and $(t,p) \notin F$ then $\mathcal{M}_2(p) = \mathcal{M}_1(p) - 1$, so for every $\mathcal{M}_{2*} \in \mathcal{M}_2^*$, $\mathcal{M}_{2*}(p)(c_p) = \mathcal{M}_1(p) - 1 = \mathcal{M}_{1*}(p)(c_{p'}) - 1$, where c_p and $c_{p'}$ are the colours in $c(p)$ at the marking $\mathcal{M}_{2*}$ and $\mathcal{M}_{1*}$ respectively, and $\mathcal{M}_{1*} \in \mathcal{M}_1^*$.
- If $(p,t) \notin R$ and $(p,t) \notin F$ and $(t,p) \in F$) then $\mathcal{M}_2(p) = \mathcal{M}_1(p) + 1$, so for every $\mathcal{M}_{2*} \in \mathcal{M}_2^*$, $\mathcal{M}_{2*}(p)(c_p) = \mathcal{M}_1(p) + 1 = \mathcal{M}_{1*}(p)(c_{p'}) + 1$, where c_p and $c_{p'}$ are the colours in $c(p)$ at the markings $\mathcal{M}_{2*}$ and $\mathcal{M}_{1*}$ respectively, and $\mathcal{M}_{1*} \in \mathcal{M}_1^*$.
- Otherwise $\mathcal{M}_2(p) = \mathcal{M}_1(p)$, so for every $\mathcal{M}_{2*} \in \mathcal{M}_2^*$, $\mathcal{M}_{2*}(p)(c_p) = \mathcal{M}_1(p) = \mathcal{M}_{1*}(p)(c_{p'})$, where c_p and $c_{p'}$ are the colours in $c(p)$ at the markings $\mathcal{M}_{2*}$ and $\mathcal{M}_{1*}$ respectively, and $\mathcal{M}_{1*} \in \mathcal{M}_1^*$.

Let us check that $\mathcal{M}'''' \in \mathcal{M}_2^*$

- The place $s1$ is marked in $\mathcal{M}''''$ with a token of a name which is different to all the other names in the marking, as the markings in $\mathcal{M}_2^*$.
- For all $t \in T$, $s2_t$ and $s3_t$ are not marked in $\mathcal{M}''''$, as the markings in $\mathcal{M}_2^*$.
- For all $p \in P$, in the marking $\mathcal{M}''''$ each place $c(p)$ is marked by a token of a different name c_p, which is not necessarily the same name as the name at the marking $\mathcal{M}_1'$.
- If $(p,t) \in R$ and $(t,p) \in F$ then the name of the token in $c(p)$ has changed for a new name when the second transition has been fired.

Then, $\mathcal{M}''''(p)(c_p) = 1$, where c_p is the (new) colour of the token in $c(p)$, as in the markings in $\mathcal{M}_2^*$.

- If $(p,t) \in R$ and $(t,p) \notin F$ then, as in the previous case, the name of the token in $c(p)$ has changed for a new name when the second transition has been fired. This time in the third transition no token has been added to p. Then, $\mathcal{M}''''(p)(c_p) = 0$, where c_p is the (new) colour of the token in $c(p)$, as in the markings in $\mathcal{M}_2^*$.
- If $(p,t) \notin R$ and $(p,t) \in F$ and $(t,p) \notin F$) then $\mathcal{M}''''(p)(c_p) = \mathcal{M}_1^*(p)(c_p) - 1$, where c_p is the colour of the token in $c(p)$, as in the markings in $\mathcal{M}_2^*$.
- If $(p,t) \notin R$ and $(p,t) \notin F$ and $(t,p) \in F$) then $\mathcal{M}''''(p)(c_p) = \mathcal{M}_1^*(p)(c_p) + 1$, where c_p is the colour of the token in $c(p)$, as in the markings in $\mathcal{M}_2^*$.
- Otherwise $\mathcal{M}''''(p)(c_p) = \mathcal{M}_1'(p)(c_p')$, where c_p and c_p' are the colours in $c(p)$ at the markings $\mathcal{M}''''$ and $\mathcal{M}_1'$, as in the markings in $\mathcal{M}_2^*$.

The new reached marking $\mathcal{M}''''$ perhaps has more tokens of different colours than the tokens that we enumerated before, but these tokens are in the places p and their colours are different to the colour of the token c_p, so these tokens do not affect the behavior of the net.

Then, we have proved that if $\mathcal{M}_1 \xrightarrow{t}_{\mathcal{N}} \mathcal{M}_2$, then for all marking $\mathcal{M}_1' \in \mathcal{M}_1^*$, there exists $\mathcal{M}_2' \in \mathcal{M}_2^*$ such that $\mathcal{M}_1' \xrightarrow{t_1t_2t_3}_{\mathcal{F}(\mathcal{N})} \mathcal{M}_2'$.

- Now, suppose that $\mathcal{M}_1' \xrightarrow{t_1t_2t_3}_{\mathcal{F}(\mathcal{N})} \mathcal{M}_2'$, where $\mathcal{M}_1' \in \mathcal{M}_1^*$ and $\mathcal{M}_2' \in \mathcal{M}_2^*$. If t_1 is enabled at $\mathcal{M}_1'$ then for every p such that $(p,t) \in F$, $\mathcal{M}_1'(p)(c_p) \geq 1$, where c_p is the colour of the token in $c(p)$. As $\mathcal{M}_1' \in \mathcal{M}_1^*$, for every p such that $(p,t) \in F$, $\mathcal{M}_1(p) \geq 1$, so t is enabled at $\mathcal{M}_1$.

  Now we know that we can fire t from $\mathcal{M}_1$. Suppose that $\mathcal{M}_1 \xrightarrow{t} \mathcal{M}$. Is $\mathcal{M} = \mathcal{M}_2$? As $\mathcal{M}_2' \in \mathcal{M}_2^*$, we have to see if for every $p \in P$, $\mathcal{M}(p) = \mathcal{M}_2'(p)(c_p)$, where c_p is the colour of the token in $c(p)$ at the marking $\mathcal{M}_2'$. We consider different cases, depending on how t affects the different places:

  - If $(p,t) \in R$ and $(t,p) \in F$ then $\mathcal{M}_2'(p)(c_p) = 1 = \mathcal{M}(p)$, where c_p is the colour of the token in $c(p)$ at the marking $\mathcal{M}_2'$.
  - If $(p,t) \in R$ and $(t,p) \notin F$ then $\mathcal{M}_2'(p)(c_p) = 0 = \mathcal{M}(p)$, where c_p is the colour of the token in $c(p)$ at the marking $\mathcal{M}_2'$.

- If $(p,t) \notin R$ and $(p,t) \in F$ and $(t,p) \notin F$ then $\mathcal{M}'_2(p)(c_p) = \mathcal{M}'_1(p)(c'_p) - 1 = \mathcal{M}_1(p) - 1 = \mathcal{M}_2(p)$, where c_p and c'_p are the colours of the token in the place $c(p)$ at the markings $\mathcal{M}'_2$ and $\mathcal{M}'_1$ respectively.
- If $(p,t) \notin R$ and $(p,t) \notin F$ and $(t,p) \in F$ then $\mathcal{M}'_2(p)(c_p) = \mathcal{M}'_1(p)(c'_p) + 1 = \mathcal{M}_1(p) + 1 = \mathcal{M}_2(p)$, where c_p and c'_p are the colours of the token in the place $c(p)$ at the markings $\mathcal{M}'_2$ and $\mathcal{M}'_1$ respectively.
- Otherwise $\mathcal{M}'_2(p)(c_p) = \mathcal{M}'_1(p)(c'_p) = \mathcal{M}_1(p) = \mathcal{M}_2(p)$, where c_p and c'_p are the colours of the token in the place $c(p)$ at the markings $\mathcal{M}'_2$ and $\mathcal{M}'_1$ respectively.

As for every $p \in P$, $\mathcal{M}(p) = \mathcal{M}'_2(p)(c_p)$, where c_p is the colour of the token in $c(p)$ at the marking $\mathcal{M}'_2$ then $\mathcal{M}_1 \xrightarrow{t}_{\mathcal{N}} \mathcal{M}_2$.

□

Given a reset net $\mathcal{N}$, and its corresponding ν-Petri net $\mathcal{F}(\mathcal{N})$, if the place $s1$ of $\mathcal{F}(\mathcal{N})$ is marked at a marking $\mathcal{M}$, then for all $t \in T$, $s2_t$ and $s3_t$ are not marked. Therefore, at $\mathcal{M}$ the only enabled transitions are that of the kind t_1, for a transition t, reaching a state in which $s2_t$ is marked, but $s1$, $s2_{t'}$, $s3_t$ and $s2_{t'}$ are not marked, for every $t' \in T$. Because of similar reasons, when a transition of the kind t_1 is fired, the two transitions that must be fired next are t_2 and t_3, reaching a marking in which the place $s1$ of $\mathcal{FN}$ is marked again. Therefore, all the transition sequences of $\mathcal{F}(\mathcal{N})$ are of the form $t^1_1, t^1_2, t^1_3, t^2_1, t^2_2, t^2_3, t^3_1...$, for $t^i_j \in P'$. Therefore, we have the following corollary:

Corollary 11 *Given a reset net $\mathcal{N}$ and the corresponding ν-Petri net $\mathcal{F}(\mathcal{N})$, given two markings of $\mathcal{N}$, $\mathcal{M}_1$ and $\mathcal{M}_2$, then*

- *If $\mathcal{M}_1 \rightarrow_{\mathcal{N}} \mathcal{M}_2$ then, for all $\mathcal{M}'_1 \in \mathcal{M}^*_1$ there exists a marking $\mathcal{M}'_2 \in \mathcal{M}^*_2$ such that $\mathcal{M}'_1 \rightarrow_{\mathcal{F}(\mathcal{N})} \mathcal{M}'_2$*
- *If there exist $\mathcal{M}'_1 \in \mathcal{M}^*_1$ and $\mathcal{M}'_2 \in \mathcal{M}^*_2$ such that $\mathcal{M}'_1 \rightarrow_{\mathcal{F}(\mathcal{N})} \mathcal{M}'_2$, then $\mathcal{M}_1 \rightarrow_{\mathcal{N}} \mathcal{M}_2$.*

Finally, we apply the last corollary to prove the undecidability of the repeated coverability problem for ν-Petri nets.

Corollary 12 *Repeated coverability is undecidable for ν-Petri nets.*

Proof: Suppose that we could decide the repeated coverability problem for ν-Petri nets. Suppose that we wanted to solve the repeated coverability problem for a reset net $\mathcal{N}$ and a marking $\mathcal{M}$. Then, to solve this problem it would be enough to solve the repeated coverability problem for the corresponding ν-Petri net $\mathcal{F}(\mathcal{N})$ and the marking $\mathcal{M}'$, which is the marking in $\mathcal{M}^*$ without any trash, that is, the marking in $\mathcal{M}^*$ in which the places p are marked with tokens of one single colour. Repeated coverability is undecidable for reset Petri nets so, so we have found a contradiction. Therefore, repeated coverability is undecidable for ν-Petri nets.

□

In this chapter we have summarized all the important definitions and results we need to continue with this work: We know what a Petri net is, and what are the extensions we consider in the following chapters: mainly reset nets and ν-Petri nets. We also know some of the important decidability results we use later. Now we are ready to face the main content of this project: the issues related to decidability (or undecidability) of model checking for Petri nets and its extensions.

Chapter 3

Model checking Petri nets

In this chapter address the main purpose of this project: model checking Petri nets. We start by explaining what is model checking and some basics about it.

3.1 Basics of model checking

When we specify systems, we would like to know if the specification we are giving satisfies certain properties. Model checking is a collection of automatic techniques to verify properties of a system specification. Clarke and Emerson introduced model checking in 1981 [11, 21]. Since then, a great amount of work about model checking has been done, not only for theoretical purposes, but also for verifying industrial systems [8, 13].

Standard model checking algorithms are based in an exhaustive visit to all the reachable states of the specification we want to check. Therefore, these techniques are useful to check finite state systems, but not infinite ones. That is why the boundedness problem of Petri nets is important to model checking issues about them, because if the net is bounded then standard model checking techniques can be used to check it.

If we want to decide if some properties hold or not at a system specification, we have to be able to express these properties on a formal logic. The logics that are usually used in model checking are temporal logics, which let us express properties like "this property holds in a reachable state", "there is a path in which this property always holds" or "this property holds in every reachable state". Now we are going to give a short introduction to temporal logics.

Temporal logics consider a set of atomic propositions (which express atomic

properties) and several temporal operators and path quantifiers, which allow us to express temporal properties. We consider two types of formulas: state formulas, which are true or false in a specific state; and path formulas, which are true or false along a specific path, starting at a specific states. Given a system specification $\mathcal{T}$, if a state $\mathcal{S}$ of this system satisfies a state formula φ, then we denote $\mathcal{T}, \mathcal{S} \vDash \varphi$. If a computation path π satisfies a path formula, then we write $\mathcal{T}, \pi \vDash \varphi$.

Some of the main temporal operators used in temporal logics are **X**, **G**, **F** and **U**. These operators describe properties which may hold or not in a path of states. We are going to explain the semantics of these temporal operators informally, before doing it formally:

Let φ be a temporal logic formula, and $\mathcal{T}$ a transition system and a computation path π with an initial state $\mathcal{S}$. Then:

- $\mathcal{T}, \pi \vDash \mathbf{X}\varphi$ (next) holds if the property φ holds in the state that follows $\mathcal{S}$.
- $\mathcal{T}, \pi \vDash \mathbf{G}\varphi$ (globally) holds if the property φ holds in every state of π.
- $\mathcal{T}, \pi \vDash \mathbf{F}\varphi$ (eventually) holds if the property φ holds in some state of π.
- $\mathcal{T}, \pi \vDash \varphi \mathbf{U} \psi$ (until) holds if there is a state of the path π such that ψ holds in that state, and at every preceding state on the path φ holds.

To define the previous operators we have considered a path of states, but not the possibly different paths of reachable states from an initial state. The path quantifiers are the operators which consider them. We explain informally two of these operators:

- $\mathcal{T}, \mathcal{S} \vDash \mathbf{E}\varphi$ if there is a path starting at $\mathcal{S}$ in which φ holds.
- $\mathcal{T}, \mathcal{S} \vDash \mathbf{A}\varphi$ if for all paths starting at $\mathcal{S}$, φ holds.

In the next definition, π^i denotes the suffix of the path π starting by the i-th state s_i. We suppose that the states of a system are labelled by the atomic propositions that they satisfy, that is, if a state s satisfies an atomic proposition p, then $p \in L(s)$, where $L(s)$ represents the labels of s. Let f_1 and f_2 be two state formulas, and g_1 and g_2 two path formulas. The formal definition of the previous operators and path quantifiers is inductively defined as [14]:

- $\mathcal{T}, s \vDash p \Leftrightarrow p \in L(s)$.
- $\mathcal{T}, s \vDash \neg f_1 \Leftrightarrow \mathcal{T}, s \nvDash f_1$.

- $\mathcal{T}, s \vDash f_1 \vee f_2 \Leftrightarrow \mathcal{T}, s \vDash f_1$ or $\mathcal{T}, s \vDash f_2$.
- $\mathcal{T}, s \vDash f_1 \wedge f_2 \Leftrightarrow \mathcal{T}, s \vDash f_1$ and $\mathcal{T}, s \vDash f_2$.
- $\mathcal{T}, s \vDash \mathbf{E}g_1 \Leftrightarrow$ there is a path π from s such that $\mathcal{T}, \pi \vDash g_1$.
- $\mathcal{T}, s \vDash \mathbf{A}g_1 \Leftrightarrow$ for every path π starting from s, $\mathcal{T}, \pi \vDash g_1$.
- $\mathcal{T}, \pi \vDash f_1 \Leftrightarrow s$ is the first state of π and $\mathcal{T}, s \vDash f_1$.
- $\mathcal{T}, \pi \vDash \neg g_1 \Leftrightarrow \mathcal{T}, \pi \nvDash g_1$.
- $\mathcal{T}, \pi \vDash g_1 \vee g_2 \Leftrightarrow \mathcal{T}, \pi \vDash g_1$ or $\mathcal{T}, \pi \vDash g_2$.
- $\mathcal{T}, \pi \vDash g_1 \wedge g_2 \Leftrightarrow \mathcal{T}, \pi \vDash g_1$ and $\mathcal{T}, \pi \vDash g_2$.
- $\mathcal{T}, \pi \vDash \mathbf{X}g_1 \Leftrightarrow \mathcal{T}, \pi^1 \vDash g_1$.
- $\mathcal{T}, \pi \vDash \mathbf{F}g_1 \Leftrightarrow$ there exists a $k \geq 0$ such that $\mathcal{T}, \pi^k \vDash g_1$.
- $\mathcal{T}, \pi \vDash \mathbf{G}g_1 \Leftrightarrow$ for all $i \geq 0$, $\mathcal{T}, \pi^i \vDash g_1$.
- $\mathcal{T}, \pi \vDash g_1 \mathbf{U} g_2 \Leftrightarrow$ there exists a $k \geq 0$ such that $\mathcal{T}, \pi^k \vDash g_2$ and for all $0 \leq j < k, \mathcal{T}, \pi^j \vDash g_1$.

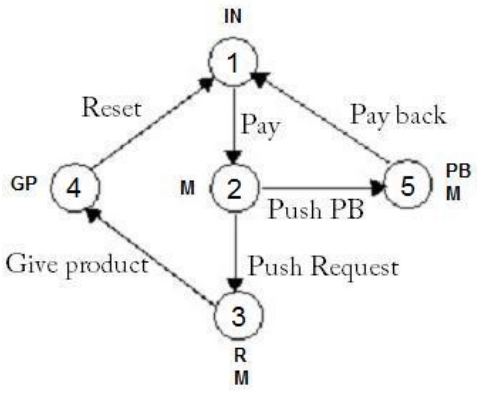

Figure 3.1: An automata modelling a vending machine

Let us give an example of the use of the previous operators, to clarify their meanings:

Example 14 *Let us consider the automaton in figure 3.1 which represents a very simplified model of a vending machine. There are some properties which can hold or not in the states of the automaton:*

- *I: The machine is in an initial state, where nobody has introduced any coin and nobody has pressed any button.*
- *M: In the machine there is a (proper) amount of money to get a product.*
- *PB: The user has pressed the button to request to be given his money back.*
- *R: The user has pressed the button to request a product.*
- *GP: The machine has given a product to the user.*

The previous properties are the atomic propositions we are going to consider. Now, we would like to express several temporal properties:

- *"If there is money in the machine, and the user has not pressed the button "request", then the user has the possibility to ask for his money back":* $M \wedge \neg R \Rightarrow \mathbf{EF} PB$.
- *"If the user has introduced the proper money in the machine, and pushed the button for requesting a product, then he will be given a product in the future":* $M \wedge R \Rightarrow \mathbf{F} GP$.
- *"If the machine has given a product, then the next state has to be the initial state":* $GP \Rightarrow \mathbf{X} IN$.

Using different combinations of the previously defined operators, we can build different temporal logics. There are mainly two kinds of logics: linear time logics and branching time logics. The difference between these two kinds of logics is that the properties that are expressed in branching time logics are properties about the different paths that are possible from an initial state [5], and in linear time logics the properties that are expressed are properties about a single path [69, 70, 57, 58]. Though linear time logics express properties about a single path, we say that a system satisfies a formula of a linear time logic iff there is a running (a path) of the systems which satisfies the formula. In other works it is considered that

a system satisfies a formula of a linear time logic iff all runnings of the system satisfy the property, but we have considered the other notion because most of model checking Petri nets results in the literature are expressed this way. Let us describe several of the most temporal important logics:

The most representative example of a linear lime logic is LTL. An LTL formula is a formula of the form $\mathbf{A}\phi$ where ϕ is a path formula. A path formula of LTL is an atomic proposition or a formula of the form $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\mathbf{X}\varphi$, $\mathbf{F}\varphi$, $\mathbf{G}\varphi$ or $\varphi\mathbf{U}\psi$, where φ and ψ are path formulas.

The most representative example of a branching time logic is CTL. In CTL each temporal operator must be immediately preceded by a path quantifier, so the combinations of operators used in CTL are: **AX**, **EX**, **AF**, **EF**, **AG**, **EG**, **AU** and **EU**. All these combinations of operators are used with $\neg$, $\vee$ and $\wedge$ to form CTL formulas.

CTL and LTL can express different properties, that is, they have different expressive power [12]. CTL* is a logic which contains both LTL and CTL, so it has more expressive power than them. It was defined in order to have a frame in which study properties of the two logics. Therefore model checking techniques have been applied to LTL and CTL more often than to CTL*. There is another important logic which contains CTL, LTL and CTL*: the modal μcalculus, which is not defined in terms of the operators explained before. Modal μ-calculus is defined in terms of a fix point operator. For detail see [50].

We have explained the temporal operators and path quantifiers, but what are the atomic propositions we consider for model checking Petri nets? We use three atomic propositions (which we will call atomic predicates):

- $cov(\mathcal{M})$, where $\mathcal{M}$ is a marking of the considered net. The fact that $cov(\mathcal{M})$ holds at a marking $\mathcal{M}'$ of a PN, means that $\mathcal{M}'$ covers $\mathcal{M}$. Sometimes the equivalent operator $ge(p, n)$, where $p \in P$ and $n \in \mathbb{N}$, is considered instead of $cov(\mathcal{M})$. $ge(p, n)$ holds at a marking $\mathcal{M}$ if $\mathcal{M}(p) \geq n$. In this work we consider cov for simplicity.

- $first(t)$, where t is a transition of the considered net. A path satisfies $first(t)$ (for a maximal path), if the first fired transition in the path is t.

- $en(t)$, where t is a transition of the considered net. A marking $\mathcal{M}$ satisfies $en(t)$ if t is enabled at $\mathcal{M}$.

Let us show an example to illustrate what the previous atomic predicates mean:

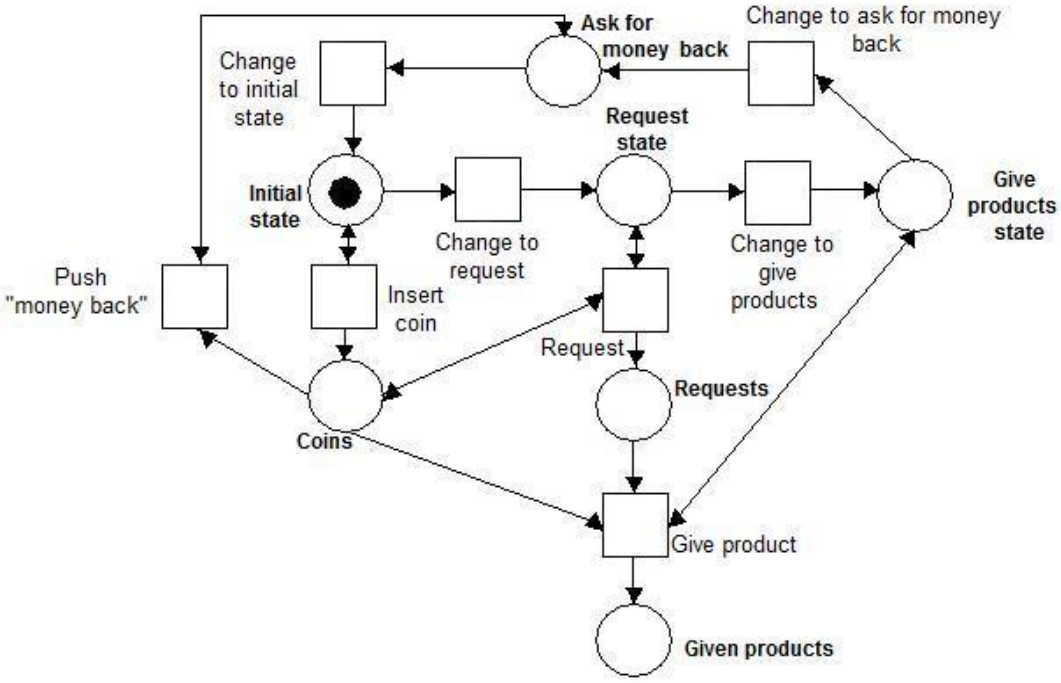

Figure 3.2: A P/T net modelling a vending machine

Example 15 *Figure 3.2 shows a P/T which represents a vending machine. In the initial marking the client can insert as many coins as he wants. When the client inserts a coin, the transition "Insert coin" is fired, and a token is added to the place "Coins", which represent the amount of coins inside the machine.*

Then, when the transition "Change to request" is fired, the client can request as many products as coins he has inserted. When the client pushes the proper button, the transition request is fired, and a token is added to the place "Requests".

When the transition "Change to give products" is fired, then the vending machine can give as many products as the client has requested to the client, by firing the transition "Give product". Then, a token is removed from "Coins" and from "Requests".

Finally, the client can ask for his money back when there is a token in "Ask for money back". If the client asks for money back, then a token is removed from "Coins". The machine gives one coin back each time the client push "Money back".

Let us use this P/T to show what the previous atomic predicates mean. We consider formulas of a linear time logic which are interpreted over paths of occurrences, so we consider a path on the P/T. Some of the properties we can express with the defined atomic predicates are:

- *If the first fired transition in the considered path is "Change to request" and there are coins in the machine, then in the next marking the client requests a product:*

  $first(Change\ to\ request) \wedge cov(C) \Rightarrow \boldsymbol{X}\ en(Request)$, *where* C *is the marking such that* $C(Coins) = 1$ *and* $C(p) = 0$, *for every place* $p \neq Coins$.

- *If there is a coin in the machine in the first marking of the path, then sometime in the future the machine will give a product:*

  $cov(C) \Rightarrow \boldsymbol{F}\ first(Give\ product)$, *where* C *is as in the previous case.*

Model checking finite systems is clearly decidable, but the algorithms for model checking linear time logics and branching time logics on finite systems suffer from the so called state explosion problem: the number of states that need to be checked grows exponentially on the size of the system. To alleviate this problem there are techniques such as symbolic model checking [62] and abstractions which decrease the state space needed to check systems. Another important technique to this purpose in the setting of concurrent systems is unfolding. Unfolding was first introduced by McMillan [61, 62]. An unfolding is a compact representation of the different computations of a system(interleavings). With it, we can obtain model checking results needing less state space. Unfolding techniques have been well studied in literature [24, 62].

Most model checking techniques consists on visiting all (or perhaps some) states and verifying some required properties. That is why these techniques are useful for finite systems, in which you have to visit a finite number of states, but not enough for infinite systems such as Petri nets and its extensions. In fact, model checking often becomes undecidable for infinite state systems.

In the following sections we summarize the facts we know until today about model checking Petri nets and its extensions, and we give a few more results.

3.2 Model checking Place/Transition nets

In this section we summarize some of the classic decidability (or undecidability) results in model checking place/transition Nets.

3.2.1 Branching time logics

First, we focus on branching time logics, in particular we consider **EF** (also called UB^-), which is less expressive than the most used branching logics CTL and CTL^*.

EF is a branching logic which have as basic predicates, predicates of the form $cov(\mathcal{M})$. **EF** also includes the boolean connectives $\neg$, $\wedge$, $\vee$ and two operators: $\mathbf{EX}\varphi$, which means that given a marking, there exists an enabled transition such that if we fire it, we reach a marking that satisfies the propertie φ; and $\mathbf{EF}\varphi$, which means that, given an initial marking $\mathcal{M}$, there exists an enabled path starting at $\mathcal{M}$ that reaches a marking that satisfies φ.

Example 16 *Let us show some of the properties you can express with the branching logic **EF** about the P/T of figure 3.1:*

- *If there is one or more coins in the machine then, a product is eventually given:* $cov(C) \Rightarrow \boldsymbol{EF}\, cov(G)$*, where* C *is the marking such that* $C(Coins) = 1$ *and* $C(p) = 0$*, for every place* $p \neq Coins$ *and* G *is the marking such that* $G(Given\ products) = 1$ *and* $G(p) = 0$*, for every place* $p \neq Given\ products$*..*
- *You can eventually reach a marking with one token in the place "Initial state" from the current marking:* $\boldsymbol{EF}\, cov(I)$*, where* I *is the marking such that* $I(Initial\ marking) = 1$ *and* $I(p) = 0$*, for every place* $p \neq Initial\ state$*.*
- *If there is a token in the place "Initial state", then any transition reaches a marking in which there is a token in the place "Give products state":* $cov(I) \Rightarrow \boldsymbol{EX}\, cov(GS)$*, where* GS *is the marking such that* $GS(Give\ products\ state) = 1$ *and* $GS(p) = 0$*, for every place* $p \neq Give\ products\ state$*.*

Esparza proved in [23] that model checking **EF** in place/transition nets is undecidable by reducing the containment problem to model checking a formula of **EF**: Given two P/Ts $\mathcal{N}_1$ and $\mathcal{N}_2$ which are an instance of the containment problem with a bijection f, Esparza builds a new P/T net that combines both P/Ts, and adds some places and transitions in order to reduce the containment problem to the model checking problem of a formula of **EF** on the new P/T.

UB, CTL and CTL* are more expressive than **EF**, so model checking UB, CTL and CTL* in place/transition nets is undecidable too. That means that model checking the most common branching logics is undecidable for Petri nets.

3.2.2 Linear time logics

Now we consider linear time logics. Let us remember that these logics are interpreted on the markings of the maximal paths of a Petri net, so a Petri net satisfies a formula if all its maximal paths satisfy it. We add the other two basic predicates defined in the previous section: $first(t)$ and $en(t)$, and we consider different linear time logics, built depending on which predicates and operators we consider.

We focus on four logics: linear time μ-calculus with only $first(t)$ as basic predicate, linear time μ-calculus with $first(t)$ and $cov(\mathcal{M})$ as basic predicates, the fragment in which negation is only applied to basic predicates (not to operators), and the operators are **F**, $\wedge$ and $\vee$, and the fragment in which the only allowed composed operator is **GF**, the operators are **F**, $\vee$ and $\wedge$m and negation is only applied to basic predicates. Let us summarize some of the results about model checking the lineal time logics built as we explained before:

- Esparza proved that model checking Linear time μ-calculus with only $first(t)$ as basic predicate is decidable for P/Ts. Here is a sketch of the proof that Esparza gives, you can see the complete demonstration in [23].

  Esparza consider a construction of Dam [16] in which, given a formula ϕ of a different version of the linear time μ calculus, an automaton such that its accepting language is the language of all the words in which ϕ holds is built. This construction is adapted to the version of the linear time μ calculus which Esparza consider. Next, given an automaton, Esparza shows how to build a P/T which accepts the same language as the automaton. So given a formula ϕ, the corresponding P/T can be built. The P/T corresponding to the negation of the formula we want to check is built. In fact, two P/Ts are built in the construction: one automaton $\mathcal{A}$ which admits the finite words of the language and another $\mathcal{B}$ which admits the infinite words. So now we have our original P/T to be tested $\mathcal{N}$ and the two P/Ts $\mathcal{A}$ and $\mathcal{B}$, which admits the words in which the negation of the property we want to check

holds. So if $\mathcal{L}(\mathcal{N}) \cap \mathcal{L}(\mathcal{A}) = \emptyset$ and $\mathcal{L}(\mathcal{N}) \cap \mathcal{L}(\mathcal{B}) = \emptyset$, then the property holds in $\mathcal{N}$.

Then, it is defined a product "$\times$" between two place/transition nets, similar to the product of automata. Finally, Esparza shows that deciding if $\mathcal{L}(\mathcal{N}) \cap \mathcal{L}(\mathcal{A}) = \emptyset$ and $\mathcal{L}(\mathcal{N}) \cap \mathcal{L}(\mathcal{B}) = \emptyset$ is equivalent to deciding if the products $\mathcal{N} \times \mathcal{A}$ and $\mathcal{N} \times \mathcal{B}$ satisfy some decidable properties:

- In particular, $\mathcal{L}(\mathcal{N}) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$ holds iff there exists a reachable dead marking of $\mathcal{N} \times \mathcal{A}$ which puts a token in a final state of $\mathcal{A}$. This property is decidable, because it can be decided by solving an exponential number of instances of the submarking reachability problem, which is decidable [40].

- Similarly, $\mathcal{L}(\mathcal{N}) \cap \mathcal{L}(\mathcal{B}) \neq \emptyset$ holds iff $\mathcal{N} \times \mathcal{B}$ has a path of occurrences which contains infinite occurrences of a transition in a special set of final transitions (see [23] for detail). This condition is decidable too [77].

The last construction solves the problem of deciding if a property expressed in Linear time μ-calculus with only $first(t)$ holds or not in a P/T, so the problem is decidable. It has been shown that the same logic adding the basic predicate $cov(\mathcal{M})$ becomes undecidable.

- Model checking the fragment in which negation is only applied to basic predicates (not to operators), and the operators are **F**, **X**, $\wedge$ and $\vee$ is reduced in polynomial time to the reachability problem. As we said before, the reachability problem is decidable for P/Ts, so model checking this fragment is decidable (see [44]).

  Example 17 *In this fragment we can express properties like "Sometime in the future the machine will give a product", which is* $\boldsymbol{F}\ first(Give\ product)$*, but we cannot express properties like "The client has always the possibility to insert coins", which is expressed as* $\boldsymbol{G}\ en(Insert\ coin)$*; or "The machine will give a product to the client infinitely often":* $\boldsymbol{GF}\ first(Give\ product)$*. That is because, to obtain the operator* $\boldsymbol{G}$ *in terms of* $\boldsymbol{F}$*, negation must be applied to* $\boldsymbol{F}$*.*

- Model checking the fragment in which the only allowed composed operator is **GF**, the operators are **F**, $\vee$ and $\wedge$ and negation is only applied to basic

predicates, can be reduced to an exponential number of instances of the reachability problem, so it is decidable (see [45]).

Example 18 *With this fragment you can express "The machine will give a product to the client infinitely often":* $\boldsymbol{GF}\ first(Give\ product)$*, but not "If the client inserts a coin infinitely often, then the machine will give a product sometime in the future":* $\boldsymbol{GF}\ first(Give\ product) \Rightarrow \boldsymbol{F}\ first(Give\ product)$*. That is because if we replace* $\Rightarrow$ *by its definition, then a negation is applied to the operator* $\boldsymbol{GF}$*.*

- In 1992 Yen gave a logic [80] (which later has been called Yen's path logic) for place/transition nets in order to obtain a uniform approach for deciding and studying the complexity of many Petri nets problems. This logic consists on path formulae of the form:

$$\exists \mathcal{M}_1, ..., \mathcal{M}_n \exists \sigma_1, ..., \sigma_n((\mathcal{M}_0 \stackrel{\sigma_1}{\rightarrow} \mathcal{M}_1 \stackrel{\sigma_2}{\rightarrow} ... \stackrel{\sigma_{n-1}}{\rightarrow} \mathcal{M}_{n-1} \stackrel{\sigma_n}{\rightarrow} \mathcal{M}_n) \wedge \phi(\mathcal{M}_1, ..., \mathcal{M}_n, \sigma_1, ..., \sigma_n)).$$

  that means that any marking $\mathcal{M}_i$ $(i \in \mathbb{N})$ is reachable from $\mathcal{M}_{i-1}$ by firing the sequence of transitions σ_i, and that the path formula $\phi(\mathcal{M}_1, ..., \mathcal{M}_n, \sigma_1, ..., \sigma_n)$ holds. With more detail, a path formula consists on [80]:

  - Variables of two types: marking variables $\mathcal{M}_1, ..., \mathcal{M}_m ...$ and transition sequences variables $\sigma_1, ..., \sigma_n$.
  - Recursively defined terms:
    - $\forall c \in \mathbb{N}^k$, where k is the number of places of the considered net, c is a term.
    - $\forall j > i$, $\mathcal{M}_j - \mathcal{M}_i$ is a term.
    - If T_1 and T_2 are terms, then $T_1 + T_2$ and $T_1 - T_2$ are terms too.
  - Atomic predicates. In [80] Yen defined two kinds of atomic predicates: transition and marking predicates, but he proved that to consider only marking predicates was enough to obtain the same expressive power that is obtained by considering the two kinds of atomic predicates. Therefore, we consider only marking predicates, which are of two types:
    - If $\mathcal{M}$ is a marking and $c \in \mathbb{N}$ is a constant, then $\mathcal{M}(i) \geq c$ and $\mathcal{M}(i) \geq c$ are predicates.

* If T_1 and T_2 are terms and $1 \leq i, j \leq k$, where k is the number of states of the considered net, then $T_1(i) = T_2(j)$ and $T_1(i) < T_2(j)$, $T_1(i) < T_2(j)$ are predicates, where $T_1(i)$ represents the i-th component of T_1 and $T_2(j)$ represents the j-th component of T_2.

If ϕ_1 and ϕ_2 are predicates then $\phi_1 \vee \phi_2$ and $\phi_1 \wedge \phi_2$ are predicates.

Let us show some examples of what kind of properties you can express with this logic:

Example 19 *Let us focus on the P/T system of figure 3.2, which represents a vending machine. With this logic we can express properties such as:*

- $\exists \mathcal{M} \exists \sigma((\mathcal{M}_0 \xrightarrow{\sigma} \mathcal{M}) \wedge (\mathcal{M}(6) \geq 1)$*, where the sixth place is "Given products", and* M_0 *is the initial marking. That formula expresses that there exists a computation in which the machine gives one or more products. This formula expresses the same as* ***EF****cov(G). In fact, every instance of the coverability problem can be expressed in terms of formulae of this logic.*
- $\exists \mathcal{M}_1, \mathcal{M}_2 \exists \sigma_1, \sigma_2((\mathcal{M}_0 \xrightarrow{\sigma_1} \mathcal{M}_1 \xrightarrow{\sigma_2} \mathcal{M}_2) \wedge (M_2(4) > M_1(4))$*, where the fourth place is "Coins", and* M_0 *is the initial marking. That formula express that there exists a computation in which the number of coins that the machine contains grows infinitely. Actually, the boundedness problem can be expressed in terms of formulae of this logic too.*

In [4] M. Faouzi Atig and P. Habermehl show that Yen's path logic can be reduced to the reachability problem, and the reachability problem can be reduced to model checking Yen's path logic. The reachability problem is decidable for P/Ts, so model checking Yen's path logic is decidable too.

3.3 Towards model-checking Petri net extensions

As we said before, if we have a formalism we would like to be able to check some properties about the systems we specify using it, that is, model checking these systems. In the previous chapter we explained some Petri net extensions which are defined in the literature in order to increment the expressive power of P/T nets and be able to model more systems. Now we would like to know whether we can keep some of the decidability results we saw in the previous section. In this

section we are going to compile some of the existing results about model checking the Petri net extensions we are considering in this work, and we will also give some new results. We are going to focus on two Petri net extensions: reset nets and ν-Petri nets.

As the extensions we are going to consider can simulate a place-transition net, the undecidability results that we have for P/Ts are inherited by reset and ν-Petri nets. Let us summarize the undecidability properties that we explained in the previous section in order to keep in mind that for reset and ν-PN these properties hold too:

- UB is undecidable for P/T nets, so CTL and CTL* are undecidable too.
- The fragments of UB, **EF** and **EG** are undecidable.
- μ-calculus with the operators $first$ and cov is undecidable.

In the last section we explained some (positive) decidability results for P/T nets. To prove them, the results were mostly reduced to instances of the reachability problem. For example, model checking the fragment in which negation is only applied to basic predicates, and the operators are **F**, $\wedge$ and $\vee$ is reduced to the reachability problem, and model checking the fragment in which the only allowed composed operator is **GF**, the operators are **F**, $\vee$ and $\wedge$ and negation is only applied to basic predicates, can be reduced to an exponential number of instances of the reachability problem. As the reachability problem is decidable for P/T nets, these problems are decidable for them too. However, as we have already pointed out reachability is not decidable for reset and ν-PN. Therefore we cannot adapt the proofs for model checking P/T nets to the extensions we consider. In fact, we will see that all of the decidable model checking problems for P/T that we are going to study for P/T extensions, are not decidable for the considered extensions.

In order to proof decidability issues about model checking Petri net one has to bear in mind the decidability results that there are in the literature, in order to reduce model checking problems to other solved problems. Let us summarize the results we explained in the previous chapter:

- Reachability is undecidable for reset and ν-Petri nets.
- Coverability and termination are decidable for both extensions.

- Repeated coverability is undecidable for resets and ν-Petri nets.
- Boundedness is decidable for ν-Petri nets but undecidable for reset Petri nets.

Some of the previous properties can be expressed in some of the temporal logics that we are going to study, so for the undecidable properties, model checking the corresponding logic will be undecidable. Let us explain some results about model checking the Petri nets extensions we are considering: reset- Petri nets and ν-Petri nets.

3.3.1 Model checking reset Petri nets

Before we start studying the decidability or undecidability of model checking reset nets, let us show a toy example to show that in some cases we need to use reset nets in order to model systems instead of P/T.

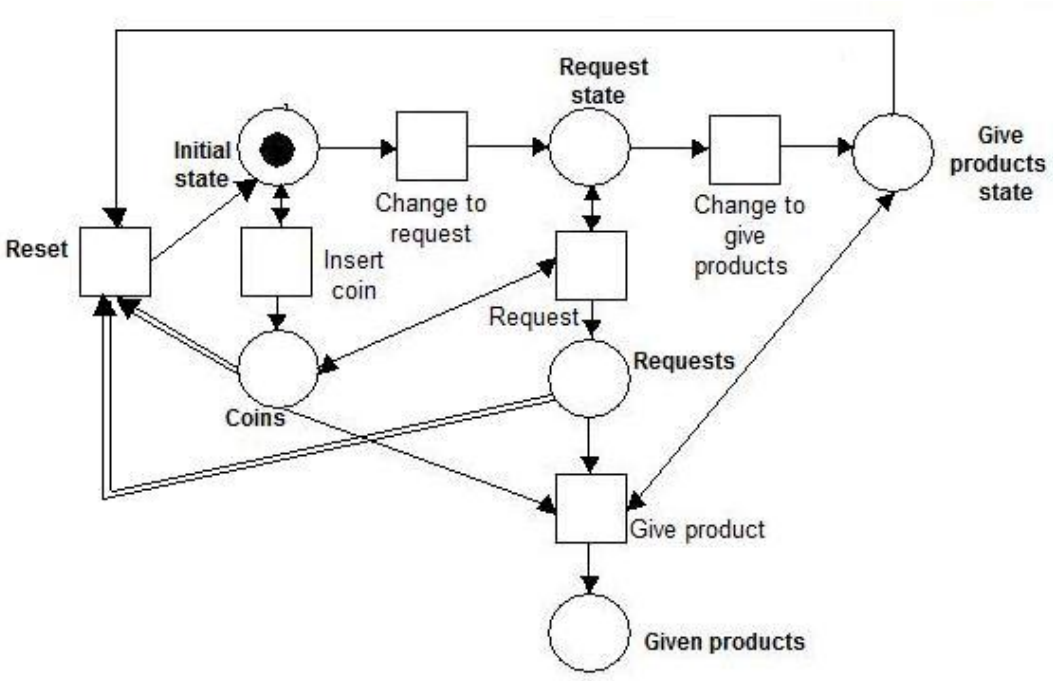

Figure 3.3: A reset net modelling a vending machine

Example 20 *In a previous example we showed a P/T which represented a vending machine. In that machine, if the client wanted to ask for his money back, then he had to press the button "money back" one time for each coin he had inserted.*

That is because in P/T nets we cannot atomically empty a place, as we explained in the previous chapter. If we model a vending machine, we would like to express that when the client pushes "money back" the machine gives him all his money back. With reset nets we can model it.

Figure 3.3 shows a reset net which represents a vending machine. The behavior of this vending machine is similar to the behavior of the machine in the previous example, except for the fact that when this machine changes from the "give products" state to the "initial" state, it gives the client all his money back without pressing any button, and it cancels all the requests. That actions occurs when the transition "reset" is fired, and the two reset arcs in the net empty the corresponding places.

As in the P/T example, here we would like to check some properties too, as "Every time the machine is in the initial state, does it have no requested products?" or "Does the machine always return to the initial state?". Once again, these questions are expressible in temporal logics, so solving these questions is solving an instance of some model checking problem.

Despite the fact that reset nets have been widely studied in the literature, it is sometimes difficult to find certain model checking results for them because in some cases they are "hidden" in results for other formalisms which are equivalent in some sense to reset nets. More precisely, in order to summarize the model checking issues for reset nets in the literature, we have looked for results about lossy vector addition systems with inhibitory arcs, also called lossy counter machines [60].

We consider two logics which were decidable for P/T nets, and we will see that both fragments are undecidable for reset nets. In particular the considered fragments are μ-calculus with $first$ and the fragment in which the only allowed composed operator is **GF**, the operators are **F**, $\vee$ and $\wedge$ and negation is only applied to basic predicates.

In [7] the model checking problem is studied for lossy vector addition system (lossy VASS). In particular, they consider lossy vector addition systems with inhibitory arcs, which consists on vector addition systems which may lose tokens every time a transition is fired and can test for zero. Intuitively, this formalism is like Petri nets with inhibitory arcs which may lose some tokens when a transition is fired. This formalism was introduced in order to model systems in which part of the information may be lost, as communication systems with unreliable channels.

There is an important issue we need to take into account when we use lossy vector addition systems to study Petri nets. In "lossy" formalisms, that is, in formalisms in which the state can be spontaneously decreased, the coverability and the reachability are put on the same level. If a marking $\mathcal{M}$ is reachable then it is, clearly, coverable. Conversely if it is coverable, then there is a reachable marking $\mathcal{M}'$ such that $\mathcal{M} \subseteq \mathcal{M}'$, and because of lossiness $\mathcal{M}'$ can lose the proper tokens to become $\mathcal{M}$, so that $\mathcal{M}$ is also reachable. In the same line, some decidable problems for lossy vector addition systems are undecidable for P/T nets.

Due to the facts of the previous paragraph, we cannot apply all model checking decidability results of lossy vector addition systems to Petri nets. But, in particular, we can apply lossy VASS with inhibitory arcs undecidability results to reset nets. That is because a reset nets can simulate the behavior of a lossy VASS with inhibitor arcs, because when we are in a lossy system testing for zero is equivalent to reseting a place, so reset nets can simulate tests for zero in this case. More precisely, given a lossy VASS with inhibitory arcs $\mathcal{L}$, there is a reset net $\mathcal{R}$ such that if s_1 and s_2 are states of $\mathcal{L}$ and $s_1 \rightarrow_L s_2$ and $s_1 \rightarrow_R s_2$ denotes that $s_1 \rightarrow s_2$ in $\mathcal{L}$ and $\mathcal{R}$ respectively, then

$$s_1 \rightarrow_L s_2 \Rightarrow \exists {s_2}' \geq s_2 \text{ such that } s_1 \rightarrow_R {s_2}' \text{ and}$$
$$s_1 \rightarrow_R s_2 \Rightarrow s_1 \rightarrow_L s_2.$$

In [7] it is proved that model checking LTL is undecidable for lossy VASS with inhibitory arcs. The proof consists on reducing the problem of deciding if there is an initial configuration such that there is an infinite run of the system starting with it, which is undecidable for lossy VASS with inhibitory arcs, to the LTL model checking problem. As we said before, this undecidability result can be applied to reset nets, so model checking LTL is undecidable for reset nets. Despite the fact that model checking μ-calculus with $first$ as the only operator is decidable for P/T, LTL is less expressive than it, so model checking this logic is undecidable for reset nets.

The fragment in which the only allowed composed operator is **GF**, the operators are **F**, $\vee$ and $\wedge$ and negation is only applied to basic predicates is also undecidable for reset Petri nets. The reason is that we can reduce the repeated coverability problem for reset nets, which is undecidable, to model checking this fragment. In particular, if the considered fragment were decidable then, given a marking $\mathcal{M}$, model checking the formula $\mathbf{GF}cov(\mathcal{M})$ would be enough to decide the repeated coverability problem for $\mathcal{M}$. As repeated coverability is undecidable

for reset nets, the fragment in which the only allowed composed operator is **GF**, the operators are **F**, $\vee$ and $\wedge$ and negation is only applied to basic predicates, is also undecidable for them.

3.3.2 Model checking ν-Petri nets

Let us give a toy example to illustrate why it is important to be able to use ν-Petri nets instead of P/Ts in order to model more kind systems and why would one like to model check ν-Petri nets.

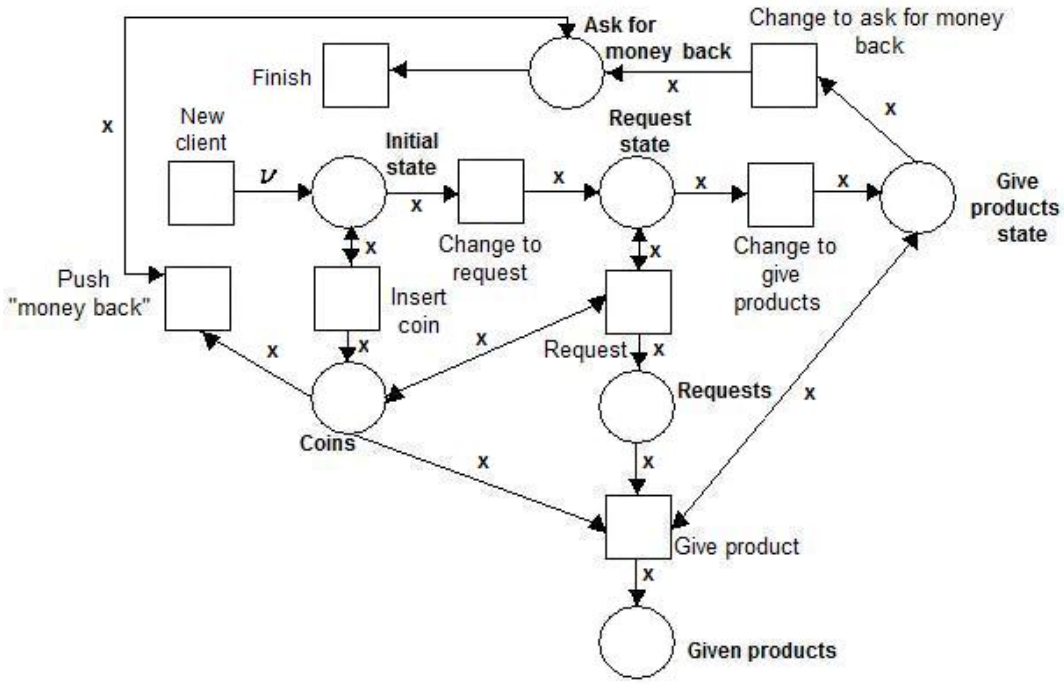

Figure 3.4: A ν-PN modelling a vending machine

Example 21 *The P/T of the example of the previous sections modelled a vending machine. Imagine that now we wish to have an online-vending machine, so that we want to add to the machine the possibility several clients to buy products simultaneously, without a bound on the number of clients. In the last P/T the machine was only able to interact with one client. Now, we are going to build a ν-Petri net which will represent a vending machine which will satisfy the new requirement. Different names will correspond to different clients, so that to represent the money, the requests, the state... of a client tokens with the same name*

will be used. In order to model the admission of a new client, there will be a transition with a ν arc, which creates a new name.

Figure 3.4 shows a ν-Petri net which represents the vending machine we want to model. When a client is admitted the transition "New client" is fired, creating a new name which will represent all the tokens in the net which are referred to this client. When a transition is fired in this net, all added and removed tokens have to be tokens of the same name (of the same client). That is why all arcs are labelled by an x, so different names do not synchronize in this net.

Now that we have the net representing the vending machine we desired, we will probably want to check some properties about the net, as "Does a client reach the final state every time a new client is admitted?" or "Does the machine give all the money back every time a client asks for it?".

There is not much literature about model checking ν-Petri nets. In fact, when one sees the last example, one tends to think that the temporal logics that there exist are not very suitable for ν-Petri nets. In fact, we would like to ask questions related to what happens with all tokens of a name (about the behavior of the machine about the issues related to one client), but the logics we are studying cannot express such questions since they are logics thought for P/T nets, which do not have distinguishable tokens. This could be a future work to do, because in this work we focus on studying model checking for temporal logics that are already defined in the literature.

As we said before, model checking ν-Petri nets has not been studied in the literature. That is why we do not have much results about it. The two results we give here are original from this work: the undecidability of model checking two logics which are μ-calculus with $first$ and the fragment in which the only allowed composed operator is **GF**, the operators are **F**, $\vee$ and $\wedge$ and negation is only applied to basic predicates. In fact, the work we do consists on translating the undecidability results about reset nets to ν-Petri nets.

The fact that μ-calculus with $first$ is undecidable is due to the fact that we can simulate a reset net with a ν-PN net by using the construction of the proof of the undecidability of the repeated coverability of ν-PN that we gave previously. The only problem there could be in the construction is that the trash that is generated (the tokens which we do not consider anymore) could affect the formulas of the logic, but this is not the case, because the logic is built based on the operator $first$, which is related with the transitions that are fired, and not

with the particular tokens in the places. Therefore, if μ-calculus with $first$ were decidable for ν-PN, it would be decidable for reset nets too. As μ-calculus with $first$ is undecidable for reset-Petri nets, it is undecidable too for ν-PN.

Now we are going to prove that model checking the fragment in which the only allowed composed operator is **GF**, the operators are **F**, $\vee$ and $\wedge$ and negation is only applied to basic predicates is undecidable for ν-Petri nets. For that purpose we will reduce the repeated coverability problem (which is undecidable for ν-PNs) to model checking an specific formula of this logic.

Proposition 13 *Model checking the fragment in which the only allowed composed operator is GF, the operators are F, $\vee$ and $\wedge$ and negation is only applied to basic predicates is undecidable for ν-Petri nets*

Proof: Suppose the considered fragment were decidable for ν-Petri nets. Then, given a ν-Petri net $\mathcal{N}$ and a marking $\mathcal{M}$, model checking the formula $\mathbf{GF}cov(\mathcal{M})$ would be enough to decide the repeated coverability problem for the marking $\mathcal{M}$ and the net $\mathcal{N}$. As repeated coverability is undecidable for ν-Petri nets, model checking the considered fragment is undecidable.

□

Unfortunately, all the model checking problems that we have studied for Petri nets extensions are undecidable. In the next section we will explain what can be done when model checking is undecidable in order to try to obtain some information about the properties to check.

3.4 Beyond undecidability

In the previous sections we saw that many model checking problems are undecidable. We can only decide if a formula holds or not at a Petri net if the formula is expressed in some very restricted temporal logics, which are not very expressive. That means that there is no algorithm to decide the satisfiability of a formula for most of the useful temporal logics for Petri nets. Although this is not the goal of our work, we are going to give some bibliographic details about what can be done beyond undecidability and how to address efficiency issues.

3.4.1 Unfoldings

Let us recall another important problem that we saw in Section 3.1: the state space explosion problem. Even for finite systems, when we want to apply a model checking algorithm, the number of states grows exponentially with the size of the system. [1]

In order to palliate the previous problem we can use unfoldings. An unfolding is a compact representation of the different interleavings of a concurrent system [61, 24]. The following example intuitively illustrates how an unfolding of a Petri net is built:

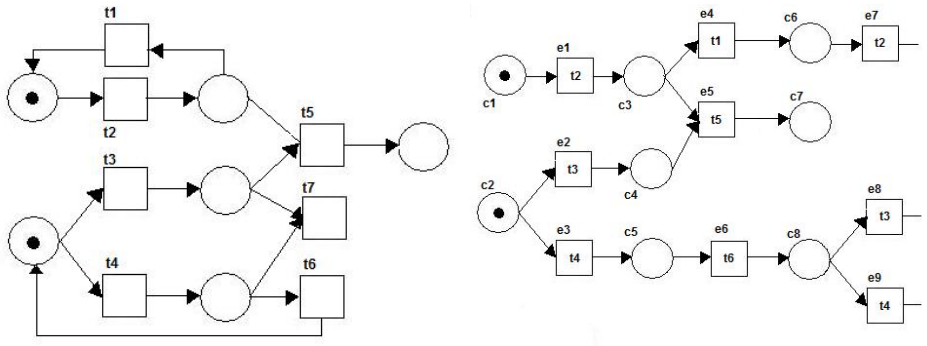

Figure 3.5: A PN and a prefix of its unfolding

Example 22 *Figure 3.5 shows a Petri net and a finite prefix of its unfolding. The nodes of an unfolding are called "conditions" and "events". They represent the occurrences of places and transitions of the corresponding net. Therefore, each place or transition can be represented by more than one condition or event, because they can be reached from different runnings. In the unfolding of Figure 3.5 the transition $t2$ is represented by two different events, because it can be fired following different transition sequences. We start building the unfolding of a Petri net by adding the conditions that represent the places which are marked at the initial marking. In this example these initial conditions are $c1$ and $c2$. Then, we process (with the help of a queue) all events which can be fired from the initial marking, until no more events can be added (or until we want to finish the unfolding), as follows:*

- *Remove the first event in the queue.*

[1] Actually, finite automata model checking for LTL is already PSPACE-complete.

- *Add the event and its postset (the postset of the transition it represents) to the unfolding.*
- *Identify the new possible events and insert them to the queue. Intuitively, the possibly events are those whose preconditions are in the unfolding and can be marked "simultaneously". For example, the transition t7 does not appear in the unfolding because its preconditions cannot be marked simultaneously.*

Let us explain how to perform the last step of the previous algorithm. More precisely, we are going to explain how to know if the preconditions of an event can be marked simultaneously. There are two fundamental concepts we have to explain: configurations and local configuration of an event. A configuration represents a partial running of a net. It is a set of events such that:

- If an event is in the configuration then all of its ancestors are in the configuration too.
- If e_1 and e_2 are two different events of a configuration, then ${}^\bullet e_1 \cap {}^\bullet e_2 = \emptyset$.

For example, the set of events $\{e1, e2, e4\}$ in the unfolding of figure 3.5 is a configuration. The local configuration of an event is the set composed of all of its ancestors (including itself). Then, a set of conditions can be simultaneously marked if the union of the local configurations of their presets form a configuration. For instance, in the previous example the configurations $c7$ and $c8$ can be marked simultaneously, because the set $\{e3, e6, e1, e5\}$ is a configuration.

Unfoldings are useful to explore the different runnings of a Petri net, so we can check some interesting properties. Petri nets have infinite states, so unfoldings of Petri nets are infinite, but McMillan identified the possibility of building a finite prefix of a Petri net unfolding which could give us enough information to solve several problems. The expensive part of the algorithm that McMillan proposed is the computation of the possible events to add to the unfolding, which is exponential in the maximal size of the presets of the transitions [27] (it is better than the complexity of the classic model checking algorithms).

The initial unfolding algorithms only allowed us to check reachability or deadlock, but these algorithms have been extended and now we can even check almost every property that is expressible in LTL [29]. More recently, in [2] P.A.Abdulla considers the coverability problem, and gives an algorithm to build a finite *backwards unfolding* of an unbounded Petri net. There are few cases in which unfold-

ing techniques has been applied to Petri net extensions. Recently, unfolding for coloured Petri nets has been studied for instance in [9].

3.4.2 Computation of the cover

Another technique which can be used in order to obtain a semialgorithm to model check certain properties of Petri nets is getting information from the so called *cover*, $\downarrow Post^*(\downarrow s)$, of a WSTS. The cover is not computable in general for well-structured transition systems, and in particular it is not computable for reset nets or for ν-Petri nets.

As we said before, given a Petri net with initial state s, $\downarrow Post^*(\downarrow s)$ gives us information about coverability and boundedness of the Petri net, and it can also be used to check some properties of certain temporal logics, as the logic **F** with the three basic operators (which we do not know if it is decidable or not for ν-PNs and reset nets, yet). Despite the fact that the algorithm to compute the cover does not terminate for ν-PNs and reset nets, so we cannot have complete algorithms based on the cover, if we want to model-check a formula of one of these temporal logics, we can start computing the cover until we find the answer to the problem we want to check (or until we consider we have spent too much time looking for an answer). This gives us a semi-algorithm for some logics, which could be useful in case of undecidability, but does not always terminate.

Chapter 4

Conclusions and future work

In this project we have discussed the existing results about model checking Petri nets and some of its monotonic extensions that exist in the literature. The next table summarizes the results on model checking for P/T nets, reset nets and ν-Petri nets. By "+" (resp. -) we denote that the logic we are considering is decidable (resp. undecidable). "?" denotes an open problem. In some cases, the references of the results are not given. In that case, the results follow directly from other results of the table, or are new. In particular, in this work we have proved the undecidability of the fragment **GF**+**F**+$first + cov + en$ for ν-Petri nets.

	P/T nets	**Reset nets**	**ν-PN**
UB	- [23]	-	-
EF	- [23]	-	-
EG	- [25]	-	-
YEN	+ [80]	?	?
μ-calculus + first (LTL)	+ [23]	- [7]	-
μ-calculus + first + cov	- [23]	-	-
F + first + cov + en	+ [44]	?	?
GF + F + first + cov + en	+ [45]	-	-

The table shows that all the model checking problems for reset nets and ν-PNs that have been studied in this project are undecidable. That exposes that more research in that issues is needed, not only to complete the table, but also to define

new logics and study them. More precisely, to find other temporal logics which would be decidable for the considered extensions and could express interesting properties would be interesting. In particular, defining new logics for ν-Petri nets in order to be able to express properties depending on their token names would be very useful, because in this extension the possibility of creating new names is added to P/T nets, so we would like to check properties related to the names. Recently, a new reachability logic for lossy counter machines has been studied in [76].

In the literature, most model checking problems are reduced to classical decidability results as the decidability of the reachability problem. A new result which is useful to prove the undecidability of certain logics for ν-PNs has been given in this project: the undecidability of the repeated coverability problem for ν-Petri nets. The proof consists on reducing the same problem for reset nets, which is undecidable, to ν-Petri nets. In fact, some of the proofs of undecidability in ν-Petri nets have been done by transferring reset net results to ν-PNs. Language theory was used to prove the difference of expressiveness between reset nets and ν-Petri nets in [75]. In this sense, it would certainly be interesting to find a logic which distinguishes between reset nets and ν-Petri nets.

As we have found that the model checking all the considered logics is undecidable for ν-Petri nets and reset nets, an important direction of research is the study of techniques that provide us with useful semialgorithms that try to answer to questions that are undecidable. One of these techniques is the computation of the so called *cover*, the set of markings $\downarrow Post^*(\downarrow s)$, where s is the initial marking of a net. In general, the cover of a well-structured transition system gives us a great deal of information about it, so we could use its computation as a semialgorithm for model checking certain properties whenever model checking them is undecidable. Regarding efficiency, unfolding techniques have been applied to P/T nets, and could be applied to reset nets and ν-Petri nets.

Bibliography

[1] P. A. Abdulla, K. Cerans, B. Jonsson, and T. Yih-Kuen. *"General decidability theorems for infinite-state systems"*. Logic in Computer Science, 313-321 (1996).

[2] P. A. Abdulla, S. Purushothaman Iyer and A. Nylén. *"Unfoldings of Unbounded Petri Nets"*. Lecture Notes in Computer Science, Springer Berlin/Heidelberg, Computer Aided Verification, 495-507 (2006).

[3] T. Araki and T. Kasami. *"Some decision problems related to the reachability problem for Petri nets"*. Theoretical Computer Science, 3, 85-104 (1977).

[4] M. F. Atig and P. Habermehl. *"On Yen's Path Logic for Petri Nets"*. LIAFA,CNRS and Univ. of Paris 7.

[5] M. Ben-Ari, Z. Manna and A. Pnueli. *"The Temporal Logic of Branching Time"*. Acta Informatica 20, 207-226(1983).

[6] J. Billington. *"Extensions to Coloured Petri nets and their applications to Protocols"*. PhD thesis, University of Cambridge (1991).

[7] A. Bouajjani and R. Mayr. *"Model Checking Lossy Vector Addition Systems"*. Lecture Notes In Computer Science, 323-333 (1999).

[8] M. C. Browne, E. M. Clarke and D. Dill. *"Checking the correctness of sequential circuits"*. Proc. 1985 Int. IEEE Conf. on Computer Design (1985).

[9] T. Chatain and E. Fabre. *"Factorization Properties of Symbolic Unfoldings of Colored Petri Nets"*. Petri Nets, 165-184 (2010).

[10] G. Ciardo. *"Petri Nets with Marking-Dependent Arc Cardinality: Properties and Analysis"*. Lecture Notes in Computer Science, 815, 179-198, Application and Theory of Petri Nets (1994).

[11] E. M. Clarke and E. A. Emerson. *"Synthesis of synchronization skeletons for branching time temporal logic"*. Proc. Workshop on Logic of Programs, 131 (1981).

[12] E. M. Clarke and I. A. Draghicescu. *"Expressibility Results for Linear Time and Branching Time Logics"*. Lecture Notes in Computer Science, 354, 428-437(1988).

[13] E. M. Clarke,D. E. Long and K. L. McMillan. *"A language for compositional specification and verification of finite state hardware controllers"*. Proc. IEEE 79 9, 1283-1292 (1991).

[14] E. M. Clarke, O. Grumberg, D. A. Peled. *"Model checking"*. The MIT Press, Cambridge (2000).

[15] E. M. Clarke and B. H. Schlingloff. *"Model Checking"*. Handbook of Automated Reasoning, 24, 1635-1790 (2001).

[16] M. Dam *"Fixpoints of Büchi automata"*. LFCS Refort ECS-LFCS-92-224, University of Edinburgh(1992).

[17] G. Decker, M. Weske. *"Instance Isolation Analysis for Service-Oriented Architectures"*. Proceedings of the 2008 IEEE International Conference on Services Computing, 1, 249-256 (2008).

[18] C. Dufourd, A. Finkel, and Ph. Schnoebelen. *"Reset Nets Between Decidability and Undecidability"*. Lecture Notes in Computer Science, 1443/1998, 103-115 (1998).

[19] C. Dufourd, P. Jančar, Ph. Schnoebelen. *"Boundedness of Reset P/T Nets"*. Lecture Notes In Computer Science, 1644, 301-310 (1999).

[20] C. Dufourd, A. Finkel, P. Jančar and Ph. Schnoebelen. *"Complexity of some basic problems for Petri nets with special arcs"*. Unpublished.

[21] E. A. Emerson and E. M. Clake. *"Using branching time logic to synthesizesynchronization skeletons"*. Science of Computer Programming 2, 241-266 (1982).

[22] J. Esparza and M. Nielsen. *"Decidability Issues for Petri Nets"*. BRICS Report Series, RS-94-8 (1994).

[23] J. Esparza. *"On the decidability of model checking for several μ-calculi and Petri nets"*. CAAP'94, LNCS 787, 115-129 (1994).

[24] J. Esparza, S. Römer, and W. Vogler. *"An improvement of McMillan's unfolding algorithm"*. Proc. of 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, 87-106 (1996).

[25] J. Esparza. *"Decidability of model checking for infinite-state concurrent systems"*. Acta informatica, 34, 85-107 (1997).

[26] J. Esparza, A. Finkel, R. Mayr. *"On the Verification of Broadcast Protocols"*. Logic in Computer Science, 352-359 (1999).

[27] J. Esparza, S. Römer and W. Vogler. *"An Improvement of McMillan's Unfolding Algorithm"*. Formal methods in system desigc, 20, 3, 285-310 (2002).

[28] J. Esparza. *"Some applications of Petri Nets to the Analysis of Parameterised Systems"*. Talk at WISP-03 (2003).

[29] J. Esparza and K. Heljanko. *"Unfoldings: a partial-order approach to model checking"*. Springer London (2008).

[30] A. Finkel. *"A generalization of the procedure of Karp and Miller to well structured transition systems"*. Lecture Notes in Computer Science, 267,499-508(1987).

[31] A. Finkel. *"Well structured transition systems"*. Research Report 365, Lab. de Recherche en Informatique. Univ Paris-Sud, Orsay(1987).

[32] A. Finkel. *"Reduction and covering of infinite reachability trees"*. Information and Computation, 89(2), 144-179 (1990).

[33] A. Finkel and Ph. Schnoebelen. *"Fundamental structures in well-structured infinite transition systems"*. Lecture Notes in Computer Science, 1380, 102-118 (1998).

[34] A. Finkel and Ph. Schnoebelen *"Well-structured transition systems everywhere!"*. Theoretical Computer Science 256(1-2), 63-92 (2001).

[35] A. Finkel, J.-F. Raskin, M. Samuelides and L. Van Begin. *"Monotonic Extensions of Petri Nets: Forward and Backward Search Revisited"*. Electronic Notes in Theoretical Computer Science, 68(6), 85-106 (2002).

[36] A. Finkel, P. McKenzie and C. Picaronny. *"A well-structured framework for analysing Petri net extensions"*. Information and Computation, 195, 1-29 (2004).

[37] A. Finkel, G. Geeraerts, J.-F. Raskin, and L. Van Begin. *"On the ω-language expressive power of extended petri nets"*. Theoretical Computer Science 356(3), 374-386 (2006).

[38] A. Finkel, J. Goubault-Larrecq. *"Forward analysis for WSTS, Part I: Completions"*. Logic in Computer Science, 433-444 (2009).

[39] A. Finkel, J. Goubault-Larrecq. *"Forward analysis for WSTS, Part II: Complete WSTS"*. Lecture Notes in Computer Science, 5556, 188-199 (2009).

[40] M. H. T. Hack. *"Decidability questions for Petri nets"*. Ph. D. Thesis, MIT (1976).

[41] S. Haddad and D. Poitrenaud. *"Recursive Petri Nets"*. Acta Informatica 44, 463-508(2007).

[42] K. M. van Hee, N.Sidorova, M.Voorhoeve, J.M. van derWerf *"Generation of Database Transactions with Petri Nets"*. Fundamenta Informaticae, 93, 171-184 (2009).

[43] C. A. R. Hoare *"Communicating sequential processes"*. Communications of the ACM 21,8, 666-677 (1978).

[44] R. Howell, L. Rosier and H. Yen. *"A taxonomy of fairness and temporal logic problems for Petri nets"*. Theoretical Computer Science 82, 341-372 (1991).

[45] P. Jančar. *"Decidability of a Temporal Logic Problem for Petri Nets."*. Theoretical Computer Science 74, 71-93 (1990).

[46] K. Jensen. *"Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use"*. Monographs in Theoretical Computer Science, Springer-Verlag (1995).

[47] R. M. Karp and R. E. Miller. *"Parallel Program Schemata"*. Journal of Computer and system Sciences 3, 147-195 (1969).

[48] M. Kishinevsky, J. Cortadella, A. Kondratyev, L. Lavagno, A. Taubin and A. Yakovlev. *"Coupling asynchrony and interrupts: Place Chart Nets "*. Lecture Notes in Computer Science, 1248, 328-347 (1997).

[49] S. R. Kosaraju. *"Decidability of Reachability in Vector Addition Systems"*. Symposium on the Theory of Computing, 267-281 (1982).

[50] D. Kozen. *"Results on the propositional mu-calculus"*. Theoretical Computer Science, 27, 333-354 (1983).

[51] J. L. Lambert. *"A structure to decide reachability in Petri nets"*. Theoretical Computer Science, 99, 1, 79-104 (1992).

[52] R. Lazic, T. Newcomb, J. Ouaknine, A. Roscoe, and J. Worrell. *"Nets with tokens which carry data"*. Lecture Notes in Computer Science, 4546, 301-320 (2007).

[53] J. Leroux. *"Vector Addition System Reachability Problem In Less Than 7 Pages"*. LaBRI, Université de Bordeaux, CNRS (2010).

[54] R. J. Lipton. *"The Reachability Problem Requires Exponential space"*. Department of Computer Science, Research Report 62, Yale University (1976).

[55] I. A. Lomazova. *"Nested Petri nets: Multi-level and recursive systems"*. Fundam. Inform., 47, 283-293(2001).

[56] G. Lowe. *"Breaking and fixing the Needham-Schroeder public key protocol using FDR"*. Lecture Notes in Computer Science, 1055, 147-166 (1996).

[57] Z. Manna and A. Pnueli. *"Verification of Concurrent Programs: The Temporal Framework"*. The Correctness Problem in Computer Science, 215-273(1981).

[58] Z. Manna, and A. Pnueli. *"The Anchored Version of the Temporal Framework"*. Lecture Notes in Computer Science, 354, 201-284(1989).

[59] E. W. Mayr. *"Persistence of Vector Replacement Systems is decidable"*. Acta informatica 15, 309-318 (1981).

[60] R. Mayr. *"Undecidable Problems in Unreliable Computations"*. Lecture Notes In Computer Science, 1776 (2000).

[61] K. L. McMillan. *"Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits"*. Lecture Notes in Computer Science, 663, 164-177 (1992).

[62] K.L.McMillan. *"Symbolic Model Checking"*. Kluwer Academic Publishers (1993).

[63] R.Milner *"Communication and Concurrency"*. Prentice-Hall International, Englewood Cliffs (1989).

[64] R. Milner, J. Parrow and D. Walker *"A Calculus of Mobile Processes"*. Information and Computation 100, 1, 1-40 (1992).

[65] M. L. Minsky. *"Computation: Finite and Infinite Machines"*. Prentice-Hall, (1971).

[66] R. Needham and M. Schroeder. *"Using encryption for authentification in large networks of computers"*. Communications of the ACM, 21, 993-999 (1978).

[67] R. M. Needham. *"Names"*. Distributed systems, ed. S. Mullender, 2nd ed., Reading, MA: Addison-Wesley, 315-326 and 531 541 (1993).

[68] C. A. Petri. *"Kommunikation mit Automaten."*. PhD thesis, Institut fur Instrumentelle Mathematik, Bonn (1962).

[69] A. Pnueli. *"The Temporal Logic of Programs"* Proc. of the 18th IEEE Symposium on Foundations of Computer Science, 46-57(1977).

[70] A. Pnueli. *"The Temporal Semantics of Concurrent Programs"* Theoretical Computer Science, 13, 1-20(1981).

[71] W. Reisig. *"Petri Nets, An Introduction"*. EATCS, Monographs on Theoretical Computer Science, Springer Verlag, Berlin, (1985).

[72] F. Rosa-Velardo, D. de Frutos-Escrig and O. Marroquín-Alonso. *"On the Expressiveness of Mobile Synchronizing Petri Nets"*. Electronic Notes in Theoretical Computer Science, 180, 77-94 (2007).

[73] F. Rosa-Velardo. *"Redes de Petri móviles para la especificación y verificación de propiedades de seguridad en sistemas ubicuos"*. Tesis Doctoral, Facultad de Ciencias Matemáticas, Universidad Complutense de Madrid (2007).

[74] F. Rosa-Velardo and D. de Frutos-Escrig. *"Name Creation vs. Replication in Petri Net Systems"*. Fundamenta Informaticae, 88, 3, 329-356 (2008).

[75] F. Rosa-Velardo and G. Delzanno. *"Language-based Comparison of Nets with Black Tokens, Pure Names and Ordered Data."*. In 4th International Conference on Language and Automata Theory and Applications, LATA 2010. Lecture Notes in Computer Science, 6031, pp. 524-535. Springer-Verlag (2010)

[76] Ph. Schnoebelen. *"Lossy Counter Machines Decidability Cheat Sheet"*. Lectures Notes Computer Science, 6227 (2010).

[77] R. Valk and M. Jantzen *"The Residue of Vector Sets with Applications to Decidability Problems in Petri Nets"*. Acta Informatica 21, 643-674(1985).

[78] R. Valk. *"Petri nets as dynamical objects"*. Workshop Proc. 16th International Conf. on Application and Theory of Petri Nets(1995).

[79] R. Valk. *"Object Petri nets: using the nets-within-nets paradigm"*. Lectures on Concurrency and Petri Nets, 819-848(2003).

[80] H. Yen. *"A unified approach for deciding the existence of certain petri net paths"*. Inf. Comput., 96(1):119-137 (1992).