
UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

# Supervisory Control of Petri Nets using Linear Temporal Logic

**Manuel Biscaia Martins**

Dissertação para obtenção do Grau de Mestre em

**Matemática e Aplicações**

**Júri**

| | |
|---|---|
| Presidente: | Doutora Maria Cristina Sales Viana Serôdio Sernadas |
| Orientador: | Doutor Pedro Manuel Urbano Almeida Lima |
| Co-orientador: | Doutor Paulo Alexandre Carreira Mateus |
| Vogal: | Doutor Jaime Arsénio de Brito Ramos |

**Janeiro, 2010**

# Resumo

Atendendo à necessidade de métodos autómaticos de síntese e análise de tarefas robóticas complexas, propõe-se uma metodologia que permite a um projectista de tarefas robóticas usando redes de Petri garantir propriedades relacionados com eventos, partindo de redes de Petri mais simples. Utilizam-se conceitos de trabalhos anteriores de *Supervisory Control* de sistemas de eventos discretos. As especificações são dadas em QPLTL, uma lógica temporal linear com quantificadores estritamente mais expressiva que LTL. Esta abordagem formal é suficientemente próxima aos nossos raciocínios internos, permitindo poupar tempo gasto ao projectar uma tarefa robótica em forma de rede de Petri. Devido à necessidade de investigar linguagens $\omega$ reconhecidas por redes de Petri para o foco principal do trabalho, sugere-se ainda uma extensão do conceito de autómato de Büchi generalizado, incluindo ainda um teorema que permite caracterizar as linguagens $\omega$ reconhecidas por redes de Petri.

### Palavras-chave

Redes de Petri, Supervisory Control, Linear Temporal Logic

# Abstract

Given the need of automatic methods for analysis and synthesis of complex robotic tasks, we propose a method that allows a designer who uses Petri nets as representations of robotic tasks to enforce an event based specification upon a simpler Petri net, utilizing concepts from earlier works in Supervisory Control of discrete event systems. The specifications are given in QPLTL, a quantified linear temporal logic strictly more expressive than LTL. This formalism is close enough to our thought processes, allowing us to effectively reduce the time spent when designing a Petri net robotic task. Due to need of researching Petri nets $\omega$-languages for our main line of work, we suggest an extension of the concept of generalized Büchi automata and present one Petri net $\omega$-language characterization theorem.

### Keywords

Petri net, Supervisory Control, Linear Temporal Logic

# Contents

# List of Figures

# Introduction

This initial chapter has two sections. The first one is the presentation of the dissertation. We will present the general and particular objectives, and how we expect to achieve the implicit goals. We will also discuss how this work is structured. The second section will present the reader with some basic knowledge about bag arithmetics, a small introduction to Language Theory, and finally some general notation used throughout the work.

## Goals and Structure of the Work

In modern robotics, the increasingly complex problems that need a solution present the robot designer with several challenges. After all, even a problem so easy to solve properly to Human intelligence like playing a soccer game is an enormous challenge to our most advanced robots, since the deconstruction of all our thought and physical processes concerned with playing a soccer game would probably fill a medium sized library. Furthermore, there are tasks much more complex than playing soccer. So, firstly, we should recognize that most of our actions are extremely complicated, if we analyze each subroutine. Secondly, we must then see that if we expect robots to be able to solve these increasingly complex tasks, then their design is going to get increasingly more complex; in particular the design of the robot actions and reactions in the environment, which is called the design of a robot task plan.

If we need to design increasingly complex robotic task plans, then we should aim to reduce the time spent by the designer in the construction and certification of a robotic task plan. In [20], Lacerda and Lima aim to achieve this goal by specifying certain parts of a robotic behaviour by means of a formalization closer to the natural thought processes of the designer. The idea presented there was applied to robotic task plans represented as finite state automata (FSA). The authors show that if we specify some desired constrains in a Linear Temporal Logic (LTL) formula, a formalization closer to a natural language than FSA, we can obtain the desired restrictions on a certain task plan computed from the original task plan and the specifications formula, by means of Supervisory Control [33, 32]. The resulting task plan will behave in the desired manner, presenting a more complex behaviour, and it will be easier to design.

Our aim is to create a similar procedure, making some extensions. The basic task plan will be represented as

a Petri net, a formalization with a higher degree of expressibility [29], and also easier to design and compose than finite state automata. Furthermore, our possible specifications are strictly more expressive, as we will use Quantified (Propositional) Linear Temporal Logic (QLTL or QPLTL)[34, 35], an extension of LTL.

Our procedure will start with a previously designed Petri net representing a robotic task, typically called a system. We will then specify a QPLTL formula representing the desired sequence of transitions. Using an equivalence between Büchi automata and QPLTL, we will construct the Büchi automaton that is the model of the specification. We will focus our method in terminating behaviours, as these are closer to the typical uses of robotic tasks. Since we are interested in terminating behaviours, some additional tunings are needed on the Büchi automaton representing the formula. The process will output a modified version of the system's Petri net such that all non-terminating behaviour has to be allowed by the temporal logic formula, and the finite behaviour is a good approximation of this infinite behaviour. Either way, we can see the QPLTL formula as a specification of a Supervisor for the original system. The resulting Petri net will represent the system Petri net/supervisor in the closed loop typical of Supervisory Control.

The fact that in the end the resulting object is still a Petri net is extremely important since we can then use techniques to analyze properties of the Petri net, and we can also use the Petri net as a diagnostic tool for possible improvements on the robot systems. For instance, we could search slower events for bottlenecks, optimize the implementation of the physical systems represented by the events, and if their local runtime was effectively improved, we would end up upgrading significantly the running time of the whole robot task plan.

The following figure should accompany us throughout the work, remembering what is the purpose of each section of the work. An inclusion drawn between $A$ and $B$ is such that there exists an element in $B$ for every element in $A$ such that $A$ and $B$ recognize the same language. $GPN_\phi$ and the question marks next to the links between it and the other elements represent the putative existence of a generalization of the concept of Büchi automata and of the links expected of such a generalization. The double dashed arrow linking $PN_\phi$ in $PN'_\phi$ crossing the barrier between finite and infinite behaviour represents a set of transformations that as we will see preserve some good properties originating in $\phi$.

This work will be split in three parts. The first part will have four chapters, and both other parts will have two chapters.

The first part will provide the reader with the necessary tools to understand our methodology on supervisory control of petri nets using temporal logic. This first part will have a chapter containing a brief review of FSA languages and FSA $\omega$-languages; the following chapter will provide the reader with the necessary knowledge of Petri nets to understand some elements of Petri net Language Theory, concerning both finite and infinite languages. The third chapter will give some basics in Temporal Logic, being specially concerned with Linear Temporal Logic. The fourth chapter will provide the user with basic knowledge of logical discrete event systems, and supervisory control over them.

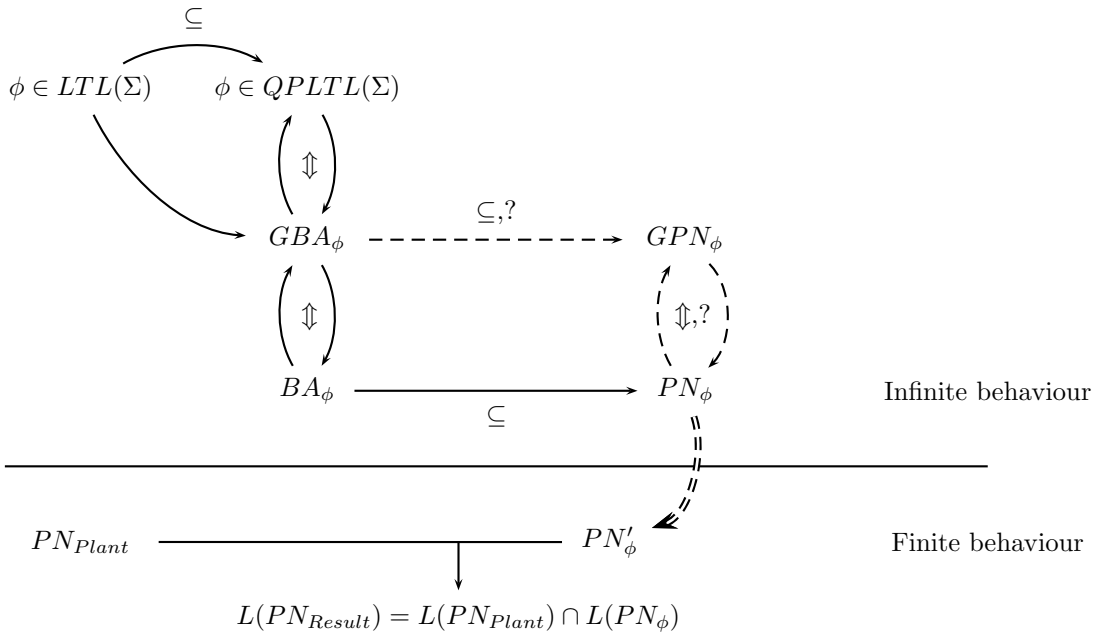


Figure 1: This graphical aid is intended to show the relations between the more used concepts mentioned in this work. The acronyms $BA$ and $GBA$ refer to a specific kind of finite state automaton over infinite sequences, the (generalized) Büchi automaton. The acronyms $PN$ and $GPN$ are similarly inspired concepts using Petri nets, instead of finite state automata. Any acronyms tagged with a $._\phi$ intends to be a reminder that the acronym is a *model* of $\phi$ in some way.

The second part will present the methodology proposed by us, using the introductory definitions of the earlier part. The first chapter of this part will present our methodology, and a thorough description of each of the methodology's steps. The second chapter will present some examples of the application of our proposed method, while discussing its strong and weak points, providing also some interesting modifications.

The third part will discuss some interesting results in Petri net $\omega$-languages Theory, which are directly related with the methodology proposed above. We present a small discussion on some of the appropriate Petri net infinite sequences accepting conditions, in the first chapter of the third part; this discussion will allows us to reprove a result already stated by Yamasaki[39]; the second chapter will present a generalization of the concept of generalized Büchi automata to Petri nets, proving an important theorem about this generalization.

After these three parts, in the last chapter, we will summarize our results, providing also some hints for future work.

There are some remarks to be made. The first is that this is supervisory control based on events, and consequently on languages. Our method directly affects the language created by the sequences of a Petri net. Not only this is an easier problem than the more difficult problem of restricting the reachable set of a Petri net, but it is also an inheritance from the early work in FSA. This also means that the designer should interpret the firing of transitions as *the robot is starting to do X*, which is definitely helpful if we intend to use our models as generalized stochastic Petri nets (GSPNs), for instance. Furthermore, as the result is a

Petri net, other methods of Supervision can be applied either before or after our methodology. The second remark is that we will not present the most efficient algorithms for most transformations. In fact, we will rarely analyze the efficiency of the transformations, since we are primarily concerned with showing that such a method exists. Optimization of the method proposed would be a future work idea.

# Preliminaries

In this section we will present some basic concepts needed in the following work. We will first show some basics on languages and then we will presents some concepts in Bag Theory.

Let $\Sigma$ be a set called the *alphabet*. Then, a *non-empty finite sequence* $s$ over $\Sigma$ is a function $s : \{1, 2, \ldots, n\} \mapsto \Sigma$ for some $n \geq 1$. The empty sequence is represented as $\lambda$. An *infinite sequence* $s$ over $\Sigma$ is a function $s : \mathbb{N} \mapsto \Sigma$. There is a basic operation over finite sequences called *concatenation*. The concatenation of two finite sequences $s_1, s_2$, represented by $s_1.s_2$ (sometimes the . is omitted), is a finite sequence $s_1.s_2 : \{1, \ldots, n_1, n_1 + 1, \ldots, n_1 + n_2\}$ where $s_1.s_2(n) = s_1(n)$, if $1 \leq n \leq n_1$, and $s_1.s_2(n) = s_2(n - n_1)$ if $n_1 < n \leq n_1 + n_2$. Note that the concatenation operation is not commutative, but it is associative. The set $\Sigma^*$ is the set of all finite sequences, empty or not. The set $\Sigma^\omega$ is the set of all infinite sequences. A *language* over $\Sigma$ is a subset of $\Sigma^*$. A *$\omega$-language* is a subset of $\Sigma^\omega$. The operation of concatenation can be extended to be executed between a finite sequence $s_1$ and an infinite sequence $s_2$. Note, however, that $s_1.s_2$ is a valid operation, while $s_2.s_1$ is not. The function $inf : \Sigma^\omega \mapsto \Sigma$ is defined as $inf(\alpha) = \{a \in \Sigma : \alpha(n) = a \text{ for infinitely many n}\}$. The function $ran : \Sigma^\omega \mapsto \Sigma$ is defined as $ran(\alpha) = \{a \in \Sigma : \alpha(n) = a \text{ for some n}\}$. The function $\omega Q : \Sigma^* \cup \Sigma^\omega \mapsto \{True, False\}$ returns $True$ if the argument is an infinite sequence and $False$ otherwise. The function $pref(\alpha) : \Sigma^* \cup \Sigma^\omega \mapsto 2^{\Sigma^*}$ is a language generating operation that produces the prefixes of a given finite or infinite word. If the word is finite, the language generated is finite, too; if the language is infinite, the language generated is infinite. Another related operation is prefix-closure. The prefix-closure of a language of finite sequences $L$ is the smallest language containing all prefixes of $L$. This operation is commonly represent as $\overline{L}$. A language is prefix-closed if $\overline{L} = L$.

There are many basic operations possible between languages. There are the normal set operations like *complement* $((.)^c)$, *union* $(L_1 \cup L_2)$ and *intersection* $L_1 \cap L_2$. All of these exist in both languages and $\omega$-languages. There are also some concatenation related operations like language concatenation $L_1.L_2$, which is only possible if $L_1$ is a language over finite sequences. Let the *concurrency operator* be represented by $||$ and be defined for $a, b \in \Sigma$, and $x_1, x_2 \in \Sigma^*$ such that $a.x_1||b.x_2 = a.(x_1||b.x_2) \cup b.(ax_1||x_2)$ and $a||\lambda = \lambda||a = a$. Then the concurrent composition of two languages is $L_1||L_2 = \{x_1||x_2 : x_1 \in L_1, x_2 \in L_2\}$. There are also two important operators $(.)^*$ and $(.)^\omega$, related with concatenation. The operator $(.)^*$ is called *Kleene star operator*. The Kleene star operator applied to a word $\alpha \in \Sigma^*$ is defined as follows: $\alpha^* = \bigcup_{i=0} \alpha^i$. The Kleene star operator of a language $L$, $L^*$ is defined as the smallest superset of $L$ that contains $\lambda$, and it is closed under the string concatenation operation. The $\omega$ operator applied to a word $\alpha \in \Sigma^*$ is defined

as the infinite concatenation of $\alpha$. The $\omega$ operator applied to a language $L$ produces the language over infinite sequences $L^\omega = \{\alpha_1 \dots \alpha_n \dots : \alpha_1, \dots, \alpha_n, \dots \in L - \{\lambda\}\}$. While not directly related to languages, the *projection function* $pr_i : A_1 \times \dots \times A_n \mapsto A_i$ is such that $pr_i((a_1, \dots, a_n)) = a_i$. It is also clear that these functions can be extended to both finite and infinite sequences, by simply applying it at each element of the sequence.

There are 4 main classes of languages: regular languages are the smallest class, followed by context-free languages, context-sensitive languages and finally type-0 languages. Regular languages are precisely the languages recognized by finite state automata, context-free Languages are recognized by pushdown stack automata, context sensitive languages are recognized by linear bounded Turing machines, and type-0 languages are the ones accepted by Turing machines. This hierarchy of languages is called the Chomsky hierarchy.

Bag theory is a natural extension of set theory. Just like sets, *bags* are collections of elements over some domain. However, bags allow multiple occurrences of elements. In set theory, there is the basic concept of *is a member of* relationship. The basic concept in bag theory is the *number of occurrences function*. For an element $x$ over a domain $X$, and a bag $B$, the number of occurrences of $x$ in $B$ is denoted by $\#(x, B)$. Consequently we have that $\#(x, B) \geq 0$ for all bags $B$ and elements $x$. We say that $x \in B$ is $\#(x, B) > 0$ and $x \notin B$ if $\#(x, B) = 0$. The *empty bag* , $\emptyset$, is the bag such that $\#(x, \emptyset) = 0$, for all elements $x$. The *cardinality* $\#B$ is the total number of occurrences of elements in the bag given by $\#B = \sum_x \#(x, B)$. $A$ is a *subbag* of $B$, $A \subseteq B$, if for all elements $x$, $\#(x, A) \leq \#(x, B)$. The bag $A$ is *equal* to the bag $B$, $A = B$, if for all elements $x$, $\#(x, A) = \#(x, B)$. We obtain the expected relation that $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$. There are four operations defined over bags. Let $A, B$ be two bags and $x$ be an element of the domain $D$:

$$A \cup B : \quad \forall_{x \in D} \quad \#(x, A \cup B) = max(\#(x, A), \#(x, B)),$$

$$A \cap B : \quad \forall_{x \in D} \quad \#(x, A \cap B) = min(\#(x, A), \#(x, B)),$$

$$A + B : \quad \forall_{x \in D} \quad \#(x, A + B) = \#(x, A) + \#(x, B),$$

$$A - B : \quad \forall_{x \in D} \quad \#(x, A - B) = \#(x, A) - \#(x, A \cap B).$$

Union, intersection and sum are commutative and associative. However, their relationship with difference is not trivial since for instance: $(A - B) + B \neq A$, for some bags $A, B$. The relationship is true, for instance if $B \subseteq A$. The set of all bags over a domain $D$ is represented as $D^\infty$. Assume the domain is $D = \{d_1, d_2, \dots, d_n\}$; then, there is a natural correspondence between each bag $B$ over $D$ and the $n$-vector $f = (f_1, f_2, \dots, f_n), f_i = \#(d_i, B)$. This correspondence is known as the *Parikh mapping*, introduced in [27]. When using Parikh mappings we tend to use the index order of the domain. As we will see further ahead, we will use both representations interchangeably, and so we are forced to index the elements of our domains, which unfortunately ends up complicating the notation used.

# Part I

# Introductory Concepts

# 1

# Finite State Automata

Finite state automata (FSA) are one of the simpler kinds of automata, and still they are able to decide a significant portion of languages. The chapter simply aims at providing the necessary knowledge to approach the following chapters. It has two sections: the first one deals with FSA on finite sequences, and the second deals with FSA over infinite sequences.

## 1.1 Finite state automata definitions on $\Sigma^*$

We will introduce some commonly used definitions of a finite state automata (FSA). These definitions are quite ubiquitous throughout the literature, for example in [16] or [4]; there are some additional functions that could be inserted in the FSA definition, and while the concept stays the same, it may be easier to express some FSA constructions if the definition is equipped with these additional functions.

More importantly, there is a distinction between deterministic finite state automata (DFSA) and nondeterministic finite state automata (NFSA). Although there is an algorithm that transforms a NFSA into a DFSA, this distinction is quite relevant, since the transformation is exponential on the number of states. We will introduce directly the definition of a NFSA and explain how it is related with DFSA.

**Definition 1.1.1.** A *finite state automaton* is a quintuple $(S, \Sigma, \delta, s_0, F)$, such that

- $S$ is a finite set, whose elements are called *states*,
- $\Sigma$ is a finite set, whose elements are called *events*,
- $\delta$ is a relation in $S \times \Sigma \times S$, called the *transition relation*,
- $s_0$ is an element of $S$, called the *initial state*,
- $F$ is a subset of $S$, called the *final states* or *accepting states*.

This is the standard definition of the more general NFSA. It is also possible to consider a version with more than one initial state, although it will always be possible to construct an equivalent FSA with only one initial state. A DFSA is a FSA where $\delta$ can be seen as a partial function where the co-domain is the set $S$.

**Definition 1.1.2.** Let $N = (S, \Sigma, \delta, s_0, F)$ be a FSA and $\alpha = (q_0, a_0, p_0) \dots (q_n, a_n, p_n) \in \delta^*$. $\alpha$ is called a *run*, represented by $s_0 \left|\alpha\right\rangle_N$ if:

$$q_0 = s_0,$$

$$\forall_{0 \leq i \leq n-1} \quad p_i = q_{i+1}.$$

Furthermore, it can be written $s_0 \left|\alpha\right\rangle_N = p_n$; it is also common to generalize this notation to other non-initial states.

With the definition of run, we are now ready to define the language accepted by a FSA, which is the basic definition that relates our structure with a set of sequences of events.

Let $N = (S, \Sigma, \delta, s_0, F)$ be an FSA. The *language generated* by $N$, represented by $L(N)$, is a subset of $\Sigma^*$ defined by:

$$L(N) = \left\{ pr_2(\alpha) : \alpha \in \delta^*, s_0 \left|\alpha\right\rangle_N \right\}.$$

Let $N = (S, \Sigma, \delta, s_0, F)$ be a FSA. The *marked language accepted* by $N$, represented by $L_m(N)$ is a subset of $\Sigma^*$ defined by:

$$L_m(N) = \left\{ pr_2(\alpha) : \alpha \in \delta^*, \exists_{f \in F} \quad s_0 \left|\alpha\right\rangle_N = f \right\}.$$

We will present a small example that merges these previous definitions.

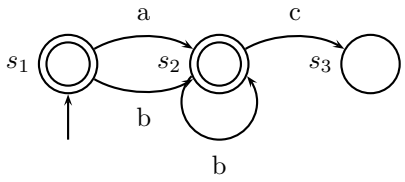


Figure 1.1: The figure depicts a common finite state automaton graphical representation. states are circles, elements of $\delta$ are arrows linking circles, the state marked by an arrow is the initial state and the final states are marked by an extra circle.

**Example 1.1.1.** The Figure 1.1 is a graphical representation of the following finite state automaton $M = (\{s_1, s_2, s_3\}, \{a, b, c\}, \{(s_1, a, s_2), (s_2, b, s_2), (s_1, b, s_2), (s_2, c, s_3)\}, s_1, \{s_1, s_2\})$. Note that it is a deterministic finite state automaton. A possible run would be $(s_1, a, s_2)(s_2, b, s_2)^2(s_2, c, s_3)$. The languages associated with this finite state automaton are $L(M) = \overline{(a+b)b^*c}$ and $L_m(M) = \overline{(a+b)b^*}$.

Another concept based on the idea of a run is the concept of reachability. In a FSA $N = (S, \Sigma, \delta, s_0, F)$, a state $t$ is said to be reachable if there exists a run $\alpha \in \delta^*$ such that $s_0 \left|\alpha\right\rangle_N = t$; a state $t$ is said to be co-reachable if there exists at least one final state $f \in F$ and a run $\alpha \in \delta^*$ such that $t \left|\alpha\right\rangle_N = f$.

There are some operations that can be made between FSA that still output a FSA. For example, union, intersection, Kleene star closure, and many more. The details on how to perform these operations will not be given except for an important operation: prefix-closure. The other operations can be consulted in [16]. The union operator allows us to define a FSA with more than one initial state.

The main idea behind prefix-closure of a marked language accepted by a FSA is to set all states as final states. In order to obtain the smallest prefix-closed language containing the initial marked language $L$ accepted by the FSA, it is necessary to trim the automaton first.

**Definition 1.1.3.** Let $N = (S, \Sigma, \delta, s_0, F)$ be a FSA. Then $trim(N) = (S', \Sigma, \delta', s_0', F')$ is defined in the following way: $S'$ is the subset of $S$ containing all reachable and co-reachable states. We can admit $s_0$ is co-reachable, as otherwise the accepted language is empty, and so $s_0' = s_0$. The transition relation is such that $\delta' = \delta_{|_{S' \times \Sigma \times S'}}$. Finally $F' = S' \cap F$.

With the *trim* operation, the following proposition is then valid [28].

**Proposition 1.1.1.** *Let $N = (S, \Sigma, s_0, \delta, F)$ be a FSA. Then, $L_m(N) = L_m(trim(N))$.*

Clearly, the languages generated by FSA are prefix-closed, hence we only apply this operation to marked languages accepted by FSA. In fact, if we apply the operation to a non-marked language accepted by a finite state automaton we will end up with the same language.

The prefix closure operator applied to the language $L$, represented by $\overline{L}$ is defined as the smallest prefix-closed language that still contains the starting language $L$.

**Proposition 1.1.2.** *Let $N = (S, \Sigma, \delta, s_0, F)$ be a FSA. Then $\overline{L_m(N)} = L(trim(N))$.*

Finally, if $\overline{L_m(N)} = L(N)$ the finite state automaton is said to be non-blocking. The definition of *non-blocking* system extends itself naturally to any system that has both an accepting condition representing all possible behaviour and an accepting condition only satisfied when some type of objective is reached.

## 1.2 Finite state automata definitions on $\Sigma^\omega$

The earlier definitions are the standard definitions commonly used to associate languages to FSA throughout Automata Theory. While regular languages are sufficiently rich to express many interesting behaviours, many more powerful systems exist, as for instance type-0 languages. However, these are only useful to describe *terminating* protocols, algorithms and circuits. But what about programs that do not terminate? A network

server for instance? When concurrent system investigation appeared, and the common temporal logics started to be developed, there was an obvious need of speaking about sets of infinite sequences ($\omega$-languages) in order to characterize non-terminating protocols. One of the questions that was answered by Büchi in [17] was to describe the analogous of regular languages for infinite sequences of events. The infinite version are named $\omega$-regular languages.

We will introduce five possible definitions of an $\omega$-language recognized by a FSA structure. Obviously, what will have to change is the acceptance criterion of the FSA, in order to change the semantics of the FSA structure. These five definitions were taken from [39], although they originated in [21]. Although we do not wish to study the reasons for the appearance of these particular accepting conditions, we will just mention that they are related to a topology on $\Sigma^\omega$, in particular the classes of languages that the accepting conditions generate correspond to certain initial elements in the Borel hierarchy over $\Sigma^\omega$, as shown in [21].

**Definition 1.2.1.** Let $N = (S, \Sigma, \delta, s_0, F)$ be a FSA and $\alpha = (q_0, a_0, p_0) \dots (q_n, a_n, p_n) \dots \in \delta^\omega$. $\alpha$ is called a *run*, represented by $s_0 \left|\alpha\right\rangle_N$ if:

$$q_0 = s_0,$$

$$\forall_{i \in \mathbb{N}} \quad p_i = q_{i+1}.$$

Let $N = (S, \Sigma, \delta, s_0, F)$ be a FSA. The $\omega$-languages accepted by $N$ are defined as follows:

$$L^\omega_{True}(N) = \{pr_2(\alpha) : \alpha \in \delta^\omega, s_0 \left|\alpha\right\rangle_N\},$$

$$L^\omega_{ran\cap}(N) = \{pr_2(\alpha) : \alpha \in \delta^\omega, s_0 \left|\alpha\right\rangle_N, ran(pr_1(\alpha)) \cap F \neq \emptyset\},$$

$$L^\omega_{ran\subseteq}(N) = \{pr_2(\alpha) : \alpha \in \delta^\omega, s_0 \left|\alpha\right\rangle_N, ran(pr_1(\alpha)) \subseteq F\},$$

$$L^\omega_{inf\cap}(N) = \{pr_2(\alpha) : \alpha \in \delta^\omega, s_0 \left|\alpha\right\rangle_N, inf(pr_1(\alpha)) \cap F \neq \emptyset\},$$

$$L^\omega_{inf\subseteq}(N) = \{pr_2(\alpha) : \alpha \in \delta^\omega, s_0 \left|\alpha\right\rangle_N, inf(pr_1(\alpha)) \subseteq F\}.$$

It is also natural to define $E_\gamma = \left\{L \subseteq \Sigma^\omega : \text{there exists a FSA } N \text{ such that } L = L^\omega_\gamma(N)\right\}$, for $\gamma = \{True, ran\cap, ran\subseteq, inf\cap, inf\subseteq\}$.

It was found initially on Büchi's work [17] that the class of languages $E_{inf\cap}$ corresponded to a certain second order logic. Furthermore, that second order logic was shown to be precisely as expressive as the definition of a $\omega$-regular language. We will now try to give some intuitive meaning to each of the definitions. Choose a finite state automata $N = (S, \Sigma, \delta, s_0, F)$.

The definition of $L^\omega_{True}(N)$ is, simply put, the set of all infinite fireable words of $N$; all the other definitions only accept $\tau \in \Sigma^\omega$ if $\tau \in L^\omega_{True}(N)$; only an infinite word that can be fired might belong to any infinite language accepted by an automaton, after all. The definition of $L^\omega_{ran\cap}(N)$ states that, not only the infinite

word has to be fireable, but it has to pass, at least once, by a final state. The third definition, $L^{\omega}_{ran\subseteq}(N)$, is much stronger. In order for a infinite word to be accepted, it has to be fireable and it can only pass by final states. The fifth definition, $L^{\omega}_{inf\subseteq}(N)$, can be summarized in the following manner: all the states that were reached infinitely often must be final states. This condition is equivalent to affirm that after some instant the automaton only passes through final states. This equivalence is due to the fact that the state space is finite. Finally the definition of $L^{\omega}_{inf\cap}(N)$ requires an infinite fireable word to be infinitely often in acceptable states, or equivalently there has to be a final state repeated infinitely often. The word can be infinitely often in non-accepting states, though, unlike the fifth definition.

It was proven in [36] the following relations:

**Theorem 1.2.1.** *Let $E_\gamma = \left\{L^{\omega}_{\gamma}(S) : S \text{ is a FSA}\right\}$ be the set of languages accepted by an FSA with one of the above accepting conditions, $\gamma = \{True, ran\cap, ran\subseteq, inf\cap, inf\subseteq\}$. Then, considering strict inclusions,*

$$E_{True} = E_{ran\subseteq} \subset E_{ran\cap} = E_{inf\subseteq} \subset E_{inf\cap}.$$

These relations will be needed later on to prove analogous results for $\omega$-Petri Nets in Proposition 7.3.1. We will call a FSA that has been associated with the accepting condition $L^{\omega}_{inf\cap}(N)$ a *Büchi automaton*. There is a possible generalization of this definition. The motivation for the generalization is simple, and easy to understand. Furthermore it allows us to specify more complex accepting conditions in a more concise manner. If $s \in L^{\omega}_{inf\cap}(N)$, the run has to pass by some final state an infinite number of times. In fact, it can pass by only one of the final states, for instance. The accepting condition can then be seen as a disjunction. But what if we need a run to pass by various different states? In fact, what if there is a finite number of sets of final states, and the run has to pass infinitely often by all sets of final states? It is intuitive that this change allows a conjunctive specification. A finite state automaton with an accepting condition of this type is called a *generalized Büchi automaton*[38, 11].

**Definition 1.2.2.** A *generalized Büchi automaton* $GN = (S, \Sigma, \delta, s_0, \mathcal{F})$. All the tuple elements are the same as in the FSA, with the exception of the last one, $\mathcal{F} \in 2^{2^S}$, with $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$. The language accepted by $GN$, represented by $L^{\omega}_{\forall inf\cap}(GN)$ is defined by:

$$L^{\omega}_{\forall inf\cap}(GN) = \left\{pr_2(\alpha) : \alpha \in \delta^{\omega}, s_0 \left|\alpha\right\rangle_{GN}, \forall_{F_i\in\mathcal{F}} \quad inf(pr_1(\alpha)) \cap F_i \neq \emptyset\right\}.$$

It was proven that this extension does not add expressiveness, although the method to construct a regular Büchi automaton from a generalized Büchi automaton is quite cumbersome. The proof hinges on the fact that if there is a run that passes infinitely often by each of the set of states, we can assume it passes orderly and still infinitely often. For example, if there are 3 sets of final states, and a given run passes by them in the order $F_1, F_3, F_2, \ldots, F_1, F_3, F_2, \ldots$, it is easy to see that the run also passes infinitely often in the more orderly order $F_1, F_2, F_3, \ldots, F_1, F_2, F_3, \ldots$. The following proof appears in many places[38, 11].

**Proposition 1.2.1.** *Let $GN = (S, \Sigma, s_0, \delta, \mathcal{F})$ be a generalized Büchi automaton, where $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$. Then there exists a Büchi automaton $N' = (S', \Sigma, \delta', s'_0, F')$ such that $L^{\omega}_{\forall inf \cap}(GN) = L^{\omega}_{inf \cap}(N')$.*

**Proof.** See proof in the Appendix, page 113.

There is also an important trimming operation similar to the one mentioned earlier. It is possible to trim a Büchi automaton, or even a generalized Büchi automaton with a similar purpose as in the FSA. The method is language preserving. We are using the reference [28] for this trimming operation. Consider the following introductory definition about a finite state automaton $M = (S, \Sigma, \delta, s_0, F)$: a state $s$ is said to be *co-acessible* if there is a infinite sequence $\alpha \in \delta^{\omega}$ such that $s\,|\alpha\rangle_M$ and $inf(pr_1(\alpha)) \cap F \neq \emptyset$. An automaton is said to be *trim* if all states are reachable and co-accessible. Furthermore, we can also trim a Büchi automaton.

**Definition 1.2.3.** Let $N = (S, \Sigma, s_0, \delta, F)$ be a finite state automaton. Then $trim^{\omega}(N) = (S', \Sigma', s'_0, \delta', F')$ is defined in the following way:

- $S'$ is a subset of $S$ with all the reachable and co-accessible states,
- $\Sigma' = \Sigma$,
- $\delta' = \delta \cap (S' \times \Sigma \times S')$,
- $s'_0 = s_0$,
- $F' = F \cap S'$.

Again, we can admit $s_0 \in S'$, because otherwise $L^{\omega}_{inf \cap}(N) = \emptyset$, and this case is not very interesting.

**Proposition 1.2.2.** *Let $N = (S, \Sigma, s_0, \delta, F)$ be a FSA. Then $L^{\omega}_{inf \cap}(N) = L^{\omega}_{inf \cap}(trim^{\omega}(N))$.*

When considering trimmed Büchi automata, we can then state a important property, somewhat reminiscent of the non-blocking property discussed in the earlier section.

**Proposition 1.2.3.** *Let $N = (S, \Sigma, s_0, \delta, F)$ be a finite state automaton. If $L^{\omega}_{inf \cap}(trim^{\omega}(N)) \neq \emptyset$, then if $\sigma \in L_m(trim^{\omega}(N))$ then there exists $\omega \in \Sigma^{\omega}$ such that $\sigma\omega \in L^{\omega}_{inf \cap}(trim^{\omega}(N))$.*

**Proof.** If $\sigma \in L_m(trim^{\omega}(N))$ then the last state that is reached by following some $\alpha \in \delta^*$ with $pr_2(\alpha) = \sigma$ belongs to $F$. Furthermore, we also know that the state reached is co-reachable, since we are dealing with trimmed automata, which in fact means that there exists a continuation of $\alpha$, which we will call $\alpha_1 \in \delta^*$, such that we will reach the set of final states again at the end of $\alpha\alpha_1$. We can of course repeat the same argument, and state that there exists $\alpha' = \alpha_1\alpha_2 \ldots \alpha_n \ldots \in \delta^{\omega}$ such that $s_0\,|\alpha\alpha'\rangle_{trim^{\omega}(N)}$ and $inf(pr_1(\alpha\alpha')) \cap F \neq \emptyset$. Finally, this implies that there exists $\omega \in \Sigma^{\omega}$ such that $\sigma\omega \in L^{\omega}_{inf \cap}(trim^{\omega}(N))$. □

**Remark 1.2.1.** Note that in general $L_m(trim^{\omega}(N)) \neq L_m(N)$, as opposed to $L_m(trim(N)) = L_m(N)$.

We will finally present the version of the trimming operation applicable to generalized Büchi automata. This time we will say that a state $s$ is *co-accessible* if there exists a infinite sequence $\alpha \in \delta^\omega$ such that $s\,|\alpha\rangle_M$ and $\forall_{F_i \in \mathcal{F}} \quad inf(pr_1(\alpha)) \cap F_i \neq \emptyset$.

**Definition 1.2.4.** Let $GN = (S, \Sigma, \delta, s_0, \mathcal{F})$ be a generalized Büchi automaton. Then, consider the generalized Büchi automaton $trim^{\forall\omega}(GN) = (S', \Sigma, \delta', s_0', \mathcal{F}')$ defined below:

- $S'$ is the subset of all reachable and co-accessible states,
- $\Sigma' = \Sigma$,.
- $\delta' = \delta \cap (S' \times \Sigma \times S')$,
- $s_0' = s_0$,
- $\mathcal{F}' = \{F_1 \cap S', \ldots, F_n \cap S'\}$.

Using the above definition it is easy to see that, if $GN$ is a generalized Büchi automaton, $L^\omega_{\forall inf \cap}(GN) = L^\omega_{\forall inf \cap}(trim^{\forall\omega}(GN))$. We will not present the proof, since it is very similar to the one presented in [28] about Büchi automata.

**Proposition 1.2.4.** *Let $GN = (S, \Sigma, \delta, s_0, \mathcal{F})$ be a generalized Büchi automaton. Then $L^\omega_{\forall inf \cap}(GN) = L^\omega_{\forall inf \cap}(trim^{\forall\omega}(GN))$.*

# 2

# Petri Nets

In order to approach our goal, we will need a little more background about Petri nets than the presented about FSA. This chapter will be divided in three sections. The first one will introduce some basic notions about Petri nets structures and markings, and their evolution. The second section will discuss the various classes of Petri net languages for finite sequences, with a subsection to give a brief overview on the decidability of many interesting problems in Petri net Theory. The third section will introduce our proposed definitions for some appropriate infinite sequence accepting conditions for Petri nets. It will also discuss some of their properties. Petri net $\omega$-languages will be more deeply studied in Part III. Most of the definitions from the first two sections were adapted from [29]. Some of the third section definitions were taken from Yamasaki [39], and the rest were inspired by the cited article.

## 2.1 Basic Petri net notions

Petri nets were initially designed to model systems with interacting concurrent components. They were introduced, in a very different form, by Carl Adam Petri [30]. His work was further developed by Holt and others in the Information System Theory Project of Applied Data Research group [15]. The Petri net initial concepts that were then developed were closer to our currently used definitions. Petri net theory was then further developed by some members in the Computation Structures Group at MIT. The revised concepts were then advertised and disseminated in two important conferences supported by the MIT researchers. As written in [29], some of the initial works on Petri net theory are [30, 15, 14, 13].

In our current world, Petri nets are frequently used in Concurrent Systems, Manufacturing Systems, and even Chemistry and Biology. In fact it seems that Petri invented Petri nets to describe some chemical processes. Examples of the application of Petri nets to all these fields can quickly be found just by browsing the Internet. In particular, the Intelligent Systems Laboratory at ISR/IST has been applying, for some years now, Petri nets as representations of robotic tasks [1, 26, 20, 5]. Further work at ISR/IST has two different

approaches, with different goals:

- supervisory control of Petri nets, whose main goal is to investigate ways of creating more complex Petri nets plans reducing the designer workload; to accomplish such a goal, simpler specifications are given and then used as supervisors controlling the original robotic tasks plans;
- generalized stochastic Petri nets, whose main goal is to investigate the created models in order to optimize the robots behaviour. This investigation deals not only with certain optimizations of the task plan, but it also can be used to discover which parts of the robot should be enhanced first.

As it was discussed in the introduction of this work we will approach the first problem. The definitions that will follow were extracted from [29].

**Definition 2.1.1.** A *Petri net structure* $C$ is a four-tuple $C = (P, T, I, O)$ where $P = \{p_1, p_2, \ldots, p_n\}$ called *places*, $T = \{t_1, t_2, \ldots, t_m\}$ called *transitions*, with $P \cap T = \emptyset$, and $I : T \mapsto P^\infty$ is the *input* function, a mapping from transitions to bags of places, and $O : T \mapsto P^\infty$ is the *output* function, a mapping from transitions to bags of places.

A place $p_i$ is a *input place* of a transition $t_j$ if $p_i \in I(t_j)$; $p_i$ is a *output place* of a transition $t_j$ if $p_i \in O(t_j)$. It is possible to define $I' : P \mapsto T^\infty$ and $O' : P \mapsto T^\infty$ and still obtain an equivalent Petri net definition.

While much of the theoretical work on Petri nets is based on the formal definition introduced above, a much more practical approach is to interpret a Petri net structure as a bipartite directed multi-graph.

**Definition 2.1.2.** A *Petri net graph* is a bipartite directed multi-graph, $G = (V, A)$, where $V = \{v_1, v_2, \ldots, v_r\}$ is a set of vertices, $A = \{a_1, a_2, \ldots, a_s\}$ is a bag of directed arcs, $a_i = (v_j, v_k) \quad v_j, v_k \in V$. The set $V$ can be partitioned into two disjoint sets $P$ and $T$, and for each directed arc $a_i = (v_j, v_k)$ then either $v_j \in P, v_k \in T$ or $v_j \in T, v_k \in P$.

Ordinarily, a place is represented as an empty circle and a transition is represented as a bar. If $\#(a_i, A) > 1$, it is possible to draw only one arc and then label it with $\#(a_i, A)$, avoiding unnecessary complications; not only this notation is natural, but it also suggests another possible way of defining Petri net structures. We will give some examples of Petri net structures and their respective graphs.

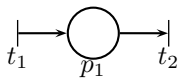


Figure 2.1: The typical representation of the Petri net structure $N = (\{p_1\}, \{t_1, t_2\}, I, O)$, with $I(t_1) = \emptyset$, $I(t_2) = \{p_1\}$, $O(t_1) = \{p_1\}$ and $O(t_2) = \emptyset$.

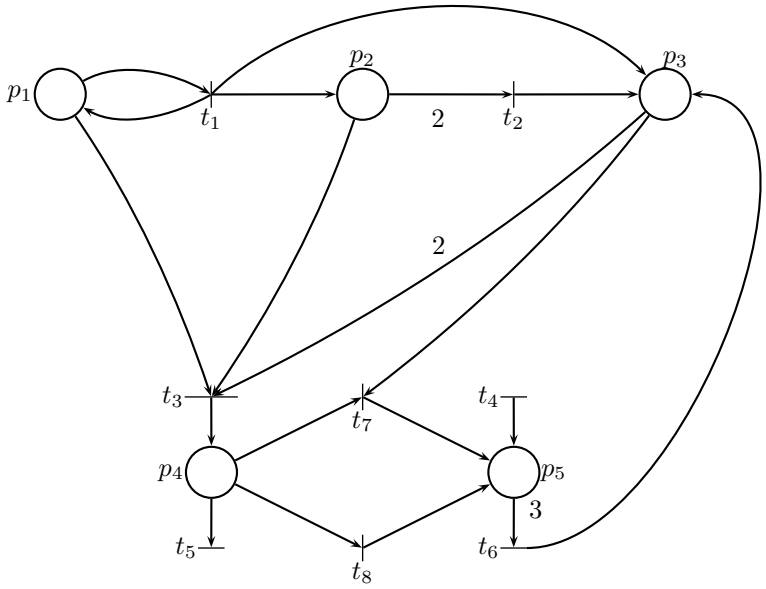


Figure 2.2: The typical representation of the Petri net structure $N = (P, T, I, O)$, with $P = \{p_1, p_2, p_3, p_4, p_5\}$ and $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$. We will just formalize the input and output of some of the transitions. $I(t_1) = \{p_1\}$, $O(t_1) = \{p_1, p_2, p_3\}$, $I(t_2) = \{p_2, p_2\}$, $O(t_2) = \{p_3\}$, $I(t_3) = \{p_1, p_2, p_3, p_3\}$ and $O(t_3) = \{p_4\}$.

A *marking* $\mu$ is an assignment of *tokens* to the places of a Petri net. A token is a primitive concept for Petri nets; tokens are assigned, and can be thought as residing in the places of a Petri net. We can then define a marking of a Petri net structure $C = (P, T, I, O)$ to be an element from $P^\infty$. A marking $\mu$ can also be seen as an element $(f_1, f_2, \ldots, f_{\#P}) \in \mathbb{N}^{\#P}$, where $f_i = \#(p_i, \mu)$. We will use both notations interchangeably. While the bag notation is more formal, it is not very practical either, so in many instances the Parikh mapping notation is preferred. Another useful marking related definition is the notion of a *filter* of markings. Let $F = \{\mu_1, \ldots, \mu_n\}$ be a set of markings. The *filter* of $F$, represented as $\uparrow F$, is defined as $\uparrow F = \{\mu : \exists_{\mu' \in F} \quad \mu' \subseteq \mu\}$.

With these marking related definitions in mind, we can now define a marked Petri net, which is simply the association of a marking to a Petri net structure. A marked Petri net $M = (C, \mu)$ is a Petri net structure $C = (P, T, I, O)$ and an initial marking $\mu$ or $\mu_0$. This is sometimes written as $M = (P, T, I, O, \mu)$. Typically a token is represented by a dot inside a place. The following example shows how we represent a marking in a Petri net. It is possible, however, that the number of tokens is inconveniently large, and so the corresponding natural number is written instead.

The execution of a Petri net is the sequence of markings and transitions followed by the Petri net as transitions *fire*. A transition is enabled depending on the current marking. Only enabled transitions may fire. When a transitions fires, it removes the tokens from its input places and creates new tokens in its output places, all according to the each arc weight. Note that it only removes the enabling tokens, or in other words, it only removes the tokens which are actually sufficient to enable a transition, meaning that any extra tokens are not removed.

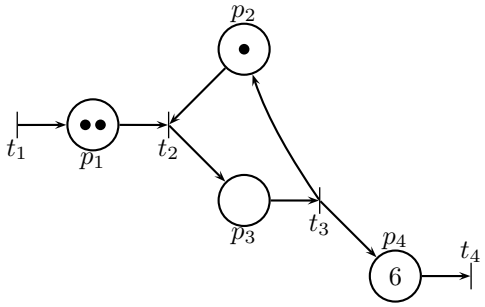


Figure 2.3: The marked Petri net $M$ shown is defined as follows $M = (P, T, I, O, \mu_0)$, with $P = \{p_1, p_2, p_3, p_4\}$, $T = \{t_1, t_2, t_3, t_4\}$. The input and output function can be determined from the figure. More importantly, the marking $\mu_0$ shown is the bag $\mu_0 = \{p_1, p_1, p_2\} + \sum_{i=1}^{6} \{p_4\}$. The Parikh mapping of the bag, assuming the index order of $p_i$s is being followed, is $\mu_0 = (2, 1, 0, 6)$.

**Definition 2.1.3.** A transition $t_j \in T$ in a marked Petri net $M = (P, T, I, O, \mu)$ is *enabled* for a marking $\mu = (\mu_1, \ldots, \mu_i, \ldots, \mu_{\#P})$ if, for all $i = 1, \ldots, \#P$, $\#(p_i, I(t_j)) \leq \mu_i$, or equivalently, $I(t_j) \subseteq \mu$. Assume now that $t_j$ is enabled in $\mu$. Then $t_j$ can fire and the marking reached after firing $t_j$ is given by:

$$\mu' = \mu - I(t_j) + O(t_j).$$

**Example 2.1.1.** The enabled transitions in the Figure 2.3 with the marking $\mu = \{p_1, p_1, p_2\} + \sum_{i=1}^{6} \{p_4\}$ are $t_1$, $t_2$ and $t_4$. If $t_1$ fires, the marking reached after the firing is $\mu' = \{p_1, p_1, p_1, p_2\} + \sum_{i=1}^{6} \{p_4\}$. If $t_2$ fires, the marking achieved will be $\mu' = \{p_1, p_3\} + \sum_{i=1}^{6} \{p_4\}$. Finally, if transition $t_4$ fires the marking reached will be $\mu' = \{p_1, p_1, p_2\} + \sum_{i=1}^{5} \{p_4\}$.

It is possible to create a *next-state* function, a concept similar to the existing one in FSA. It is useful to remind that, unlike FSA, the state of a Petri net is not a place, but instead the state is the distribution of tokens along the places.

**Definition 2.1.4.** Let $C = (P, T, I, O)$ be a Petri net structure. Then *next-state function* $\delta : \mathbb{N}^n \times T \mapsto \mathbb{N}^n$ at marking $\mu$ and for transition $t_j$ is defined if $t_j$ is enabled on marking $\mu$. In that case:

$$\delta(\mu, t_j) = \mu - I(t_j) + O(t_j).$$

Clearly, it is possible to modify the function to accept a sequence of transitions instead of a single one. We will give a recursive definition, with $\alpha \in T^*$:

$$\delta(\mu, \lambda) = \mu,$$

$$\delta(\mu, \alpha t_j) = \delta(\delta(\mu, \alpha), t_j).$$

This extension is only defined if all transitions are enabled on the relevant markings.

This function allows us to fire successive enabled transitions; it will also be used to define what is understood as a Petri net run.

**Definition 2.1.5.** Let $M = (P, T, I, O, \mu)$ be a marked Petri net and $\alpha = t_{j_1}, t_{j_2}, \ldots, t_{j_n} \in T^*$.

Then the sequence $(\mu, t_{j_1}, \mu_1), (\mu_1, t_{j_2}, \mu_2), \ldots, (\mu_{n-1}, t_{j_n}, \mu_n)$ is called a *Petri net run*, or a *run*, if $\mu_1 = \delta(\mu, t_{j_1})$ and $\forall_{1<i\leq n} \quad \mu_i = \delta(\mu_{i-1}, t_{j_i})$. A run is represented as $\mu \left|\alpha\right\rangle_M$. It is possible to write $\mu \left|\alpha\right\rangle_M = \mu_n$. It might be useful also to consider $M(\alpha)$, which is the sequence of the Petri net markings passed as $\alpha$ is executed. It is also common to generalize the definition to other markings, other than $\mu_0$.

**Example 2.1.2.** Using again Figure 2.3, we can see that $\alpha = t_1 t_2 t_3 t_4$ is a fireable transition sequence.

The actual Petri net run is:

$$\mu_0 \left|\alpha\right\rangle_M = ((2,1,0,6)t_1(3,1,0,6)), ((3,1,0,6)t_2(2,0,1,6)), ((2,0,1,6)t_3(2,1,0,7)), ((2,1,0,7)t_4(2,1,0,6)).$$

It would also be possible to state that $\mu_0 \left|\alpha\right\rangle_M = (2,1,0,6)$. On the other side, $t_2 t_2$ is not a fireable transition sequence, since before the second firing of $t_2$ it is necessary to check that $t_2$ is actually enabled, which it is not.

## 2.2 Petri net languages and some Petri net related problems

After having established the basic rules of Petri nets, and how the tokens evolve during execution, we are now interested in analyzing the sequences of transitions associated with a Petri net. Clearly, we will have to create an accepting condition, just like in FSA. Moreover, it is interesting to consider labeling functions; while we are already familiar with accepting conditions, it is necessary to introduce the concept of labeling functions: a labeling function is an application from the set of transitions to an alphabet. The definition can be trivially extended to a sequence of transitions, just by applying it to each transition. This application can be of three different types:

- It can be injective, which means that if $\sigma$ is the labeling function, then $\sigma(t_i) = \sigma(t_j)$ implies $t_i = t_j$. Furthermore, no transition can be mapped to the empty letter $\lambda$. Petri nets equipped with labeling functions of this kind are called *free-labeled Petri nets*.
- The only restriction on the second type of labeling functions is that no transition will be mapped to the empty letter $\lambda$. Petri nets equipped with these labeling functions are called *$\lambda$-free labeled Petri nets*.
- Furthermore, if the labeling function can also map a transition to the empty letter, the Petri nets equipped with this kind of labeling functions are called *$\lambda$-labeled Petri nets*. This is, of course the more general type of labeling functions.

Why is it necessary to introduce the concept of labeling functions? There are some practical considerations that must be present. For some reasons, the designer of the Petri net may wish to establish a certain relation of equivalence between all the different transitions. By using a labeling function, we alter the structure of the language recognized by the Petri net. It can be an extremely useful tool, as in some cases, even if the system is in totally different states, still the same response from the system should be expected.

From the three types of labeling functions will emerge a certain type of language. We are mainly interested in the second type, $\lambda$-free labeling functions. The free-labeling functions are quite restrictive, and the classes of languages obtained with $\lambda$-labeled Petri nets are quite hard to analyze.

**Remark 2.2.1.** The labeling function is represented on a Petri net graph by marking each transition with the mapped element of the alphabet. Thereafter, we will consider the alphabet to be some finite set $\Sigma$.

With labeling functions in mind, we are now ready to present the four main classes of Petri net languages.

**Remark 2.2.2.** Petri nets are, for the rest of this document, a marked Petri net structure, equipped with a labeling function, $\sigma$, and with a set of markings $F$, called the *accepting* or *final* markings; a Petri net $PN$ is ordinarily represented as $PN = (P, T, I, O, \mu_0, \sigma, F = \{\mu_1, \mu_2, \ldots, \mu_n\})$.

The first class of Petri net languages is one of the simplest. The following definition introduces the L-Type Petri net languages.

**Definition 2.2.1.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net. Let $\sigma : T \mapsto \Sigma$ be our labeling function. The L-Type language accepted by $PN$, represented by $L_L(PN)$, is defined as follows:

$$L_L(PN) = \left\{\sigma(\alpha) : \alpha \in T^* \text{ and } \exists_{\mu_f \in F} \quad \mu_0 \left|\alpha\right\rangle_{PN} = \mu_f\right\}.$$

This definition is similar to the marked languages accepted by a FSA, but it is often suggested that the fact that the sequence has to end exactly in a final marking is contrary to the general philosophy of Petri net. This comment is based on the observation that if $\delta(\mu, t_j)$ is defined then, for all $\mu \subseteq \mu'$, $\delta(\mu', t_j)$ is also defined. This simple observation gives birth to the G-Type Petri net languages.

**Definition 2.2.2.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net. Let $\sigma : T \mapsto \Sigma$ be our labeling function. The G-Type language accepted by $PN$, represented by $L_G(PN)$, is defined as follows:

$$L_G(PN) = \left\{\sigma(\alpha) : \alpha \in T^* \text{ and } \exists_{\mu_f \in \uparrow F} \quad \mu_0 \left|\alpha\right\rangle_{PN} = \mu_f\right\}.$$

There is another definition that originates from the possibility of deadlock, which is much more frequent in Petri nets than in FSA. In order to define the third type of language it is necessary to introduce the concept of *terminating states*. A state $\mu_s$ is *terminating* if $\delta(\mu_s, t_j)$ is undefined for all $t_j \in T$.

**Definition 2.2.3.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net. Let $\sigma : T \mapsto \Sigma$ be our labeling function.

The T-Type language accepted by $PN$, represented by $L_T(PN)$, is defined as follows:

$$L_T(PN) = \{\sigma(\alpha) : \alpha \in T^* \text{ and } \mu_0 \left|\alpha\right\rangle_{PN} = \mu_t \text{ with } \mu_t \text{ as terminating state}\}.$$

Finally, there is a class of languages defined as to accept all sequences of transitions between reachable markings. All these languages are prefix-closed, and, as we will shall see, they benefit from good properties that languages from other types may not have.

**Definition 2.2.4.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net. Let $\sigma : T \mapsto \Sigma$ be our labeling function. The P-Type language accepted by $PN$, represented by $L_P(PN)$, is defined as follows:

$$L_P(PN) = \{\sigma(\alpha) : \alpha \in T^* \text{ and } \mu_0 \left|\alpha\right\rangle_{PN}\}.$$

It is interesting to note that only the first two types of languages actually use the set of final markings. While it is true that it is possible to see the last two kinds of languages with certain types of final sets, the definitions presented underline the importance of the Petri net structure. For instance, we could compute all terminating states, and set them as final states. It is clear then that the L-Type language accepted by such a Petri net would be equivalent to its T-Type language.

All these definitions are valid, and currently in use, although the class of L-Type languages is more studied, especially since, as we will see, they enclose all the other languages. In our work, we will use more often the class of G-Type languages and P-Type languages as not only they are intuitive to work with, but also the acceptance criterion of the class of G-Type languages will be useful later on. We chose not to use the L-Type accepting criterion due to two reasons. First of all, the G-Type accepting criterion allows us to describe many of the L-Type languages much more quickly; furthermore, while as we shall see all G-Type languages are also L-Type languages, the Petri net transformation that proves this relation is extremely heavy, computationally speaking. Secondly, by actually using G-Type accepting conditions we are able to prove many interesting relations concerning Petri net languages of infinite sequences. Note, however, while each class of languages may be distinct, most of the time this problem is irrelevant for a Petri net designer; it is normally possible to model a given system within any of the first three classes of languages.

We will now show then that the class of G-Type languages with $\lambda$-free labeling functions are closed for some common language operations. These closure properties were found in [29] with reference for [13]. Similar properties could be proven for the other types of languages, but our main focus will be G-Type languages with $\lambda$-free labeling functions. First of all, the union of two G-Type languages with $\lambda$-free labeling functions is still a G-type language with $\lambda$-free labeling function.

**Proposition 2.2.1.** *Let $L_1$ and $L_2$ be two G-type languages with $\lambda$-free labeling functions. Then $L_1 \cup L_2$ is a G-Type language with a $\lambda$-free labeling function.*

The union construction is fairly simple, and allows us to assume, for instance, that there is more than one initial marking. This proof can be extended for L-Type Petri net languages and for P-Type Petri net languages. We will now show that the intersection of two Petri net languages is still a Petri net language. The construction is a bit harder than the previous one, specially since it leads to many spurious places which, for practical purposes, should be eliminated. This time we will present the construction, since it will be needed later on.

**Proposition 2.2.2.** *Let $L_1$ and $L_2$ be two G-type languages with $\lambda$-free labeling functions. Then $L_1 \cap L_2$ is a G-Type language with a $\lambda$-free labeling function.*

**Proof.** See proof in the appendix, on page 113.

Finally, two Petri nets can be composed concurrently and still output a Petri net. We will remind the reader that Petri nets were developed in order to simplify concurrent systems modeling, so the original purpose of the Petri nets is reflected on the extremely simple construction.

**Proposition 2.2.3.** *Let $L_1$ and $L_2$ be two G-type languages with $\lambda$-free labeling functions. Then $L_1||L_2$ is a G-Type language with $\lambda$-free labeling function.*

We will finally show how to translate between FSA to Petri nets. The reverse operation will also be referenced for the particular case of bounded Petri nets. A Petri net is $k$-bounded if all reachable markings have at most $k$ tokens in each place. If a certain language is not recognized by some bounded Petri net, then it is not recognizable by a FSA.

**Proposition 2.2.4.** *Let $L$ be a regular language such that $L = L_m(N)$ for some FSA $N = (S, \Sigma, s_0, \delta, F = \{s_1, \ldots, s_n\})$. Then, there exists a Petri net $PN' = (P', T', I', O', \mu'_0, \sigma', F')$, such that $L = L_G(PN')$.*

The idea behind the transformation is quite simple. It is necessary to transform all states into places, and create different transitions for each event accessible from each state. Let $P' = S$, $T' = \delta$, $I'(t) = I'((s, e, r)) = s$, with $t \in \delta$, $O'(t) = O'((s, e, r)) = r$, with $t \in \delta$, $\mu'_0 = \{s_0\}$, $\sigma'(t) = \sigma'((s, e, r)) = e$ and $F' = \{\{s_1\}, \ldots, \{s_n\}\}$. The proof of both inclusions should be now clear.

The idea behind the reverse transformation is simply to compute the reachability graph of the Petri net, a technique shown, for instance in [29, 1], and mark as final states all reachable markings belonging to $\uparrow F$. This operations produces a finite number of states, as the Petri net is bounded and so the state space is bounded. We will give a small example of a bounded Petri net and its reachability graph.

**Example 2.2.1.** More generally, the construction of the FSA structure using a marked Petri net is always the same; we only need to apply small modifications between different desired language types. For instance, if we were considering the P-Type language, we should not worry ourselves about the accepting states.

Now that we have the basics of Petri net languages, we will summarize how Petri net languages relate to each other. Most of this work was established by Hack [13], and, as we will not go into details on the proofs,

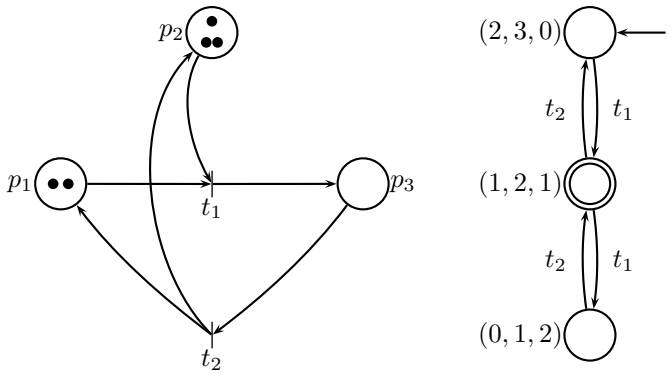


Figure 2.4: Consider the marked Petri net $PN$ drawn above. Consider the labeling function $id : T \mapsto T$. If the final markings set is $F = \{(1,2,1),(3,2,0)\}$, the automaton $M$ on the left is such that $L_m(M) = L_L(PN)$.

the relationships expressed here were taken from [29, 13]. We will only address languages originated from a $\lambda$-free labeling function.

First of all, it is easy to see that any P-Type Petri net language is a G-Type Petri net language. We simply have to set the final set of markings as the set with only the null marking (the marking with zero tokens in all places). Furthermore, it is possible to transform a Petri net $PN$ into a Petri net $PN'$ such that $L_G(PN) = L_L(PN')$. The inverse is not possible. For example the language $\{a^n.b^n,\, n \geq 0\}$ is a L-Type Petri net language but it is not a G-Type Petri net language. For the proof of this fact, consult [22]. The transformation involves adding for all existing transitions $t$, an additional transition with the same input as $t$, but the output is a proper subbag of $O(t)$, for each such subbag. This will allow to reach exactly the required marking of $F$, by choosing the adequate transition to fire, without producing excess tokens. Finally, it is also possible to show that if $PN$ is a Petri net, then there exists a Petri net $PN'$ such that $L_T(PN) = L_L(PN')$. This effectively means that the type of Petri net languages that offer better expressibility is the L-Type. The idea for this transformation is given in [29]. Furthermore, in [22], it is shown that there exists a $PN'$ for all Petri nets $PN$ such that $L_L(PN') = L_T(PN)$, showing that after all, for $\lambda$-free labeling functions, the T-Type accepting condition is equally expressive as the more natural L-Type accepting condition.

We will now briefly discuss how Petri net languages fit into the basic hierarchy of classes of languages. We already saw that Petri net languages contain FSA languages. Furthermore, the language $\{a^n.b^n,\, n > 1\}$ is a L-Type Petri net language. This language, however, is not a regular language. It is only a context-free language, which effectively means it can only be recognized by a stack automaton. There are, however, Petri net languages that are not context-free, they are context-sensitive, meaning that they can be recognized by a linear bounded Turing machine. An example is the L-Type Petri net language $\{a^n.b^n.c^n,\, n > 1\}$. Furthermore, there are context-free languages that are not recognizable by Petri nets. An intuition for this duality, given in [29], is the fact that the state space in a stack automaton grows exponentially, while the growth of state space in a Petri net is combinatorial. However, stack automata can only access the top of the stack, while Petri nets have access to all of its counters, which effectively gives more richness to the Petri net behaviour. All Petri nets languages are context-sensitive languages, and of course type-0 languages, recognizable by Turing machines. It is also interesting to note that if we introduce a new type of arc, called

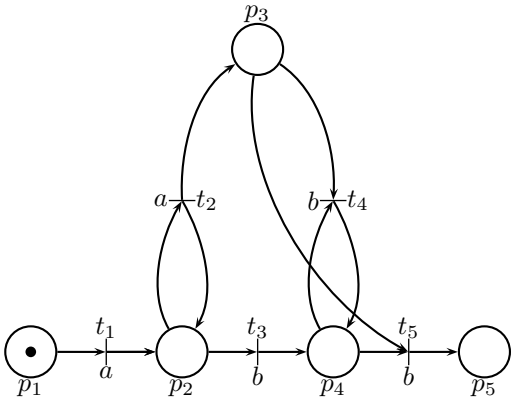


Figure 2.5: This Petri net $N$ is such that $L_L(N) = \{a^n.b^n,\ n > 1\}$, if we set $F = \{p_5\}$.

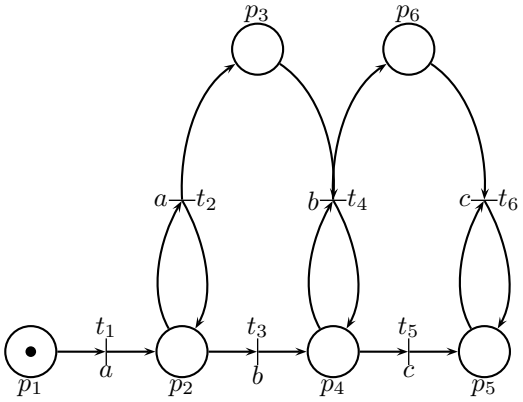


Figure 2.6: This Petri net $N$ is such that $L_L(N) = \{a^n.b^n.c^n,\ n \geq 1\}$, if we set $F = \{p_5\}$.

an inhibitor arc, the Petri net effectively gains the power of a Turing machine. This inhibitor arc feature can be seen in [29]. Moreover, most of the possible extensions of Petri nets have two possible effects: the extensions effectively are equivalent to inhibitor arcs and consequently to Turing machines, or the extensions can already be constructed using classical Petri net tools.

### 2.2.1 Decidability of Petri net problems

We shall now discuss the decidability of certain problems that arise naturally in the study of Petri nets, particularly those in Petri net language Theory. For instance, to decide whether a Petri net is bounded is computationally difficult, and it is known that it requires exponential space. The harder problem of deciding whether a marking $\mu$ is reachable was open for many years, and it is EXPSPACE-hard. It was only proven by Mayr [23] in 1981. Furthermore, the problem of deciding whether the reachability set of any two Petri nets is the same is undecidable, by being reducible to Hilbert's $10^{th}$ problem. These are some of the more

common problems in Petri nets.

In Petri net Language Theory, generally speaking, most problems concerning P-Type languages are known to be decidable for some time; problems concerning the L/T/G-Type languages were harder to solve, and they only were solved with the discovery of the algorithm to decide the reachability problem. Nevertheless, their computational complexity is very high. Some of the more common problems are:

- the Membership problem, which concerns the question whether a given word in an alphabet belongs to a language; this problem is quite easy to solve for both L and P types of languages, provided that we are using $\lambda$-free labeling functions
- the Emptiness problem, which concerns the question whether the Petri net language actually accepts any given word. This problem was only shown to be decidable in L-Type languages when the reachability problem was solved. This again means that solving the problem in a time/space efficient manner is impossible, at least for the general case. For P-Type languages the problem is trivial.
- the Finiteness problem, which concerns the questions whether a Petri net language is a finite set of words.
- the Equivalence problem, which is a problem similar to the equivalence of reachable markings, but now concerned whether two languages are the same or not. This problem is undecidable for general Petri nets.

For more information, consult the older review by Hack [13], or a more recent one made by Esparza and Nielsen [8] in 1994.

## 2.3 Petri net $\omega$-languages

As in FSA we are interested in studying the infinite sequences of events accepted by a Petri net. We will, once again, restate the usefulness of the concept of $\omega$-languages. The applications of the study of such infinite behaviour are many, as many procedures, protocols, algorithms, services, chemical reactions, are non-terminating. In our case, we intend, as described in the Introduction of this work, to create a Petri net that accepts precisely the same sequences of valuations as a certain (QP)LTL formula. In order to achieve our goal, we will have to define what infinite behaviours can be expected from a given Petri net. The set of all infinite behaviours of a Petri net will be a language; in our case, this language will coincide with the language specified by the QPLTL formula.

Just like in FSA, we will have to transform the accepting conditions of the different languages of Petri nets in order to accept infinite sequences. Initially, in [3], the $\omega$-languages accepting conditions presented were a mixture of L-Type Petri nets language acceptance criterion and the FSA infinite acceptance conditions.

The version we will present here is based on G-Type Petri net languages, and it is an adjustment from the one presented in [39]. The adjustment was direly needed, as we will see further ahead in Part III.

We will start by adapting some of the notation to their infinite version.

**Definition 2.3.1.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net and $\alpha = t_{i_1}, t_{i_2}, \ldots, t_{i_n}, \ldots \in T^\omega$. Then $(\mu_0, t_{i_1}, \mu_1), (\mu_1, t_{i_2}, \mu_2), \ldots, (\mu_{n-1}, t_{i_n}, \mu_n), \ldots$ is called a *Petri net infinite run* if $\mu_j = \delta(\mu_{j-1}, t_{i_j}) \quad \forall_j$. A run is represented as $\mu_0 \left|\sigma\right\rangle_{PN}$. It is also possible to represent $PN(\alpha) = \mu_0\mu_1\mu_2 \ldots \mu_n \ldots$ as the infinite sequence of markings.

As it will be seen, all the language definitions are related to the G-Type Petri net languages accepting condition.

**Definition 2.3.2.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net. Let $\sigma : T \mapsto \Sigma$ be the $\lambda$-free labeling function. Let $C = P^\infty$ be the infinite set of all markings, reachable or not. The $\omega$-languages accepted by $PN$ will be defined:

$$L^\omega_{True}(PN) = \{\sigma(\alpha) : \alpha \in T^\omega, \mu_0 \left|\alpha\right\rangle_{PN}\},$$

$$L^\omega_{ran\cap}(PN) = \{\sigma(\alpha) : \alpha \in T^\omega, \mu_0 \left|\alpha\right\rangle_{PN}, ran(PN(\alpha)) \cap \uparrow F \neq \emptyset\},$$

$$L^\omega_{ran\subseteq}(PN) = \{\sigma(\alpha) : \alpha \in T^\omega, \mu_0 \left|\alpha\right\rangle_{PN}, ran(PN(\alpha)) \subseteq \uparrow F\},$$

$$L^\omega_{inf\cap}(PN) = \{\sigma(\alpha) : \alpha \in T^\omega, \mu_0 \left|\alpha\right\rangle_{PN}, \omega Q(PN(\alpha)|_{\uparrow F}) = True\},$$

$$L^\omega_{inf\subseteq}(PN) = \{\sigma(\alpha) : \alpha \in T^\omega, \mu_0 \left|\alpha\right\rangle_{PN}, \omega Q(PN(\alpha)|_{C-\uparrow F}) = False\}.$$

Again, it is obvious that each language is a subset of $\Sigma^\omega$. For each accepting condition $\gamma = \{True, ran\cap, ran\subseteq, inf\cap, inf\subseteq\}$ we can define a class of Petri net language $P_\gamma = \left\{L^\omega_\gamma(PN), PN \text{ is a Petri net.}\right\}$.

Let us analyze carefully the meaning of each accepting condition, for any given $PN = (P, T, I, O, \mu_0, \sigma, F)$. If $\tau \in L^\omega_{True}(PN)$, then there exists an infinite sequence of transitions labeled by $\sigma$ as $\tau$ that is fireable. Just like in finite state automata, this restriction is used by all the other types of accepting conditions. If $\tau \in L^\omega_{ran\cap}(PN)$, then there exists an infinite sequence transitions labeled by $\sigma$ as $\tau$ that is fireable, and it passes by at least one marking in $\uparrow F$. If $\tau \in L^\omega_{ran\subseteq}(PN)$, then there exists an infinite sequence of transitions labeled by $\sigma$ as $\tau$ that is fireable, and it only passes by markings in $\uparrow F$. Up until now, there exists a clear parallel between these definitions and the FSA ones; as we will, even though the notation will change, the interpretations of the following accepting conditions will still coincide with the FSA ones. If $\tau \in L^\omega_{inf\cap}(PN)$, then there exists an infinite sequence of transitions labeled by $\sigma$ as $\tau$ that is fireable, and that passes by $\uparrow F$ an infinite number of times. We call a Petri net associated with this kind of accepting condition a *Büchi Petri net*. Finally, $\tau \in L^\omega_{inf\subseteq}(PN)$, then there exists an infinite sequence of transitions labeled by $\sigma$ as $\tau$ that is fireable, and that passes by markings not in $\uparrow F$ only a finite number of times. The following proposition should now be proven easily using Proposition 2.2.4.

**Proposition 2.3.1.** $E_\gamma \subset P_\gamma$.

Finally, we will end with one important remark: while these definitions, as we shall see in Part III, are invaluable, we could simply use either the definitions proposed in [3] or [39], since all these different definitions collapse if the Petri net is in fact a finite state automata, as it can be easily seen.

# 3

# Linear Temporal Logic

The development and conceptualization of Philosophical Logic was one of the greatest Greek inheritances that were passed on to our current day civilization. Even in those days, Humans actively studied the principles of valid reasoning and many types of logics appeared. All these logics, and many of our $20^{th}$ Century logics, are good formal models of our reasoning capabilities, and in many cases they are quite more intuitive to work with than computers or automata. In our case, we intend to use this intuitive connection between thoughts and logic by expressing interesting properties about a Petri net plant in an appropriate logic.

Modal logics are an important class of logics that explore the modalities of truth, whether it is *necessary* that $p$ occurs, or if it is only *possible* that $p$ occurs, for instance. With the advent of Kripke Semantics, Modal Logic had a powerful mathematical framework which stimulated research in the area. One very successful area of research of Modal Logics is Temporal Logic; this should not be a surprise, as it easy to see that the sentence $p$ *will occur always* or $p$ *will occur once* in the future share similar properties with *necessity* and *possibility*.

Either way, Amir Pnueli [31], argued that Temporal Logic could be extremely useful for verifying (and specifying, too) the correctness of computer programs, and especially useful for nonterminating computer programs, as there existed already a formalism for sequential programs, Hoare's Calculus, but it was not adequate for reactive nonterminating protocol. Amir Pnueli and Zohar Manna introduced the Linear Temporal Logic (LTL), and Edmund Clarke and E. Allen Emerson developed the Computational Tree Logic (CTL) and these still are the most utilized temporal logics.

All temporal logics are characterized by the basic concepts inbuilt in their syntax and semantics. Even if our objective in this chapter is to provide a small introduction to Linear Temporal Logic, we should clarify these inbuilt concepts, and present alternatives that originate other temporal logics.

One of the more evident concepts is the fact that Linear Temporal Logic directly extends Propositional Logic. In fact, LTL is frequently named Propositional Linear Temporal Logic (PLTL). The non-temporal part of

LTL is then classical propositional logic. Formulas are built using atomic propositions that are either true or false. Only then are the temporal modalities applied. Another approach would be to consider variables, functions, and predicates and build a First Order Linear Temporal Logic (FOLTL), which would extend LTL. There are however a number of possibilities to consider when using FOLTL. There are two main variants of First Order Linear Temporal Logic, that distinguish themselves by what is interpreted. Non-interpreted FOLTL does not make any special assumption on the interpretation of the domains, variables functions or predicates, while interpreted FOLTL can be fully interpreted, when all structures are fixed, or partially interpreted if only certain classes of structures are fixed. Furthermore, it is also common to restrict the syntax, allowing only Temporal modalities to be applied to first order formulas.

If the earlier concept dealt with the concept of Truth, since we are dealing with temporal logics, we now have to analyze Time. First of all, LTL has a discrete approach to Time. Since both LTL and CTL were intended to reason about computer programs, it is natural to consider the current, past and future states of the programs, which are discrete in nature. Tense logics where time is seen as the continuum were also studied by logicians, although they are fairly less utilized. Another distinction between Temporal logics is which time instants are considered. Both LTL and CTL only consider time instants in the future, using only future temporal modalities. Again, computer programs have an initial state, and it can be proven that LTL equipped with some past temporal modalities is equally expressive as LTL, although some properties are much easier to specify. Finally, there exists another important distinction that deals with how many futures might exist. If we allow more than one possible future at each instant, we are dealing with branching time logics, which CTL is an example. On the other side, if only one possible future exists then the logic is named a linear time logic.

The following chapter will have four sections. The first one will introduce LTL syntax. The second section will introduce LTL semantics. The third one will solely be about proving a relation between generalized Büchi automata and a LTL formula. The final section will introduce an extension of LTL, called QPLTL, and will adapt the relation to QPLTL formulas. Much of the beginning of this chapter was wrote based on [7]. The fourth section, in particular, was written using the article from Wolper [38]. The introduction of QPLTL was based in [7, 18, 37, 35]. The relation between Büchi automata and a QPLTL formula was extracted from [9].

## 3.1 Syntax

In this section we will introduce the formal syntax of Linear Temporal Logic. LTL formulas are formed using a set of propositional symbols $\Pi$, the truth-functional connectives $\neg, \wedge, \vee$, and three basic temporal modalities $X, U, R$. It is important to note that each set of propositional symbols defines a set of LTL formulas.

**Definition 3.1.1.** The set of *well formed LTL formulas* over the set of propositional symbols $\Pi$, represented by $LTL(\Pi)$, is defined inductively by the following rules:

- $True, False \in LTL(\Pi)$,
- if $p \in \Pi$, then $p \in LTL(\Pi)$,
- if $\phi \in LTL(\Pi)$, then $(X\phi), (\neg\phi) \in LTL(\Pi)$,
- if $\phi, \psi \in LTL(\Pi)$, then $(\phi \wedge \psi), (\phi \vee \psi), (\phi U \psi), (\phi R \psi) \in LTL(\Pi)$.

The formulas $(X\psi)$, $(\phi U \psi)$ and $(\phi R \psi)$ should be read respectively as *nexttime* $\psi$", $\phi$ *until* $\psi$, and $\phi$ *releases* $\psi$. There are some additional connectives that are introduced by abbreviation. Although the syntax is not minimal as for instance $\wedge$ and $R$ could be defined by abbreviation, we chose to present the LTL syntax in this way due to our future need of translating between LTL and Büchi automata.

**Definition 3.1.2.** Let $\Pi$ be a set of propositional symbols. Then, if $\phi, \psi \in LTL(\Pi)$ we define:

- $(\phi \Rightarrow \psi) \equiv_{abv} ((\neg\phi) \vee \psi)$,
- $(\phi \Leftrightarrow \psi) \equiv_{abv} ((\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi))$,
- $(F\phi) \equiv_{abv} (TrueU\phi)$,
- $(G\phi) \equiv_{abv} (FalseR\phi)$.

The first two connectives are also well known truth functional connectives already existing in Propositional Logic, and the bottom two are important temporal modalities whose origin is rooted in the Modal Logic definition. The formulas $(F\phi)$ and $(G\phi)$ are read, respectively, as *eventually* $\phi$ and *always* $\phi$.

## 3.2 Semantics

First of all, we will define the timeline, or in other words, how is time understood in Linear Temporal Logic; only then can we formalize the notion of satisfaction of a LTL formula.

In Linear Temporal Logic, the structure of time is a totally ordered set $(S, <)$, where $(S, <)$ is order isomorphic to the set of natural numbers $(\mathbb{N}, <)$. This implies, of course, that time is discrete, that it has an initial moment with no predecessors, and that it is infinite into the future. The first property reflects the fact that modern day computers are discrete, the second property is useful since the computation begins at an initial state, and finally the third property allows us to reason about nonterminating behaviour.

Consider the following suffix notation: if $\sigma : \mathbb{N} \mapsto A$, then $\sigma^i = \sigma(i)\sigma(i+1)\sigma(i+2)\ldots$. We are now ready to define the notion of satisfaction of a LTL formula.

**Definition 3.2.1.** Let $\sigma : \mathbb{N} \mapsto 2^{\Pi}$. Let $\phi \in LTL(\Pi)$. We write $\sigma \models^{LTL} \phi$ in order to state that *the infinite sequence of valuations* $\sigma$ *satisfies the LTL formula* $\phi$. We define the *satisfaction relation* $\models^{LTL}$ inductively on the structure of $\phi$:

- $\sigma \models^{LTL} True$,
- $\sigma \not\models^{LTL} False$,
- $\sigma \models^{LTL} p$ iff $p \in \sigma(0)$,
- $\sigma \models^{LTL} (\phi \wedge \psi)$ iff $\sigma \models^{LTL} \phi$ and $\sigma \models^{LTL} \psi$,
- $\sigma \models^{LTL} (\phi \vee \psi)$ iff $\sigma \models^{LTL} \phi$ or $\sigma \models^{LTL} \psi$,
- $\sigma \models^{LTL} (\neg\phi)$ iff $\sigma \not\models^{LTL} \phi$,
- $\sigma \models^{LTL} (X\phi)$ iff $\sigma^1 \models^{LTL} \phi$,
- $\sigma \models^{LTL} (\phi U \psi)$ iff $\exists_j(\sigma^j \models^{LTL} \psi$ and $\forall_{k<j}(\sigma^k \models^{LTL} \phi))$,
- $\sigma \models^{LTL} (\phi R \psi)$ iff $\forall_j(\sigma^j \not\models^{LTL} \psi$ implies $\exists_{k<j}(\sigma^k \models^{LTL} \phi))$.

If the satisfaction relation $\models^{LTL}$ is known from the context, we will simply write $\models$.

**Definition 3.2.2.** A formula $\phi \in LTL(\Pi)$ is *satisfiable* if there exists a infinite sequence of valuations $\sigma : \mathbb{N} \mapsto 2^{\Pi}$ such that $\sigma \models \phi$. In this case $\sigma$ is called a *model* of $\phi$. Furthermore, if all possible infinite sequences of valuations verify $\sigma \models \phi$, then $\phi$ is a *valid formula*. It shall be represented as $\models \phi$.

We will now provide a list of important valid formulas that will be used later on.

**Proposition 3.2.1.** *Let $\Pi$ be a set of propositional symbols. Let $\phi, \psi \in LTL(\Pi)$. Then the following formulas are valid:*

*We will present the important fixpoint characterization of the temporal modalities.*

- $\models ((F\phi) \Leftrightarrow (\phi \vee (X(F(\phi)))))$,
- $\models ((G\phi) \Leftrightarrow (\phi \wedge (X(G(\phi)))))$,
- $\models ((\phi U \psi) \Leftrightarrow (\psi \vee (\phi \wedge (X(\phi U \psi)))))$,
- $\models ((\phi R \psi) \Leftrightarrow (\psi \wedge (\phi \vee (X(\phi R \psi)))))$.

*We will also present a group of equivalences that can be used to write any LTL formula in a normal negated form, a form where the operator $\neg$ is only applied to propositional symbols.*

- $\models ((\neg False) \Leftrightarrow True)$,
- $\models ((\neg True) \Leftrightarrow False)$,
- $\models ((\neg(\neg\phi)) \Leftrightarrow \phi)$,

- $\models ((\neg(\phi \vee \psi)) \Leftrightarrow ((\neg\phi) \wedge (\neg\psi)))$,
- $\models ((\neg(\phi \wedge \psi)) \Leftrightarrow ((\neg\phi) \vee (\neg\psi)))$,
- $\models ((\neg(X\phi)) \Leftrightarrow (X(\neg\phi)))$,
- $\models ((\neg(\phi U\psi)) \Leftrightarrow ((\neg\phi)R(\neg\psi)))$,
- $\models ((\neg(\phi R\psi)) \Leftrightarrow ((\neg\phi)U(\neg\psi)))$.

We will assume thereafter that any LTL formula $\phi$ is in a normal negated form.

## 3.3 From LTL to generalized Büchi automata

In this section we will develop a method to generate a Büchi automata that accepts the subset of $(2^{\Pi})^{\omega}$ that satisfies a given $LTL(\Pi)$ formula. We will need some introductory definitions. These definitions were based on [38].

**Definition 3.3.1.** Let $\phi \in LTL(\Pi)$. We define the *closure* of $\phi$, represented by $cl(\phi)$, inductively:

- $\phi \in cl(\phi)$,
- if $(\phi_1 \vee \phi_2) \in cl(\phi)$, then $\phi_1, \phi_2 \in cl(\phi)$,
- if $(\phi_1 \wedge \phi_2) \in cl(\phi)$, then $\phi_1, \phi_2 \in cl(\phi)$,
- if $(X\phi_1) \in cl(\phi)$, then $\phi_1 \in cl(\phi)$,
- if $(\phi_1 U\phi_2) \in cl(\phi)$, then $\phi_1, \phi_2 \in cl(\phi)$,
- if $(\phi_1 R\phi_2) \in cl(\phi)$, then $\phi_1, \phi_2 \in cl(\phi)$.

The closure of a formula $\phi$ is intuitively the set of all subformulas that might influence the truth value of $\phi$.

**Definition 3.3.2.** Let $\phi \in LTL(\Pi)$ and $\sigma : \mathbb{N} \mapsto 2^{\Pi}$. A *valid closure labeling* of $\sigma$ is a function $\tau : \mathbb{N} \mapsto 2^{cl(\phi)}$ such that the following rules are satisfied:

1. $False \notin \tau(i)$,
2. for all $p \in \Pi$, if $p \in \tau(i)$ then $p \in \sigma(i)$ and if $(\neg p) \in \tau(i)$ then $p \notin \sigma(i)$,
3. if $(\phi_1 \wedge \phi_2) \in \tau(i)$ then $\phi_1 \in \tau(i)$ and $\phi_2 \in \tau(i)$,
4. if $(\phi_1 \vee \phi_2) \in \tau(i)$ then $\phi_1 \in \tau(i)$ or $\phi_2 \in \tau(i)$,
5. if $(X\phi_1) \in \tau(i)$ then $\phi_1 \in \tau(i+1)$,

6. if $(\phi_1 U \phi_2) \in \tau(i)$ then either $\phi_2 \in \tau(i)$ or $\phi_1 \in \tau(i)$ and $(\phi_1 U \phi_2) \in \tau(i+1)$ both have to be satisfied,

7. if $(\phi_1 R \phi_2) \in \tau(i)$ then $\phi_2 \in \tau(i)$ and either $\phi_1 \in \tau(i)$ or $(\phi_1 R \phi_2) \in \tau(i+1)$,

8. if $(\phi_1 U \phi_2) \in \tau(i)$ then there exists $j \geq i$ such that $\phi_2 \in \tau(j)$.

**Remark 3.3.1.** Notice that all of these rules mirror the definition of $\models$, and in particular the rules for $U$ and $R$ use the fixpoint characterization of the temporal modalities. Finally, notice that all rules, except the last, only take into account $\tau(i)$ and $\tau(i+1)$. The final rule is needed since the $U$ modality is not satisfied if the second argument is not satisfied, and the other rule referring to $U$ does not enforce that. The following lemma states that the valid closure labellings $\tau : \mathbb{N} \mapsto 2^{cl(\phi)}$ of $\sigma$ are functions that for each $i$, only contain formulas that are satisfied by $\sigma$ in the instant $i$.

**Lemma 3.3.1.** *Let $\phi \in LTL(\Pi)$ and $\sigma : \mathbb{N} \mapsto 2^{\Pi}$. If $\tau : \mathbb{N} \mapsto 2^{cl(\phi)}$ is a valid closure labeling of $\sigma$ then, if $\psi \in \tau(i)$ then $\sigma^i \models \psi$.*

**Proof.** This proof will be done by structural induction on $\phi$. The Basis case will contemplate $\phi = True$, $\phi = False$, $\phi = p$, with $p \in \Pi$, and $\phi = (\neg p)$, with $p \in \Pi$.

- $\sigma^i \models True$ for any sequence $\sigma$.
- Using rule 1, $False \notin \tau(i)$.
- if $p \in \tau(i)$, using condition 2, we know that $p \in \sigma(i)$. Then, by definition of satisfaction, $\sigma^i \models p$.
- if $(\neg p) \in \tau(i)$, using condition 2, we know that $p \notin \sigma(i)$. Then, using twice the definition of satisfaction $\sigma^i \not\models p$ and so $\sigma^i \models (\neg p)$.

The Step should contemplate all the introduced connectives, however, we will present the more delicate cases of the modalities $U$ and $R$.

- if $(\phi_1 R \phi_2) \in \tau(i)$, then $\phi_2 \in \tau(i)$, using rule 7. By the induction hypothesis, we know that $\sigma^i \models \phi_2$. Let $j > i$ be the smallest natural such that $\sigma^j \not\models \phi_2$. By the induction hypothesis, $\phi_2 \notin \tau(j)$ and by rule 7, $(\phi_1 R \phi_2) \notin \tau(j)$. As $(\phi_1 R \phi_2) \in \tau(i)$ and $(\phi_1 R \phi_2) \notin \tau(j)$ pick the smallest $i \leq k < j$, such that $(\phi_1 R \phi_2) \notin \tau(k+1)$. Then $(\phi_1 R \phi_2) \in \tau(k)$, $\phi_2 \in \tau(k)$ and as $(\phi_1 R \phi_2) \notin \tau(k+1)$, we conclude that $\phi_1 \in \tau(k)$. By the induction hypothesis, $\sigma^k \models \phi_1$. We conclude then that $\sigma^i \models (\phi_1 R \phi_2)$, using the definition of satisfaction. However, if there is not such $j > i$, then the semantic condition is vacuously satisfied.
- if $(\phi_1 U \phi_2) \in \tau(i)$, then by rule 8 there is $j \geq i$ such that $\phi_2 \in \tau(j)$. By the induction hypothesis, $\sigma^j \models \phi_2$. Choose the smallest such $j$. If $j = i$, then $\sigma^i \models (\phi_1 U \phi_2)$. Otherwise, consider all $i \leq k < j$. Then $\phi_2 \notin \tau(k)$. As $(\phi_1 U \phi_2) \in \tau(i)$, then $(\phi_1 U \phi_2) \in \tau(k)$ and $\phi_1 \in \tau(k)$, for all such $k$. Using the induction hypothesis, $\sigma^k \models \phi_1$. We can then conclude that $\sigma^i \models (\phi_1 U \phi_2)$.

□

We will now prove the other implication.

**Lemma 3.3.2.** *Let $\phi \in LTL(\Pi)$ and $\sigma : \mathbb{N} \mapsto 2^{\Pi}$ an infinite sequence of valuations. If $\sigma \models \phi$, then there exists a valid closure labeling $\tau : \mathbb{N} \mapsto 2^{cl(\phi)}$ of $\sigma$ such that $\phi \in \tau(0)$.*

**Proof.** Let $\tau$ be defined as $\psi \in \tau(i)$ if and only if $\sigma^i \models \psi$, for all $\psi \in cl(\phi)$. Since $\sigma \models \phi$, we have that $\phi \in \tau(0)$. It is obvious that $\tau$ is a valid closure labeling since the valid closure labeling rules were defined mirroring the LTL semantics. □

We can then unite the two lemmas in a necessary and sufficient condition for the satisfaction of a formula in terms of valid closure labellings.

**Lemma 3.3.3.** *Let $\phi \in LTL(\Pi)$ and $\sigma : \mathbb{N} \mapsto 2^{\Pi}$ an infinite sequence of valuations. Then $\sigma \models \phi$ if and only if there exists a valid closure labeling $\tau : \mathbb{N} \mapsto 2^{cl(\phi)}$ such that $\phi \in \tau(0)$.*

The idea behind the translation between a LTL formula and Büchi automaton is simple. Consider a certain sequence of valuations that satisfies a formula $\phi$. Consider then the infinite sequence of valuations created by the linear time structure. Our automaton has to accept this infinite sequence, and by doing so, it should pass by an infinite sequence of states. The idea is that a valid closure labeling of sequence of valuations will be precisely the sequence of states passed. Since we have an "if and only if" in the earlier lemma, we will also be able to conclude that if a word is accepted in the constructed automaton, then there exists a particular type of linear time structure that satisfies the formula $\phi$.

This construction was taken from [38]. One should also note that this algorithm is not an efficient one, in fact it is very far from it. However, it is more easier to implement and understand than more efficient algorithms [38, 11].

**Definition 3.3.3.** The language accepted by $\phi \in LTL(\Pi)$, represented as $L(\phi)$, is defined as:

$$L_{LTL}(\phi) = \left\{\sigma : \sigma : \mathbb{N} \mapsto 2^{\Pi}, \sigma \models \phi\right\}.$$

When considering the other LTL logics introduced in the next section, we will also use this notation replacing the subscript by the appropriate logic.

**Theorem 3.3.1.** *Let $\phi \in LTL(\Pi)$, with $\Pi$ a finite set. Then there exists a FSA $M = (X, 2^{\Pi}, \delta, S_0, \mathcal{F})$ such that $L^{\omega}_{\forall inf \cap}(M) = L_{LTL}(\phi)$.*

**Proof** (Sketch)**.** We will only show how to construct the FSA $M$. The set of states is a subset of $2^{cl(\phi)}$ such that $x \in X$ if and only if:

- $False \notin x$,
- if $\phi_1 \wedge \phi_2 \in x$ then $\phi_1, \phi_2 \in x$,

- if $\phi_1 \vee \phi_2 \in x$ then $\phi_1$ or $\phi_2 \in x$.

In order for $d \in \delta$, $d$ has to verify the following rules:

- for all $p \in \Pi$, if $p \in pr_1(d)$ then $p \in pr_2(d)$,
- for all $p \in \Pi$, if $(\neg p) \in pr_1(d)$ then $p \notin pr_2(d)$,
- if $(X\phi_1) \in pr_1(d)$, then $\phi_1 \in pr_3(d)$,
- if $(\phi_1 U \phi_2) \in pr_1(d)$, then either $\phi_2 \in pr_1(d)$, or $\phi_1 \in pr_1(d)$ and $(\phi_1 U \phi_2) \in pr_3(d)$,
- if $(\phi_1 R \phi_2) \in pr_1(d)$, then $\phi_2 \in pr_1(d)$ and, either $\phi_1 \in pr_1(d)$ or $(\phi_1 R \phi_2) \in pr_3(d)$,

The initial state is the subset $S_0$ of $X$ such that $S_0 = \{s \in X : \phi \in s\}$. In order for this construction to comply to the definition of a FSA, it is necessary to use the fact that we could always produce an equivalent Büchi automaton with only one initial state, and so we chose to present the more straightforward version. Finally, for final set $\mathcal{F}$, consider all formulas $(\psi_1 U \psi'_1), \ldots, (\psi_n U \psi'_n)$ in $cl(\phi)$. These formulas are called eventualities. Do not forget, that according to the valid closure labeling rules, the eventualities have to be fulfilled, meaning that if they continually appear, eventually we will reach a state where $\psi'_i$ has to appear. And so, $\mathcal{F} = \{F_1, \ldots, F_n\}$, with $F_i = \{x \in X : (\psi_i U \psi'_i), \psi'_i \in x \text{ or } (\psi_i U \psi'_i) \notin x\}$.

We will give an idea of the proof. For more details, consult [38]. We have to prove that $\sigma \in L^{\omega}_{\forall inf \cap}(M)$ if and only if $\sigma \in L_{LTL}(\phi)$. Using the lemma presented before, we have then to prove that $\sigma \in L^{\omega}_{\forall inf \cap}(M)$ if and only if there exists a valid closure labeling $\tau$ of $\sigma$ such that $\tau(0) = \phi$. The proof hinges on showing that if $\sigma$ is a valid sequence of events on the Büchi automaton, then there exists at least one sequence of states that were passed by. One of these sequences is the valid closure labeling $\tau$. It is also easy to see that the eighth rule of the valid closure labeling is fulfilled, since for each eventuality, using the Büchi automaton acceptance criterion we are ensured that, if we are in a non accepting state, after a finite number of events we will be in an accepting state again. □

## 3.4 Adaptation to QPLTL

Linear Temporal Logic has an interesting flaw: a result by Wolper [37] shows that certain fairly simple properties in LTL are not expressible in the logic. One such property is $(G_2 p)$, which means *p is true at all even instants*. Furthermore, while in the last section we discussed how to translate a LTL formula to a Büchi automaton, we did not approach the opposite problem. In fact, since Büchi automata can easily express $(G_2 p)$, we can already guess that a translation between Büchi automata and LTL is impossible.

In the context of our work, our specification is expressed by a LTL formula, but we have to translate it to a Büchi automaton, which is a formalism more expressible than LTL. In order to use the full expressiveness

of Büchi automata, we should look for a logic containing LTL with expressible power equivalent to Büchi automata. In this way, the designer, if he is only familiar with LTL, can still use the system; a more knowledgeable designer might effectively use the extra expressive power that the extension brings.

In the article already mentioned, Wolper suggests introducing in LTL operators corresponding to one or more right-linear grammars. This type of extension, called Extended Temporal Logic (ETL), is an effective extension, already powerful enough to specify ($G_2p$). Another such extension introduced in his dissertation thesis[34] by Sistla, is a quantified version of LTL. The logic developed was named Quantified Propositional Linear Temporal Logic (QPLTL); although it was then adapted to be used with both past and future temporal modalities, as seen in [18] known as Quantified Propositional Linear Temporal Logic with Both Temporal Operators (QPLTLB). This nomenclature is present in [7], and as such, we will follow it. Later on it was discovered that the introduction of past modalities did not introduce expressibility, and so the equivalent logic with only future quantifiers was named QPLTL. Either way, all these extensions have similar degrees of expressibility, and all of them are able to translate a Büchi automaton into a particular formula that accepts precisely the same language as the original automaton. Furthermore, the fragment derived from QPLTL by removing the universal quantifier, named Existentially Quantified Propositional Linear Temporal Logic, can be shown to be equally expressive; the fragment can be used to translate a Büchi automaton into a formula, as seen for instance in [9, 10]. Moreover, assuming we have a LTL to Büchi automaton translator, it is also very easy to create the EQLTL to Büchi automaton translator, which are good news since the QPLTL to Büchi automaton translator is an algorithm of *non-elementary space complexity*.

This section will first introduce the result by Wolper that certain properties are not expressible in LTL but they are expressible by Büchi automata. We will then present one version of QPLTL [10, 7], without the Past Temporal operators, while listing some important properties about QPLTL [35], and finally we will present EQLTL and show how to translate from Büchi automata to EQLTL and vice-versa [9, 10].

**Proposition 3.4.1.** *Given a proposition $p \in \Pi$, any $LTL(\Pi)$ formula $\phi$, with $p$ appearing on $\phi$, with less than $n$ $X$ (nexttime) connectives has the same value on all sequences of the form $p^i\emptyset p^\omega$, $i > n$ and $n \geq 0$.*

**Proof.** The proof is done using induction on the structure of $\phi$. Let the truth value of $\phi$ in $p^i\emptyset p^\omega$ be represented by $|\phi|_i$. The basis case is trivial: if $\phi = p$ then $p^i\emptyset p^\omega \models p$.

The step case concerning boolean connectives is also simple. For example if $\phi = (\phi_1 \vee \phi_2)$, then $p^i\emptyset p^\omega \models \phi_1$ or $p^i\emptyset p^\omega \models \phi_2$, by the induction hypothesis, and so $p^i\emptyset p^\omega \models (\phi_1 \vee \phi_2)$.

If $\phi = (G\phi_1)$, then $|(G\phi_1)|_i \Leftrightarrow |\phi_1|_i \wedge |\phi_1|_{i-1} \wedge \ldots |\phi_1|_{n+1} \wedge |(G\phi_1)|_n$. But, by the induction hypothesis, all the initial values are the same, and so $|(G\phi_1)|_i \Leftrightarrow |\phi_1|_{n+1} \wedge |(G\phi_1)|_n$.

If $\phi = (X\phi_1)$, then $|(X\phi_1)|_i = |\phi_1|_{i-1}$ and, by the induction hypothesis $|\phi_1|_{i-1}$ is independent of $i$ as $i - 1 > n - 1$ and $\phi_1$ contains less than $n - 1$ $X$ connectives.

If $\phi = (\phi_1 U \phi_2)$, then $|(\phi_1 U \phi_2)|_i = |\phi_2|_i \vee (|\phi_1|_{i-1} \wedge (|\phi_2|_{i-1} \vee (|\phi_1|_{i-1} \wedge \ldots \wedge (|\phi_2|_{n+1} \vee (|\phi_1|_{n+1} \wedge$

$|(\phi_1 U \phi_2)|_n))\ldots)))$ and so, by the induction hypothesis, $|(\phi_1 U \phi_2)|_i = (|\phi_2|_{n+1} \vee (|\phi_1|_{n+1} \wedge |(\phi_1 U \phi_2)|_n))$ which does not depend on $i$. □

And so Wolper proves that:

**Theorem 3.4.1.** *For any given $m \geq 2$, the property "p is true in every state $s_i$ where $i = km$ (integer $k \geq 0$)" is not expressible in LTL.*

**Proof.** Consider a formula $\phi$ that would express the pretended property for a certain $m$. It has a fixed number $l$ of $X$ connectives. Then, by the theorem, its truth value on $p^{km}\emptyset p^\omega$ and $p^{km-1}\emptyset p^\omega$ is the same, considering $k$ is such that $km - 1 > l$. But, clearly the required property is true on $p^{km}\emptyset p^\omega$ but not on $p^{km-1}\emptyset p^\omega$. So $\phi$ does not express the pretended property. □

We will now introduce QPLTL syntax and semantics, following the definitions on [10, 7]. As discussed above, we will introduce the quantified version of LTL without past temporal operators.

**Definition 3.4.1.** Let $\Pi$ be a set of propositional symbols. The set of *well formed formulas* in $QPLTL(\Pi)$ will be defined inductively:

- if $p \in \Pi$ then $p$ is a formula in $QPLTL(\Pi)$,
- $True$, $False$ are formulas in $QPLTL(\Pi)$,
- if $\phi, \psi$ are formulas in $QPLTL(\Pi)$, then $(\phi \vee \psi), (\phi \wedge \psi), (\neg\phi)$ are also formulas in $QPLTL(\Pi)$,
- if $\phi, \psi$ are formulas in $QPLTL(\Pi)$, then $(\phi U \psi), (\phi R \psi), (X\phi)$ are also formulas in $QPLTL(\Pi)$.
- if $\phi \in QPLTL(\Pi)$ and $\pi$ is a free propositional symbol in $\phi$, then $\exists\pi\phi \in QPLTL(\Pi)$.

A symbol $\pi \in \Pi$ is said to be free in formula $\phi$ as usual . Besides the LTL abbreviation introduced connectives, it is very useful to introduce $\forall \equiv \neg(\exists(\neg))$.

**Definition 3.4.2.** Let $\Pi$ be a set of propositional symbols, and let $\psi_1, \psi_2 \in QPLTL(\Pi)$. Consider also a $\sigma : \mathbb{N} \mapsto 2^\Pi$. We write $\sigma \models^{QPLTL} \phi$ in order to state that *the sequence of valuations $\sigma$ satisfies the QPLTL formula $\phi$*. We will define the *satisfaction relation* $\models^{QPLTL}$ inductively in the structure of $\phi$:

- $\sigma \models^{QPLTL} True$,
- $\sigma \not\models^{QPLTL} False$,
- $\sigma \models^{QPLTL} p$ iff $p \in \sigma(0)$,
- $\sigma \models^{QPLTL} (\phi \wedge \psi)$ iff $\sigma \models^{QPLTL} \phi$ and $\sigma \models^{QPLTL} \psi$,
- $\sigma \models^{QPLTL} (\phi \vee \psi)$ iff $\sigma \models^{QPLTL} \phi$ or $\sigma \models^{QPLTL} \psi$,

- $\sigma \models^{QPLTL} (\neg\phi)$ iff $\sigma \not\models^{QPLTL} \phi$,
- $\sigma \models^{QPLTL} (X\phi)$ iff $\sigma^1, L \models^{QPLTL} \phi$,
- $\sigma \models^{QPLTL} (\phi U \psi)$ iff $\exists_j(\sigma^j \models^{QPLTL} \psi$ and $\forall_{k<j}(\sigma^k \models^{QPLTL} \phi))$,
- $\sigma \models^{QPLTL} (\phi R \psi)$ iff $\forall_j(\sigma^j \not\models^{QPLTL} \psi$ implies $\exists_{k<j}(\sigma^k \models^{QPLTL} \phi))$,
- $\sigma \models^{QPLTL} (\exists\pi\phi)$ iff there exists a sequence of valuations $\sigma' : \mathbb{N} \mapsto 2^{\Pi}$, $\pi$-equivalent to $\sigma$, such that $\sigma' \models^{QPLTL} \phi$.

Two sequences of valuations $\sigma, \sigma'$ over $\Pi$ are said to be $\Gamma$-equivalent if $\forall_{p\in(\Pi-\Gamma)}\forall_{n\in\mathbb{N}} \quad p \in \sigma(n) \Leftrightarrow p \in \sigma'(n)$. If $\Gamma = \{\pi\}$ is a singleton, the notation $\pi$-equivalent is chosen over $\Gamma$-equivalent. If it is obvious which satisfaction relation we are writing about, we will ignore the superscript $(.)^{QPLTL}$.

In [35], the authors present a normal form for a QPLTL formula. Using the fact that Büchi automata are closed for complementation, they present an algorithm to transform a normal form of a QPLTL formula into a Büchi automaton. They also show that this transformation, when used to analyze satisfiability of a QPLTL formula, has non-elementary space complexity, as for each alternation of the quantifiers the space complexity is increased by exactly one exponential. We will now present the normal form described in the article above.

**Proposition 3.4.2.** *Let $\phi \in QPLTL(\Pi)$. Then, there exists $\psi \in LTL(\Pi)$, a finite sequence of existential or universal quantifiers $\{Q_1, Q_2, \ldots, Q_n\}$, and a finite sequence of propositional symbols $\{\pi_1, \pi_2, \ldots, \pi_n\}$ such that, if $\sigma : \mathbb{N} \mapsto 2^{\Pi}$, then $\sigma \models \phi$ iff $\sigma \models Q_1\pi_1 Q_2\pi_2 \ldots Q_n\pi_n\psi$.*

In order to understand the huge space complexity that characterizes the satisfiability problem for a $k$-alternating QPLTL formula in normal form we will present a small argument, already present in [35], that uses the known algorithms for complementing Büchi automata. Assume that our formula, already in normal form, is $\forall\pi_1\exists\pi_2\phi(\pi_1, \pi_2, \pi_3)$, for some $\phi \in LTL(\{\pi_1, \pi_2, \pi_3\})$. As discussed in the last section, we can construct a Büchi automaton, exponential in the size of $\phi$ that accepts exactly the same language as $\phi(\pi_1, \pi_2, \pi_3)$. Assume that this Büchi automaton has $n$ states. The Büchi automaton that accepts $\exists\pi_2\phi(\pi_1, \pi_2, \pi_3)$ has exactly the same number of states, and as we shall see, it is easy to construct. The main issue comes with the $\forall$ quantifier. First note that our formula is equivalent to $(\neg\exists\pi_1(\neg(\exists\pi_2\phi(\pi_1, \pi_2, \pi_3))))$, and that we already have a Büchi automaton for $\exists\pi_2\phi(\pi_1, \pi_2, \pi_3)$. If we use the complementation algorithm first, followed by the $\exists$ algorithm next, and finally we run another complementation algorithm, we obtain a Büchi automaton accepting precisely our initial QPLTL formula. Unfortunately, even using a more recent complementation algorithm [10], the number of states after the first run of the complementation algorithm is already $16^{n^2}$. The $\exists$ algorithm does not change the number of states, which means that after the second run of the complementation algorithm, the number of states obtained will be huge. As we saw here, using the abbreviation for $\forall$, means that we have to run the complementation algorithm once for each alternation of the quantifiers.

Fortunately, more recent investigations have shown that there exists a more specific normal form.

**Proposition 3.4.3.** *Let $\phi \in QPLTL(\Pi)$. Then, there exists $\psi \in LTL(\Pi)$, and a finite set of propositional symbols $\{\pi_1, \pi_2, \ldots, \pi_n\}$ such that, if $\sigma : \mathbb{N} \mapsto 2^{\Pi}$, then $\sigma \models \phi$ iff $\sigma \models \exists\pi_1 \exists\pi_2 \ldots \exists\pi_n \psi$.*

Clearly, the fact that this Normal form exists suggests that we may only consider the fragment of QPLTL obtained by applying existential quantifiers to LTL formulas. This fragment is naturally called EQLTL. EQLTL has another also expected property: it is possible to describe a Büchi automaton by an EQLTL formula. The syntax could be easily obtained by considering only the fragment of QPLTL; we chose to present it for completeness sake.

**Definition 3.4.3.** Let $\Pi$ be a set of propositional symbols. The set of *well formed formulas* in $EQLTL(\Pi)$ is defined inductively as follows:

- if $\phi \in LTL(\Pi)$ then $\phi \in EQLTL(\Pi)$,
- if $\phi \in EQLTL(\Pi)$, and $\pi \in \Pi$ is a free propositional symbol in $\phi$, then $\exists\pi\phi \in EQLTL(\Pi)$.

**Definition 3.4.4.** Let $\Pi$ be a set of propositional symbols. Consider also $\sigma : \mathbb{N} \mapsto 2^{\Pi}$. We write $\sigma \models^{EQLTL} \phi$ in order to state that *the sequence of valuations $\sigma$ satisfies the EQLTL formula $\phi$*. We will define the *satisfaction relation* $\models^{EQLTL}$ inductively in the structure of $\phi$:

- if $\phi \in LTL(\Pi)$ then $\sigma \models^{LTL} \phi$ iff $\sigma \models^{EQLTL} \phi$,
- if $\phi = \exists\pi\psi$, with $\psi \in EQLTL(\Pi)$, and $\pi$ is a free propositional symbol in $\psi$, then $\sigma \models \exists\pi\psi$ iff $\sigma' \models \psi$ for some $\sigma'$ $\pi$-equivalent to $\sigma$.

If it is obvious which semantic rules to apply, $\models^{EQLTL}$ is written $\models$.

So, finally, following [9], we establish the equivalence between Büchi automata and EQLTL. Since the EQLTL semantics given here are not exactly the same as in the article cited, we had to adapt the result. In the following theorem if $\sigma' : \mathbb{N} \mapsto 2^{\Pi'}$, with $\Pi \subseteq \Pi'$ then $\sigma'|_{\Pi} = \sigma'(0)|_{\Pi}\sigma'(1)|_{\Pi} \ldots \sigma'(n)|_{\Pi} \ldots$.

**Theorem 3.4.2.** *Let $M = (S, 2^{\Pi}, \delta, s_0, F)$ be a finite state automata. Then there exists $\phi \in EQLTL(\Pi \cup P)$, for some set of extra propositional symbols $P$, such that $\sigma \in L^{\omega}_{inf\cap}(M)$ iff $\forall \quad \sigma' : \mathbb{N} \mapsto 2^{\Pi \cup P}$ such that $\sigma'|_{\Pi} = \sigma$, $\sigma' \models \phi$.*

**Proof.** Consider $S = \{s^1, s^2, \ldots, s^n\}$. Define $P = \{ats_1, \ldots, ats_n\}$, with $ats_i$ representing "at state $s^i$".

Define $\phi$ as follows:

$$\begin{aligned}
\phi \equiv \exists ats_1 \exists ats_2 \ldots \exists ats_n (G(\bigwedge_{i \neq j}^{n} (\neg(ats_i \wedge ats_j)))) \wedge \\
( \bigvee_{\{(s_0,a,s^i) \in \delta\}} ((\bigwedge_{\pi_l \in a} \pi_l) \wedge (\bigwedge_{\pi_l \notin a} (\neg \pi_l))) \wedge ats_i) \wedge \\
(G( \bigvee_{\{(s^i,a,s^j) \in \delta\}} (ats_i \wedge (X(\bigwedge_{\pi_l \in a} \pi_l) \wedge (\bigwedge_{\pi_l \notin a} (\neg \pi_l))) \wedge (X ats_j)))) \wedge \\
( \bigvee_{\{s_f \in F\}} (G(F s_f)))
\end{aligned}$$

The first element of the conjunction assures that an automaton can only be in one state at each instant. The second element is an initializing element, in what states can you start. The third conjunction describe the transition rule $\delta$, and the fourth conjuction states the Büchi accepting condition. In [18], a similar formula in QPLTLB is provided, and although the details are different, there is a good description on how to prove the correctness of the translation. □

**Theorem 3.4.3.** *Let $\phi \in EQLTL(\Pi)$ and $\sigma : \mathbb{N} \mapsto 2^{\Pi}$. Then there exists a FSA $N$ such that $\sigma \models \phi$ iff $\sigma \in L^{\omega}_{inf\cap}(N)$.*

**Proof.** If $\phi \in LTL(\Pi)$ then, as we have seen in the last section, there exists a FSA $N$ such that $L_{LTL}(\phi) = L^{\omega}_{inf\cap}(N)$. Let now $\phi \in EQLTL(\Pi) - LTL(\Pi)$. Then, $\phi \equiv \exists \pi_{i_1} \exists \pi_{i_2} \ldots \exists \pi_{i_k} \psi$. Consider the FSA $N' = (S', 2^{\Pi}, \delta', s'_0, F')$ such that $L_{LTL}(\psi) = L^{\omega}_{inf\cap}(N')$. We will now modify the transition relation $\delta'$. Consider the FSA $N = (S = S', 2^{\Pi}, \delta, s_0 = s'_0, F = F')$, with $\delta$ defined as follows:

$$\text{let } v' = \{\rho_{k_1}, \ldots, \rho_{k_r}, \pi_{j_1}, \ldots, \pi_{j_s}\} \in \delta' \text{ in } (s_i, v', s_j) \text{ then}$$

$$(s_i, v_i, s_j) \in \delta \text{ with } v_i = \{\rho_{k_1}, \ldots, \rho_{k_r}\} \cup (2^{\{\pi_{i_1}, \ldots, \pi_{i_k}\}})_i$$

where in the last equation we are considering the $i$-th element of the set $2^{\{\pi_{i_1}, \ldots, \pi_{i_k}\}}$ under an arbitrary order. It is easy to see that $L_{EQLTL}(\phi) = L^{\omega}_{inf\cap}(N)$. We will prove both inclusions: Let $\phi = \exists \pi_{i_1} \exists \pi_{i_2} \ldots \exists \pi_{i_k} \psi$, $\sigma \in L_{EQLTL}(\phi)$ and $N'$ the FSA such that $L^{\omega}_{inf\cap}(N') = L_{LTL}(\psi)$.

$$\begin{aligned}
&\sigma \models \exists \pi_{i_1} \ldots \exists \pi_{i_k} \psi \Leftrightarrow \\
\Leftrightarrow &\exists \sigma' \ \{\pi_{i_1}, \ldots, \pi_{i_k}\}\text{-equivalent to } \sigma \text{ such that } \sigma' \models \psi \Leftrightarrow \\
\Leftrightarrow &\exists \sigma' \ \{\pi_{i_1}, \ldots, \pi_{i_k}\}\text{-equivalent to } \sigma \text{ such that } \sigma' \in L^{\omega}_{inf\cap}(N')
\end{aligned}$$

From the last equivalence we know that there exists $\alpha' \in \delta'^{\omega}$, with $pr_2(\alpha') = \sigma'$ such that $s'_0 \, |\alpha'\rangle_{N'}$ and $inf(pr_1(\alpha')) \cap F \neq \emptyset$.

Consider $\alpha$ obtained as follows: $\alpha(i) = (pr_1(\alpha'(i)), \sigma(i), pr_3(\alpha'(i)))$. Let us show that $\alpha(i) \in \delta$. We know that $(pr_1(\alpha'(i)), \sigma'(i), pr_3(\alpha'(i))) \in \delta'$. Moreover, $\sigma(i)$ can only be different from $\sigma'(i)$ in relation to $\{\pi_{i_1}, \ldots, \pi_{i_k}\}$. However, taking into account how we defined $\delta$ we do know that $(pr_1(\alpha'(i)), \sigma(i), pr_3(\alpha'(i))) \in \delta$. It is then

obvious to see that $s_0 \left|\alpha\right\rangle_N$ and that $inf(pr_1(\alpha)) \cap F \neq \emptyset$. And so we concluded one of the implications.

Assume now that $\sigma \in L^{\omega}_{inf\cap}(N)$. Then there exists $\alpha \in \delta^{\omega}$ such that $pr_2(\alpha) = \sigma$, $s_0 \left|\alpha\right\rangle_N$ and $inf(pr_1(\alpha)) \cap F \neq \emptyset$. Due to the way $N$ was constructed from $N'$, we know that for all $d \in \delta$ there exists $d' \in \delta'$ such that $pr_2(d')|_{\Pi - \{\pi_{i_1},\ldots,\pi_{i_j}\}} = pr_2(d)$, with the start and ending states equal in both $d$ and $d'$. This fact then implies that there exists $\alpha' \in \delta'^{\omega}$ such that $pr_2(\alpha')|_{\Pi - \{\pi_{i_1},\ldots,\pi_{i_k}\}} = pr_2(\alpha) = \sigma$ with $s'_0 \left|\alpha'\right\rangle_{N'}$. We can then conclude that there exists $\sigma'$, $\{\pi_{i_1}, \ldots, \pi_{i_k}\}$-equivalent to $\sigma$, such that $\sigma' \in L^{\omega}_{inf\cap}(N')$, which ends the proof of our implication. □

We will finally give an example [7] of a EQLTL formula that can express $(G_2 p)$.

**Example 3.4.1.** $(G_2 p) \equiv \exists p'(p' \wedge (X(\neg p')) \wedge (G(p' \Leftrightarrow (X(Xp')))) \wedge (G(p' \Rightarrow p))$.

# 4

# Discrete Event Systems

In this Chapter we will give some basic definitions of discrete event systems and supervisory control. The first section will present a short introduction to Discrete Event Systems and the second section will discuss supervisory control of a specific type of logical discrete event systems, the Petri net.

## 4.1 Initial concept

A discrete event system (DES) is a dynamical system that evolves through *time* as a response to events and only to events. The development of discrete event systems has its roots on differential equations problems, when the solution of the O.D.E. only assumes a discrete set of values. It is natural to think of different approaches to these kinds of differential equations, since Real Analysis is not quite well equipped to deal with these phenomena. These problems appear increasingly often on our digitalized world, as the number of different configurations in a computer memory is finite, and so the evolution of them through time constitutes a discrete event system as the configuration of the memory only changes as a response to instructions. Besides any computer application, DES are frequently used as models in manufacturing, and as we will show some examples, in robotic planning.

**Definition 4.1.1.** A discrete event system is composed of a discrete set $X$ of possible states and a finite set of events $E = \{e_1, e_2, \ldots, e_n\}$. At a given time, the DES is always in a certain state $x \in X$, which is all the information needed to characterize the system. The state of a DES can only be changed by the occurrence of an event $e \in E$ and these events will occur instantaneously and asynchronously.

The set $X$ is called the *state* set of the DES and the set $E$ is called the *event space* of the DES. For instance, the total possible configurations of the memory in a computer is the state set, and all events $e_{0,i}$ and $e_{1,i}$ which change the $i-th$ slot of memory to 0 or 1, respectively, can be seen as the event space.

There is a important distinction in discrete event systems. The distinction is based on what are the designer's

goals when modeling something by a DES. There are three main views on discrete event systems that have their distinction on how Time is viewed by the designer:

- *untimed or logical discrete event systems*, where the interest of the designer is the study of sequences of behaviours produced by the discrete event system. The time at which they occur is irrelevant for this study, only the order matters. There are two main ways of modeling this problem: using finite state automata or Petri nets.
- *deterministic timed discrete event systems* are used when not only the sequences of behaviours are important, but also if the instant at which they occur is important. The time between each type of events is deterministic, though. Both timed automata and timed Petri nets are used to model these DES.
- *stochastic timed discrete event systems* where the time between each of the events is given by a probability distribution. In this case, Markov chains and generalized semi-Markov processes are two of the formalisms used to study these problems.

**Example 4.1.1.** Consider the $n$ philosophers dining problem initially introduced in the form of computer tapes by Edsger Djisktra[6]. For those unfamiliar with the dining philosophers problem, it consists on $n$ philosophers sat at a round dining table, each with a plate in front of them. There are also $n$ Chinese chopsticks between all of them. The plates are filled with food. The philosophers are either thinking or eating and are unable to talk between them. In order for them to eat, they must grab the chopstick at each one's left and right. Only then do they stop thinking and can start eating. When they are full, they release the chopsticks, one at a time, and they start thinking again.

We can try to create a logical discrete event system modeling the situation. For reasons of space, we will focus on the problem with only three philosophers. There are three chopsticks and three philosophers. Each of the chopsticks can be either on the table or in the hand of the neighboring philosopher. If the philosophers have two chopsticks on their hands they are effectively eating, otherwise they are thinking. So let us try to compute all possible states:

- There are three chopsticks $CH = \{ch_i\}_{i=1}^3$ and three philosophers $SOPH = \{soph_i\}_{i=1}^3$. The chopstick $ch_i$ is at philosopher's $ch_i$ right.
- Every philosopher can be in four different states. The philosopher $soph_i$ can be in each of these $\{(empty, empty), (empty, ch_i), (ch_{i-1 mod\ 3}, empty), (ch_{i-1 mod\ 3}, ch_i)\}$, so a philosopher is characterized by the chopsticks in his hands. If at least one chopstick is not present on the hand of the philosopher, we say that the philosopher is thinking. Otherwise, he is eating.
- The state space is then a subset of $soph_1 \times soph_2 \times soph_3$, since there are impossible cases that have to be removed. Let $s \in soph_1 \times soph_2 \times soph_3$. Then $s = (s_1, s_2, s_3)$; if in all components of all $s_k$ there exists more than three $ch_j$, for some $j$, then we should reject $s$. Furthermore, if $ch_i = pr_2(s_i) = pr_1(s_{i+1 mod\ 3})$, for some $i$, we should also reject $s$.

We have defined the state space $X$ of the discrete event system. We shall now approach the event space:

- The philosopher $soph_i$ can release his right or left chopstick, respectively $ch_i$ and $ch_{i-1 mod 3}$. This act is represented as $release(soph_i, ch_i)$ or $release(soph_i, ch_{i-1 mod\ 3})$. Notice that the philosopher can only release a chopstick at a time.
- They can also grab the right or left chopstick; this is represented by $grab(soph_i, ch_i)$ or $grab(soph_i, ch_{i-1 mod\ 3})$. Again, they can only grab a chopstick at a time.

Finally, we have to discuss which events can actually change the state of the DES. Clearly, if a philosopher has not its right/left fork, he should not be able to release it and if he has it, he should not grab it. Moreover, he cannot grab the right/left fork if his right/left philosopher has it in his left/right hand.

The figure below represents the Petri net representation of our discrete event system. We will use the abbreviations $grab(soph_i, ch_j) \equiv_{abv} g_{i,j}$ and $release(soph_i, ch_j) \equiv_{abv} r_{i,j}$. We will just name some of the more important places.

Each of the three structures with four places represents the stance of a particular philosopher, whether he is eating, thinking, or with his right or left chopstick in his hands. According to the description of the problem, only if he has both chopsticks in his hand he is considered eating, otherwise he is thinking. The description given here is not, however, the typical formulation of the problem, as we will discuss.

We can now analyze this discrete event system using Petri net analysis tools. Let the Petri net represented here be known as $Soph3$.

If we want to focus on the discrete event system firing of events we should analyze the Petri net with language theoretic tools. Since we did not define any accepting goals, we will not be interested for now in $L_L(Soph3)$ or $L_G(Soph3)$. However, both $L_P(Soph3)$ and $L_T(Soph3)$ are well defined and it is easy to see that $L_T(Soph3) = \emptyset$, since there are no terminating markings reachable from the initial marking.

The language $L_P(Soph3)$ would then be the set of all possible sequences of events. We could then continue our analysis and conclude, for instance, that is a regular language, and so we actually could produce the regular expression that defines the language. More importantly, we could compare this language with the intended behaviour for our conceptual discrete event system and see if there are any substantial differences. If there were none, we could use the Petri net as a model of our system, performing either stochastic analysis as in [5, 1], or we could use our model as submodels of a greater discrete event system. Our goal in this work, as explained in the Introduction, is not to analyze the actual behaviours (stochastic or not) of the system but to allow the designer to produce more complex systems, fulfilling his goals, in a more time efficient manner.

Consider the following example:

**Example 4.1.2.** The model constructed earlier does not accurately describe the problem formulated by

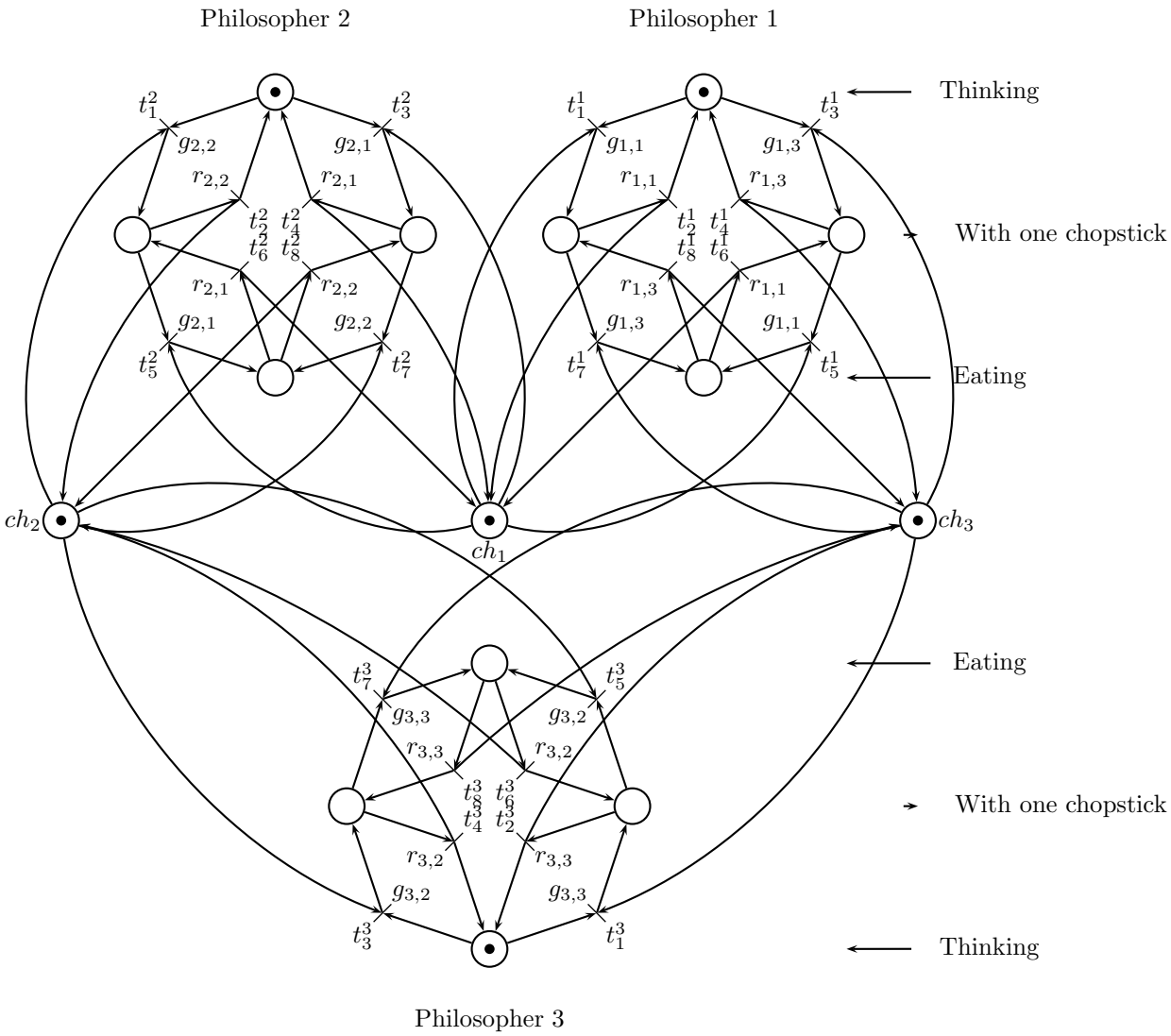


Figure 4.1: A Petri net version of the discrete event system that models the Three Dining Philosophers Problem as we stated it. The direction of the arrows are a reminder of what is the intended meaning of each place.

Dijkstra. While there are shared and limited resources, deadlock does not occur as it did in the original formulation. Analyzing $L_P(Soph3)$, it is easy to see that there are subsequences of the type $g_{i,i}r_{i,i}$ or $r_{i,i}g_{i,i}$ in those words. These subsequences of events are originated by the following subsequences of transitions: $g_{i,i}r_{i,i} = \sigma(t_1^i t_2^i) = \sigma(t_7^i t_8^i)$. In Dijkstra's formulation of the problem, if a philosopher is thinking, with no chopsticks in his hands, then if he grabs a chopstick, he should not be able to return it to the table, since he actually wants to eat. The inverse situation is also true: if the philosopher has eaten enough and puts down a chopstick, he should not be able to grab it right away.

If we wish to transform our initial model to better reflect Dijkstra formulation, we need to analyze carefully the global Petri net; the representation in Figure 4.2 is constructed from the earlier representation by discriminating the intermediate *one chopsticks* places, splitting them into *after thinking* and *after eating* places.

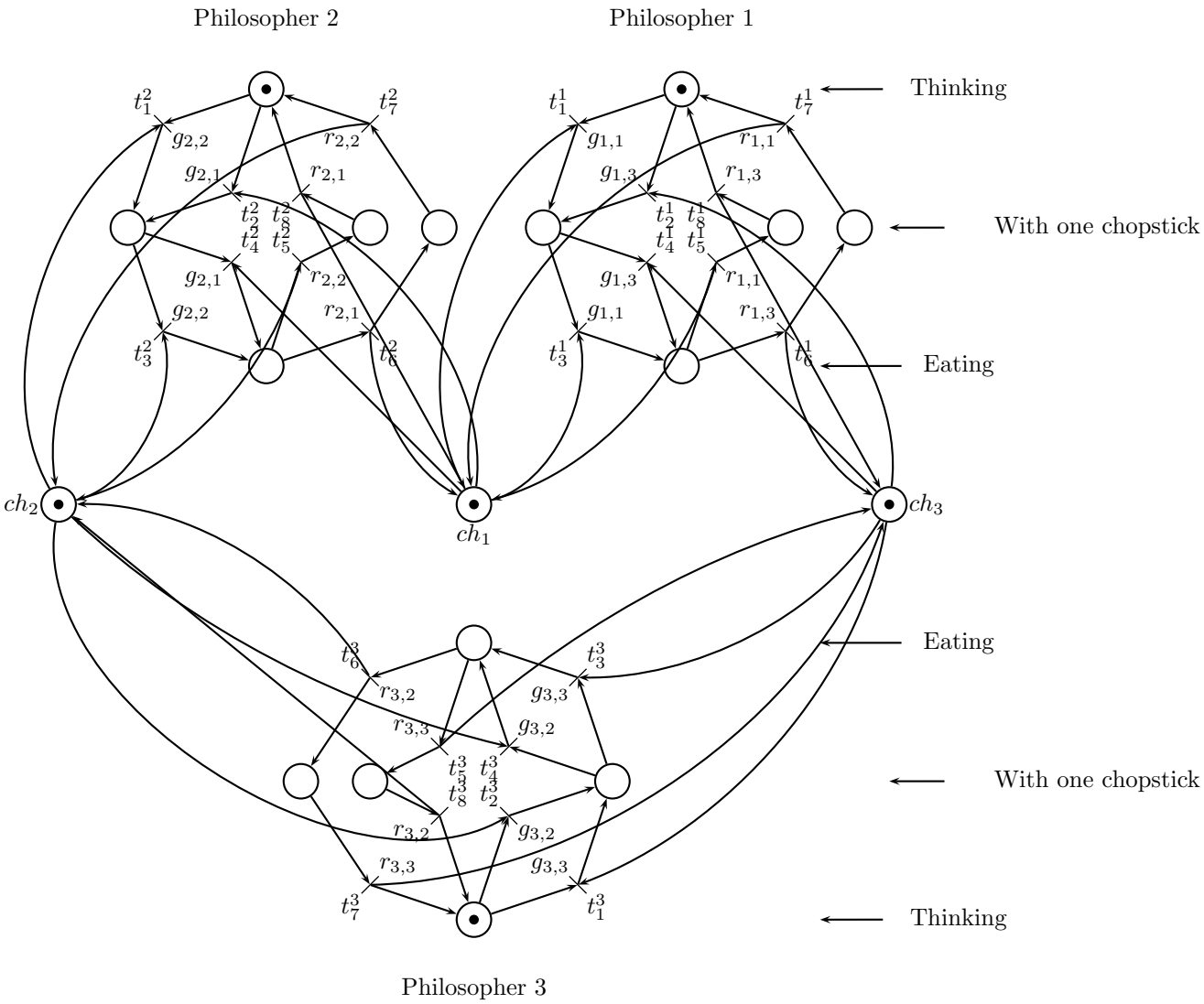


Figure 4.2: A Petri net version of the discrete event system that models the Three Dining Philosophers Problem as we stated it. The direction of the arrows are a reminder of what is the intended meaning of each place.

In this new version of the discrete event system and its representation, the Petri net $Soph3'$, deadlock can now occur. The only transition sequence generating the sequence of events $g_{1,1}.g_{2,2}.g_{3,3}$, beginning in the initial marking, leads to a terminating state, where no transitions are enabled. The new model now accurately describes Dijkstra's formulation of the problem.

There are various issues that should be addressed with this small but illustrating example. First of all, it is clear that in order to adapt a pre-existing DES system model to some additional conditions, we had to systematically analyze the whole Petri net model. It is then clear that, if the Petri net model was too big, even small alterations in the properties required of the DES could have an huge impact on the Petri net model, forcing us to spend valuable time not only analyzing the original net, but also adapting it to the new specifications. Even more importantly, we have no way of knowing if our work is bug-free. We can, of course, extensively test the system obtained, but even then we might only get a certain degree of trust in

our model.

These issues provide us the motivation for our next section.

Furthermore, we should also discuss the issue of determinism/non-determinism. A system is informally called *deterministic* if given a sequence of events and an initial state, we can identify without error the states passed by as the system is evolving in response to the given sequence. Consider now a finite state automaton modeling a certain discrete event system. We can design the automaton to be deterministic, representing exactly the intended modeled behaviour. But what would we gain from it? If we are only worried about having a model that decides whether a certain behaviour is intended or not, then we can certainly use a non-deterministic finite state automaton. However, if we wish to use a finite state automaton as a plan for a physical model, then we should really focus ourselves in deterministic FSA.

## 4.2 Supervisory control

As we saw earlier, we need to find a method that allows a designer to adapt pre-existing representations of a discrete event system in order to model related represention. Ideally, this method should also automatically certify the discrete event system model obtained, without us needing to extensively test it.

One of the strategies that fulfills these requirements, developed initially for finite state automata representations, is supervisory control. Given a discrete event system, the idea of supervisory control is to merge the DES with an entity, called the supervisor, such that the resulting structure exhibits desired behaviours. This idea, developed initially in the 80s by Ramadge and Wonham [32], was aimed at finite state automata, in particular deterministic finite state automata. As we know, all finite state automata can be transformed into deterministic ones, preserving the language over finite sequences, so the restriction is only relevant if the size of the FSA matters.

Unfortunately for general (not-bounded) Petri nets, supervisory control is not quite as straightforward as in the FSA case, as can be seen in [12]. However, we are in a interesting position: while much of the work in supervisory control is aimed at proving that a certain smaller language can be obtained from the system initial language, proving that there exists an adequate supervisor which in close loop with the system outputs the smaller language, in our case, we already have a Petri net obtained from a (EQ)LTL specification, we only need to test whether this Petri net is an adequate supervisor. Before discussing our approach however, we will introduce supervisory control in FSA, and present a completely different approach on how to use the concepts of supervisory control in Petri nets.

### 4.2.1 Supervisory control on finite state automata

Before we actually begin to introduce the definition of a supervisor, we have to introduce a concept that, as we will see, limits the power of a supervisor. Consider a computer program; clearly, we construct our program by allowing and restricting certain evolutions in the memory of our computer. We can effectively act upon the memory by restricting commands that would clean all the memory, for example. However, we cannot restrict an eventual power breakdown, or a memory failure, or some sort of electronic error, because those events are accidental in nature; we also cannot restrict an act from a user, for instance if the user decides to press a key, we can ignore the fact that he did press a key, but we cannot restrict the user action. These types of events are called uncontrollable events. The other ones, those that can be restricted are called controllable events; with this idea in mind, we are now ready to define a supervisor of a finite state automata:

**Definition 4.2.1.** Let $M = (S, \Sigma, \delta, s_0, F)$ be a finite state automaton. Let $\Sigma = \Sigma_{uc} \cup \Sigma_c$, with $\Sigma_{uc} \cap \Sigma_c = \emptyset$. $\Sigma_c$ is the set of controllable events, and $\Sigma_{uc}$ is the set of uncontrollable events. Then a *supervisor* of the finite state automaton $M$ is a function $Sp : L(M) \mapsto 2^{\Sigma}$. Furthermore, if $Sp$ obeys the following restriction, then it is called an admissible supervisor.

- $UAE_\sigma \subseteq Sp(\sigma)$, where $\sigma \in L(M)$ and $e \in UAE_\sigma$ iff there exists $\alpha \in \delta^*$ and $d \in \delta$ such that $pr_2(\alpha) = \sigma$, $s_0 \left|\alpha.d\right\rangle_M$ and $pr_2(d) = e \in \Sigma_{uc}$.

$UAE$ is an abbreviation for Uncontrollable Active Events.

Clearly, a supervisor is a function that given a certain system behaviour, outputs a subset of all possible events. The admissibility condition enforces that all uncontrollable events that could be fired after producing a certain system behaviour have to be allowed. This definition is clearly the same as in [33, 32] if we admit that $M$ is deterministic. Even if $M$ is not deterministic, we know that we can effectively construct a deterministic FSA $M'$ that exhibits the same behaviour. Using this definition we are ready to discuss what is effectively a supervised finite state automaton.

In common applications, the supervisor should be seen as an outside entity, separated from the system, but capable of analyzing the system behaviour. Intuitively, the supervisor knows what the system did earlier, and it will restrict what will the system do next, allowing a certain set of choices to the system.

**Definition 4.2.2.** Let $M = (S, \Sigma, \delta, s_0, F)$ be a finite state automaton and $Sp : L(M) \mapsto 2^{\Sigma}$ a supervisor of $N$. A *supervised finite state automaton* is an automaton $Sp/M = (S, \Sigma, \delta, s_0, F)$ but with a different evolution rule. A run $\alpha \in \delta^*$ in $Sp/M$ will be defined inductively below:

- if $\alpha = \lambda$ then $\alpha$ is a run in $Sp/M$,
- if $\alpha = d$, with $d \in \delta$, then $\alpha$ is a run in $Sp/M$ if $s_0 \left|d\right\rangle_M$ and $p_2(d) \in Sp(\lambda)$,

- if $\alpha = (q_0, a_0, q_1)\dots(q_{n-1}, a_{n-1}, q_n)$, then if $\alpha_1 = (q_0, a_0, q_1)\dots(q_{n-2}, a_{n-2}, q_{n-1})$ is a run in $Sp/M$, then $\alpha$ is a run in $Sp/M$ if $s_0 \left|\alpha\right\rangle_M$ and $a_{n-1} \in Sp(\sigma)$.

If $\alpha$ is a run on $Sp/M$ it will be represented as $s_0 \left|\alpha\right\rangle_{Sp/M}$. Finally, the language accepted by a supervised finite state automaton, represented by $L(Sp/M)$, is defined as follows:

$$L(Sp/M) = \left\{\sigma : \alpha \in \delta^*, pr_2(\alpha) = \sigma, s_0 \left|\alpha\right\rangle_{Sp/M}\right\}.$$

Note that there are no guaranties that this language is even computable. It all depends of the variety of behaviour allowed by the supervisor. In fact, the following theorem shows that for any language that satisfies a controllability condition, there exists a supervisor that transforms a finite state automaton language into the controllable language.

**Theorem 4.2.1.** *Let $M = (S, \Sigma, \delta, s_0, F)$ be a finite state automaton and $K \subseteq L(M)$ a non-empty language. Then there exists an admissible supervisor $Sp$ such that $L(Sp/M) = K$ iff $K$ is prefix-closed and satisfies the controllability condition:*

$$\overline{K}.\Sigma_{uc} \cap L(M) \subseteq \overline{K}.$$

This controllability condition is essential to understand what behaviours cannot be restricted by a supervisor. If a certain desired behaviour is in $K$, and so a certain prefix of it is in $\overline{K}$, then if an uncontrollable event can happen, then it cannot be restricted, and the goal should be attainable still. Many different structures can be seen as supervisors, but the most commonly used structure to represent a supervisor are finite state automata themselves. One can see a finite state automaton as a supervisor if we use a commonly used function defined by each FSA: the active events function, $\Gamma : S \mapsto 2^{\Sigma}$. This function outputs the set of active events on a given state. We can then adapt this $\Gamma$ function to be defined over the language $L(System)$. By using finite state automata as supervisors, we remain in the same class of languages; furthermore we are provided with an easy method to describe how the supervisor and the system interact.

**Proposition 4.2.1.** *Let $M_i = (S_i, \Sigma, \delta_i, s_{0,i}, F_i) \quad i = 1, 2$ be two finite state automata, and let $\Sigma = \Sigma_{uc} \cup \Sigma_c$. Let $M_2$ be a supervisor of $M_1$. Then $L(M_2/M_1) = L(M_1) \cap L(M_2)$.*

In the above proposition $M_2$ should be seen as a supervisor as discussed in the above paragraph. Now we should focus our attention into marked languages. After all, they represent the actual desired behaviours, and as such, we have to expect that we can still apply supervisor control successfully to marked languages. First, we need to define the accepted language of $M$ controlled by $Sp$.

**Definition 4.2.3.** Let $M = (S, \Sigma, \delta, s_0, F)$ be a finite state automaton and $Sp$ a supervisor of $M$. Then the *marked language of $M$ controlled by $Sp$*, represented as $L_c(Sp/M)$, is defined as follows:

$$L_c(Sp/M) = L(Sp/M) \cap L_m(M).$$

With this notion we can extend the theorem about the existence of a supervisor to marked languages.

**Theorem 4.2.2.** *Let $M = (S, \Sigma, \delta, s_0, F)$ be a finite state automaton and $K \subseteq L_m(M)$ a non-empty language. Then there exists a supervisor $Sp$ of $M$ such that $L_c(Sp/M) = K$ and $Sp/M$ is non-blocking iff $K$ is controllable and $\overline{K} \cap L_m(M) = K$.*

Clearly, if the supervisor is provided by a finite state automaton, then as we know $L(Sp/M) = L(Sp) \cap L(M)$, and so $L_c(Sp/M) = L(Sp) \cap L_m(M)$. Lacerda and Lima article [20] uses this approach and produces supervisors given by an LTL formula. We end up obtaining a system supervisioned by a LTL formula in the form of a finite state automaton; we can then use the model as a robot task plan, analyzing it further if necessary.

### 4.2.2 Supervisory control of Petri nets by restricting the reachability set

Supervisory control on Petri nets is a much more complex issue, due to the extra expressive power of the Petri nets; many of the properties required in the above propositions are hard to check. For instance, whether a Petri net is non-blocking is a non trivial issue; moreover, as can be seen in [12] it is impossible in the general case to trim a Petri net. Furthermore, while the above concept of a supervisor allow us to affect the behaviours exhibited, Petri nets have a very important component that is completely forgotten with the concept of supervisor we have introduced: markings. In some cases, we might be really interested to directly affect the reachability set of a Petri net, affecting indirectly the language the Petri net exhibits, clearly diverging from the approach that focuses on restricting active events, which directly affecs the Petri net language.

There are some techniques [19] that directly approach the problem of supervising a *deterministic* Petri net by regular languages, using generalized definitions to Petri nets originating in [33]. Our approach is different, as we do not require any specific property from our Petri net system and we also present, in order to further our goals, the definition of an extended supervisor, linking it directly with the controllability condition; finally, we reiterate that our main interest is to discover whether a certain Petri net can be used as a supervisor or not.

We will first present an example on how to supervise a Petri net if we wish to directly affect the reachability set; this technique enforces certain place invariants upon the original Petri net. We present it here, providing a simple example to not only illustrate the differences between this technique and the one we will discuss and use in the next chapter, but also because supervisory control on Petri nets should not, in our view, monopolize a single method; for different objectives we should choose the appropriate method.

This approach and the tutorial example were taken from [24].

We wish now to enforce a constraint onto the original Petri net. Consider $K$ a fixed natural number and the constraint $\mu_2 + K\mu_4 \leq K$. The act of enforcing of this constraint means that all reachable markings on the

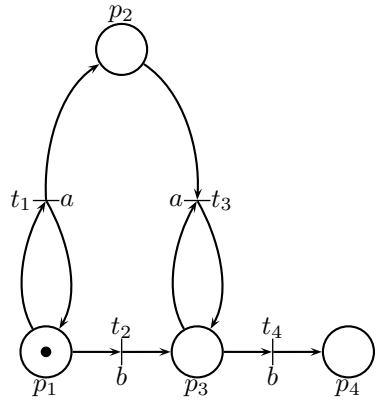


Figure 4.3: This Petri net $N$ is such that $L_T(N) = \left\{a^j b a^k b,\, 0 \leq k \leq j\right\}$, since the only terminating states are the reachable markings with one token in $\{p_4\}$.

obtained Petri net must respect $\mu_2 + K\mu_4 \leq K$. How can we synthesize a supervisor from this constraint?

In order to answer this question, we first need to assume the reader is familiar with the matrix notation for Petri nets. Even if he is not, we provide the incidence matrix $D_p$ and input and output matrix for our example:

$$D_p = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = D_p^+ - D_p^- = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The meaning of position $D_{p_{i,j}}$ is the following: if transition $t_j$ fires, the place $p_i$ gains $D_{p_{i,j}}$ tokens (if the number is negative, it loses). The basic idea now is to introduce an extra place for each of the constraints and to calculate the appropriate transitions input and output, in order to fulfill our place invariant. We also have to calculate the initial marking of these new places. Since we only have one constraint, we can rewrite it as follows:

$$L.\mu \leq K \iff \begin{bmatrix} 0 & 1 & 0 & K \end{bmatrix} \cdot \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{bmatrix} \leq K \iff \begin{bmatrix} 0 & 1 & 0 & K \end{bmatrix} \cdot \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{bmatrix} + \mu_c = K.$$

This allows us to calculate the initial marking in the controller place. To calculate the input and output transitions, notice that we wish the original Petri net merged with the controller places to obey the following equation:

$$LD_p + D_c = 0,$$

where $D_p$ is the incidence matrix of our original Petri net, $L$ is the constraint we wish to enforce, and $D_c$ is

matrix representing the input and output functions of the controller place(s). It is easy to see then that the Petri net represented in Figure 4.4, the merge of $D_p$ with $D_c$, with the correct initial marking obeying the restriction.

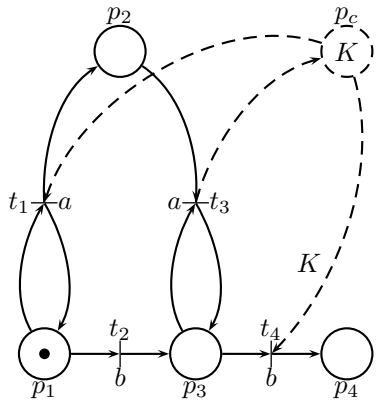


Figure 4.4: This Petri net $N_c$ now follows the restriction we specified: $\mu_2 + K\mu_4 \leq K$. The controller place(s), input and output functions are marked with dashed lines.

Clearly, the supervised Petri net T-Type language has dramatically changed. The T-Type language accepted by $N_c$ is now $\{a^k b a^k b,\ 0 \leq k \leq K\}$.

This small example, so small that it did not even introduce restrictions like uncontrollable or unobservable transitions and with only one specification, is nevertheless sufficient for our analysis. If we had wished to alter the T-Type language accepted by the system in order for the supervised version of it to be $L_T(N_c) = \{a^k b a^k b,\ 0 \leq k \leq K\}$, we had to look into the Petri net structure, searching for the condition that made impossible to enter a deadlocking state with tokens in $p_2$. Only then, after writing the specification, could we build the supervisor that would alter the language. Our reasoning to say that this method is not particularly tailored to affect a language accepted by a Petri net is two-fold:

- we had a transition behaviour problem, we transformed it into a place behaviour problem, restricting the reachability set. The change in the accepted language was simply a collateral happening.
- the transformation of a transition behaviour problem into a place behaviour problem is relatively unique to each Petri net. This means that with big Petri net models, if the problem is not *local*, it can be a very tedious job to find the correct restrictions to apply. Furthermore, the method itself does not guaranty that the restrictions are the right ones.

### 4.2.3 Supervisory control of Petri nets, as we propose it

We will now present our version of supervisory control applied to Petri nets. Our type of supervisory control focuses itself on the language, and its concepts are largely adapted from the ones introduced by [33, 32], as presented above. However, we do not intend to discuss whether there exists or not an admissible supervisor that, if used in closed loop with the system, would restrict the original Petri net as we specified. Our goal is quite different and although we will discuss at length the next observation in the next chapters, we will provide here the motivation for our particular interest in supervisory control.

We now have the tools to build a Petri net that is, in some way, a model of a QPLTL formula. We would like to use this Petri net $PN_\phi$ as supervisor of the original system $PN_{system}$. Clearly, if we are just interested in having a Petri net model whose accepting language (for a certain type of accepting condition) is the intersection between the language of the initial system and the language of the QPLTL formula represented in Petri net form, we just need to run the appropriate Petri net intersection algorithm.

However, if we wish to consider uncontrollable events, we need to discuss whether the desired language $L(PN_{system}) \cap L(PN_\phi)$ is controllable or not, since if it is not controllable we should think of another QPLTL formula that might express a similar property, as our desired one is not physically implementable. We need to find a sufficient condition such that we are ensured that the intersection of the language of the system with the specification given by the QPLTL is controllable; in that case, even if the Petri net representing the specification ($PN_\phi$) is not an admissible supervisor, nevertheless we can use $PN_\phi$ as a supervisor, since the only moments when $PN_\phi$ does not obey the admissibility restriction are when $\sigma \in L(PN_{system})$, $\sigma \notin L(PN_\phi)$, as we shall see. Note that these moments are irrelevant when considering the closed loop between the supervisor and the initial system, as the closed loop will impossibilitate behaviour $\sigma$ to happen.

**Definition 4.2.4.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net, with $\sigma : T \mapsto \Sigma$ a $\lambda$-free labeling function. Let $\Sigma = \Sigma_c \cup \Sigma_{uc}$, with $\Sigma_c \cap \Sigma_{uc} = \emptyset$. Again, $\Sigma_c$ is the set of controllable events, and $\Sigma_{uc}$ is the set of uncontrollable events. Then a *supervisor of* $PN$ is a function $Sp : L_P(PN) \mapsto 2^\Sigma$. If the following restriction is satisfied, we say that $Sp$ is an *admissible* supervisor.

- $UAE_\tau \subseteq Sp(\tau)$, where $\tau \in L_P(PN)$ and $e \in UAE_\tau$ iff there exists $\alpha \in T^*$ and $t \in T$ such that $\sigma(\alpha) = \tau$, $\mu_0 \left|\alpha.t\right\rangle_{PN}$ and $e = \sigma(t) \in \Sigma_{uc}$.

Again, the restriction enforces that all enabled transitions that are mapped by $\sigma$ into an uncontrollable event, if they could be reached by following $\tau \in \Sigma^*$ in the plant $PN$ have to be allowed. Following the work for finite state automata, we are ready to present a supervised Petri net.

**Definition 4.2.5.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net and $Sp : L_P(PN) \mapsto 2^\Sigma$ a supervisor of $PN$. A *supervised Petri net*, represented as $Sp/PN$, is a tuple $Sp/PN = (P, T, I, O, \mu_0, \sigma, F, Sp)$. A run $\alpha$ in this modified Petri net will be defined inductively below:

- $\alpha = \lambda$ is a run in this modified structure,
- if $\alpha = t$, with $t \in T$, then $\alpha$ is a run in $Sp/PN$ if $\mu_0 \left|t\right\rangle_{PN}$ and $\sigma(t) \in Sp(\lambda)$,
- if $\alpha = t_1 t_2 t_3 \dots t_n$ and if $\alpha_1 = t_1 t_2 t_3 \dots t_{n-1}$ is a run in $Sp/PN$ then $\alpha$ is a run in $Sp/PN$ if $\mu_0 \left|\alpha\right\rangle_{PN}$ and if $\sigma(t_n) \in Sp(\sigma(\alpha_1))$.

If $\alpha$ is a run in $Sp/PN$ it will be represented as $\mu_0 \left|\alpha\right\rangle_{Sp/PN}$. Finally, the languages accepted by a supervised Petri net language, represented as $L(Sp/PN)$ is defined as follows:

$$L(Sp/PN) = \left\{\sigma(\alpha) : \alpha \in T^*, \mu_0 \left|\alpha\right\rangle_{Sp/PN}\right\}.$$

We are now interested in studying the case when the supervisor is given by a Petri net; we are particularly interested in the case when the language of the supervisor is a regular language, and when the Petri net is trimmed. The following result is similar to the one presented in FSA.

**Proposition 4.2.2.** *Let $PN_i = (P_i, T_i, I_i, O_i, \mu_{0,i}, \sigma_i, F_i)$, $i = 1, 2$ be two Petri nets. Let $PN_2$ be a supervisor of $PN_1$. Then $L(PN_2/PN_1) = L_P(PN_1) \cap L_P(PN_2)$.*

Concerning marked languages, we have to specify the accepting condition, unlike FSA, which only had one natural accepting condition. We will use the G-Type accepting condition.

**Definition 4.2.6.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net and $Sp : L_P(PN) \mapsto 2^T$ a supervisor of $PN$. Then the G-Type language of $PN$ controlled by $Sp$, represented as $L_c(Sp/PN)$ is defined as $L_c(Sp/PN) = L(Sp/PN) \cap L_G(PN)$.

**Definition 4.2.7.** Let $PN = (P, T, I, O, \mu_0 \sigma, F)$ be a Petri net, and $\Sigma = \Sigma_{uc} \cup \Sigma_c$ be the partition in uncontrollable and controllable events. The language $K \subseteq L_P(PN)$ is called controllable if it obeys the following condition:

$$(\overline{K}.\Sigma_{uc}) \cap L_P(PN) \subseteq \overline{K}.$$

It is clear that all admissible supervisors $Sp$ of $PN$ are such that $L(Sp/PN)$ obeys the controllability condition. However, there are supervisors $Sp : L_P(PN) \mapsto 2^\Sigma$ such that $L_P(Sp/PN)$ is controllable but $Sp$ is not an admissible supervisor. We call such supervisors quasi-admissible. For instance, when considering the case when the supervisor $Sp$ is realized by a Petri net, it is possible that $L_P(Sp) \cap L_P(PN)$ is controllable, without $Sp$ being an admissible supervisor, because the controllability condition only requests that $\forall_{\sigma \in L_P(PN) \cap L_P(Sp)} \forall_{e \in \Sigma_{uc}} \sigma.e \in L_P(PN) \Rightarrow \sigma.e \in L_P(Sp)$, while the admissibility restriction forces the same condition for the bigger universe of $\sigma \in L_P(PN)$, unless of course if $L_P(Sp) \subseteq L_P(PN)$, which is not always true in the general case.

We wish now to establish criteria that allows us to decide whether a certain Petri net can be used as a quasi-admissible, or in other words, we want to know what is required from the Petri net initial system and the supervisor to ensure that the closed system is controllable.

**Proposition 4.2.3.** *Let* $PN = (P, T, I, O, \mu_0, \sigma, F)$ *be a Petri net, and let* $Sp = (P_{Sp}, T_{Sp}, I_{Sp}, O_{Sp}, \mu_{0,Sp}, \sigma_{Sp}, F_{Sp})$ *be a supervisor realized in a Petri net. Then the controllability condition for* $L_P(PN) \cap L_P(Sp) \neq \emptyset$ *is equivalent to* $\forall_{\sigma \in L_P(PN) \cap L_P(Sp)} \forall_{e \in \Sigma_{uc}} \sigma.e \in L_P(PN) \Rightarrow \sigma.e \in L_P(Sp)$. *Furthermore if* $L_P(Sp)$ *is a regular language, there exists an algorithm to decide if the controllability condition is verified.*

**Proof.**

$$(\overline{L_P(PN) \cap L_P(Sp)}\Sigma_{uc}) \cap L_P(PN) \subseteq \overline{L_P(PN) \cap L_P(Sp)} \Leftrightarrow$$

$$(L_P(PN) \cap L_P(Sp)\Sigma_{uc}) \cap L_P(PN) \subseteq L_P(PN) \cap L_P(Sp) \Leftrightarrow$$

$$\forall_{\sigma \in L_P(PN) \cap L_P(Sp)} \forall_{s \in \Sigma_{uc}} \quad \sigma.s \in L_P(PN) \Rightarrow \sigma.s \in L_P(PN) \cap L_P(Sp) \Leftrightarrow$$

$$\forall_{\sigma \in L_P(PN) \cap L_P(Sp)} \forall_{s \in \Sigma_{uc}} \quad \sigma.s \in L_P(PN) \Rightarrow \sigma.s \in L_P(Sp).$$

Assume now that $L_P(Sp)$ is a regular language. The complement of a regular language is also a regular language, although the finite state automaton that represents it is given by the powerset construction. We know then that there exists a finite state automaton $N$ such that $L_P(Sp)^c = L_m(N)$. Then, there exists a Petri net $N'$ such that $L_L(N') = L_m(N)$ (we could also use G-Type languages).

Now, $L_P(Sp) \cap L_P(PN)$ is a Petri net P-Type language, $(L_P(Sp) \cap L_P(PN))\Sigma_{uc}$ is also a Petri net P-Type language, and therefore, $(L_P(Sp) \cap L_P(PN))\Sigma_{uc} \cap L_P(PN)$ is also a Petri net P-Type language. We can then assume there exists a Petri net $PN'$ such that $L_P(PN') = (L_P(Sp) \cap L_P(PN))\Sigma_{uc} \cap L_P(PN)$.

Using the information from both earlier paragraphs we can then assume there exists a Petri net $PN''$ such that $L_L(PN'') = L_L(N') \cap L_P(PN')$. Finally, an algorithm that decides whether the controllability condition is verified or not just has to check whether the Petri net L-Type language $L_L(PN'')$ is empty or not. If it is empty, the controllability condition is satisfied. If it is not empty, then there is a breach of the controllability condition and therefore $Sp$ is not a quasi-admissible, much less an admissible supervisor.

□

**Proposition 4.2.4.** *Let* $PN = (P, T, I, O, \mu_0, \sigma, F)$ *be a Petri net, the system, and let* $Sp = (P_{Sp}, T_{Sp}, I_{Sp}, O_{Sp}, \mu_{0,Sp}, \sigma_{Sp}, F_{Sp})$ *be a supervisor, realized in a Petri net. If the conditions described below are met,* $L_P(Sp) \cap L_G(PN)$ *is controllable if and only if* $L_P(Sp) \cap L_P(PN)$ *is controllable. The conditions are:*

- $PN$ *is non-blocking,* $\forall_{\sigma \in L_P(PN)} \exists_{\sigma' \in \Sigma^*} \quad \sigma\sigma' \in L_G(PN)$,
- $L_P(Sp)$ *and* $L_G(PN)$ *are non-conflicting,* $L_P(Sp) \cap \overline{L_G(PN)} = \overline{L_P(Sp) \cap L_G(PN)}$.

**Proof.** We will just prove that the controllability conditions are equivalent if those assumptions are valid.

$$
\begin{aligned}
&[(L_P(Sp) \cap L_P(PN))\Sigma_{uc}] \cap L_P(PN) \subseteq L_P(Sp) \cap L_P(PN) \Leftrightarrow \\
\Leftrightarrow &[(L_P(Sp) \cap \overline{L_G(PN)})\Sigma_{uc}] \cap L_P(PN) \subseteq L_P(Sp) \cap \overline{L_G(PN)} \Leftrightarrow \\
\Leftrightarrow &(\overline{L_P(Sp) \cap L_G(PN)}\Sigma_{uc}) \cap L_P(PN) \subseteq \overline{L_P(Sp) \cap L_G(PN)}.
\end{aligned}
$$

The initial expression represents the controllability condition of $L_P(Sp) \cap L_P(PN)$; the equivalence between the first and second expression is justified due to the fact that $PN$ is non-blocking, which allows us to conclude that $L_P(PN) = \overline{L_G(PN)}$; the equivalence between the second and third expressions is a direct use of the non-conflicting property, and the third expression is, of course, the controllability condition for $L_P(Sp) \cap L_G(PN)$. □

Note that, so far when dealing with the marked language of controlled systems, never have we used the fact that the sequences marked are obtained using the G-Type accepting condition; in fact the results would be exactly the same if we used the L-Type accepting condition, although the intersection of a P-Type language and a L-Type language is harder to compute than the intersection between a G-Type language and a P-Type language.

Regarding the conditions required to assure controllability, the non-blocking condition of the system is a Petri net specific condition, as we can always assume that a certain FSA is trimmed, and therefore non-blocking, an assumption that we can not always fulfill in Petri net. The non-conflicting condition is a common request in supervisory control of finite state automata, whenever we wish to use two supervisors on the same system.

As we stand, we are equipped with sufficient conditions that allow us to ensure that if we intersect a P/G-Type system language and a P-Type supervisor language we obtain a controllable closed system, assuring us that $Sp$ is a quasi-admissible supervisor, and perhaps even an admissible supervisor. Either way, we are assured that the Petri net $Sp$ will not disallow accessible (by both $Sp$ and the system) uncontrollable events. However, we are interested in something more, not usually contemplated in the classical formulation of supervisory control. We wish to allow our "supervisor" to mark sequences. Why do we need this extra feature? Although the formal justification will only be shown in the next chapter, we will provide the sole motivation for it: consider we write an $LTL$ formula $\phi$ representing a desired specification. If we look at the LTL to Büchi automata translation, we see that we first translate $\phi$ into a generalized Büchi automaton ($GBA_\phi$), and only afterwards do we translate it to a proper Büchi automata($BA_\phi$). Carefully looking into the generalized Büchi automata to Büchi automata translation we are assured that if $GBA_\phi$ had $n$ types of sets of accepting states ($\#\mathcal{F} = n$), and if we make the trimmed Büchi automaton $BAtr_\phi$ only accept finite sequences, then $\sigma \in L_m(BAtr_\phi)$ belongs to the language if and only if $\sigma$ passes at least once for each of set of accepting states, meaning that only with the last transition did we actually visit all necessary conditions. If we could use $L_m(BAtr_\phi)$ as a "supervisor", we would be ensured that the system behaviour would only accept a sequence of events when all the $n$ conditions of the "supervisor" created from the specification $\phi$ were reached.

We then need a way to define what is this "supervisor" we are writing about, what are its properties, and what are its relations with controllable and uncontrollable events, because, again, as in the beginning of the exposition of our approach to supervisory control on Petri nets, if we do not wish to consider the restrictions brought upon us by uncontrollable events, then we can directly use the intersection of the G/P-Type language of the system with the G-Type language of a Petri net, created by a specification. Note, however, that throughout our discussion, a "supervisor" is always seen as a Petri net, we are not interested in other systems.

The simplest way of extending the definition of a supervisor in Petri net form is to state that $L_P(Sp)$ or $L_G(Sp)$ is an *extended supervisor* of a given Petri net system $PN$ if $L_P(Sp) \cap L_P(PN)$ is controllable, which is the case of a system obeying Proposition 4.2.3, or if $L_P(Sp) \cap L_G(PN)$ is controllable, which is the case of a system satisfying the three assumptions of Proposition 4.2.4, or if $L_G(Sp) \cap L_P(PN)$ is controllable and finally $L_G(Sp) \cap L_G(PN)$ is controllable. The first two cases are what we already called quasi-admissible supervisors; the extension of the definition of a supervisor of a Petri net in Petri net form is simply whether the language given by the intersection of the language of the system with the language of the potential extended supervisor is controllable or not. We will formally define an extended supervisor:

**Definition 4.2.8.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net and $Sp = (P_{Sp}, T_{Sp}, I_{Sp}, O_{Sp}, \mu_{0,Sp}, \sigma_{Sp}, F_{Sp})$ another Petri net, potential extended supervisor. Consider $\Sigma = \Sigma_{uc} \cup \Sigma_c$ the usual partition of events. Consider $\Gamma = \{P, G\}$ the two accepting criteria. $L_{\gamma_1}(Sp), \gamma_1 \in \Gamma$ is called an extended supervisor of $L_{\gamma_2}(PN), \gamma_2 \in \Gamma$ if $L_{\gamma_1}(Sp) \cap L_{\gamma_2}(PN)$ obeys the controllability condition:

$$[(\overline{L_{\gamma_1}(Sp) \cap L_{\gamma_2}(PN)})\Sigma_{uc}] \cap L_P(PN) \subseteq \overline{L_{\gamma_1}(Sp) \cap L_{\gamma_2}(PN)}.$$

If the languages can be understood from context, we can simply say that $Sp$ is an extended supervisor.

Note that this new condition envelops the classical definition of supervisors, absorbing also the semi-formal definition given already here of quasi-admissible supervisors. Furthermore, we are now ready to search for sufficient conditions that would ensure us controllability.

**Proposition 4.2.5.** *Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net, the system, and let $Sp = (P_{Sp}, T_{Sp}, I_{Sp}, O_{Sp}, \mu_{0,Sp}, \sigma_{Sp}, F_{Sp})$ be a Petri net, a possible extended supervisor. If the conditions described below are met, $L_G(Sp) \cap L_P(PN)$ is controllable if and only if $L_P(Sp) \cap L_P(PN)$ is controllable. The conditions are:*

- *$Sp$ is non-blocking, $\forall_{\sigma \in L_P(Sp)} \exists_{\sigma' \in \Sigma^*} \quad \sigma\sigma' \in L_G(Sp)$,*
- *$L_G(Sp)$ and $L_P(PN)$ are non-conflicting, $L_P(PN) \cap \overline{L_G(Sp)} = \overline{L_P(PN) \cap L_G(Sp)}$.*

**Proof.** We will just prove that the controllability conditions are equivalent if those assumptions are valid.

$$\begin{aligned}
&[(L_P(Sp) \cap L_P(PN))\Sigma_{uc}] \cap L_P(PN) \subseteq L_P(Sp) \cap L_P(PN) \Leftrightarrow \\
\Leftrightarrow &[(L_P(PN) \cap \overline{L_G(Sp)})\Sigma_{uc}] \cap L_P(PN) \subseteq L_P(PN) \cap \overline{L_G(Sp)} \Leftrightarrow \\
\Leftrightarrow &(\overline{L_P(PN) \cap L_G(Sp)}\Sigma_{uc}) \cap L_P(PN) \subseteq \overline{L_P(PN) \cap L_G(Sp)}.
\end{aligned}$$

The initial expression represents the controllability condition of $L_P(Sp) \cap L_P(PN)$; the equivalence between the first and second expression is justified due to the fact that $Sp$ is non-blocking, which allows us to conclude that $L_P(Sp) = \overline{L_G(Sp)}$; the equivalence between the second and third expressions is a direct use of the non-conflicting property, and the third expression is, of course, the controllability condition for $L_P(PN) \cap L_G(Sp)$. □

Finally we present the case when both the extended supervisor and system mark strings.

**Proposition 4.2.6.** *Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net, the system, and let $Sp = (P_{Sp}, T_{Sp}, I_{Sp}, O_{Sp}, \mu_{0,Sp}, \sigma_{Sp}, F_{Sp})$ be a Petri net, a possible extended supervisor. If the conditions described below are met, $L_G(Sp) \cap L_G(PN)$ is controllable if and only if $L_P(Sp) \cap L_P(PN)$ is controllable. The conditions are:*

- *$Sp$ is non-blocking, $\forall_{\sigma \in L_P(Sp)} \exists_{\sigma' \in \Sigma^*} \quad \sigma\sigma' \in L_G(Sp)$,*
- *$PN$ is non-blocking, $\forall_{\sigma \in L_P(PN)} \exists_{\sigma' \in \Sigma^*} \quad \sigma\sigma' \in L_G(PN)$,*
- *$L_G(Sp)$ and $L_G(PN)$ are non-conflicting, $\overline{L_G(PN)} \cap \overline{L_G(Sp)} = \overline{L_G(PN) \cap L_G(Sp)}$.*

**Proof.** We will just prove that the controllability conditions are equivalent if those assumptions are valid.

$$\begin{aligned}
&[(L_P(Sp) \cap L_P(PN))\Sigma_{uc}] \cap L_P(PN) \subseteq L_P(Sp) \cap L_P(PN) \Leftrightarrow \\
\Leftrightarrow &[(\overline{L_G(PN)} \cap \overline{L_G(Sp)})\Sigma_{uc}] \cap L_P(PN) \subseteq \overline{L_G(PN)} \cap \overline{L_G(Sp)} \Leftrightarrow \\
\Leftrightarrow &(\overline{L_G(PN) \cap L_G(Sp)}\Sigma_{uc}) \cap L_P(PN) \subseteq \overline{L_G(PN) \cap L_G(Sp)}.
\end{aligned}$$

The initial expression represents the controllability condition of $L_P(Sp) \cap L_P(PN)$; the equivalence between the first and second expression is justified due to the fact that both $Sp$ and $PN$ are non-blocking, which allows us to conclude that $L_P(Sp) = \overline{L_G(Sp)}$ and $L_P(PN) = \overline{L_G(PN)}$; the equivalence between the second and third expressions is a direct use of the non-conflicting property, and the third expression is, of course, the controllability condition for $L_G(PN) \cap L_G(Sp)$. □

If we are considering $L_G(PN)$ to be a regular language, (consequently $\overline{L_G(PN)}$ is a regular language, too), as we have seen, we can effectively test if $L_P(PN) \cap L_P(Sp)$ is controllable or not. The non-blocking property is more difficult to prove, and the author does not know any algorithm to decide the non-blocking property for all Petri nets. Finally, the non-conflicting property is also a difficult property to verify. Still, if the designer

is really interested in establishing controllability, he has two choices after establishing controllability of the $L_P(PN) \cap L_P(Sp)$: either he is careful on the design of the net, and ensures that the sufficient conditions are verified, or he can test-run all strings until some arbitrary length; that way he would ensure controllability up to a certain sequence length.

# Part II

# Supervisory Control of Petri nets using Temporal Logic

# 5

# Proposed Methodology for Supervisory Control of Petri Nets

In this chapter we will try to bind all theory previously discussed into a working method that allows a designer of a Petri net, which might represent a robotic task, to enforce event based specifications into his Petri net. This chapter will be divided into two sections: on the first section we will briefly present the method and in the second section we will discuss each step carefully.

## 5.1 Methodology for Supervisory Control of Petri Nets using temporal Logic specifications

We will now enumerate the basic steps of our proposed methodology.

**Step 5.1.1** (Design of the initial system)**.** The designer creates a Petri net $PN_{system}$, modeling an initial system. The design of the Petri net should already have in mind the future specifications that will be forced upon it, and what type of supervision he wishes to use, in order to fulfill the sufficient controllability conditions if he desires, right at the start.

**Step 5.1.2** (Design of the EQLTL specification)**.** The designer chooses a formula of $EQLTL(\Sigma \cup Q)$, if necessary enriched with a set of free propositional symbols $Q = \{\pi_1, \pi_2, \ldots, \pi_n\}$. Thereafter this formula shall be called $\phi$, also known as specification. Furthermore, we also assume that $\phi = \exists\pi_1 \ldots \exists\pi_n\psi$ for $\psi \in LTL(\Sigma)$.

**Step 5.1.3** (Translation from EQLTL to generalized Büchi automaton)**.** Let $\phi \in EQLTL(\Sigma \cup Q)$ be our specification. Using Theorem 3.3.1 and Theorem 3.4.3 we obtain a generalized Büchi automaton $GBA_\phi$ such that $L_{EQLTL}(\phi) = L^{\omega}_{inf\cap}(GBA_\phi)$.

**Step 5.1.4** (Transforming $GBA_\phi$)**.** The first transformation is only performed if $\phi \in EQLTL(\Sigma \cup Q) - LTL(\Sigma)$. If so, consider the generalized Büchi automaton $GBA'_\phi = (S, 2^\Sigma, \delta', s_0, \mathcal{F})$ obtained from $GBA_\phi =$

$(S, 2^{\Sigma \cup P}, \delta, s_0, \mathcal{F})$ using the following transformation: $d \in \delta$ iff $(pr_1(d), pr_2(d)|_{2^\Sigma}, pr_3(d)) \in \delta'$. If $\phi \notin EQLTL(\Sigma \cup P) - LTL(\Sigma)$, then set $GBA'_\phi = GBA_\phi$.

The second transformation is quite simple: consider $GBA''_\phi = (S, \Sigma, \delta', s_0, F)$ obtained from $GBA'_\phi = (S, 2^\Sigma, \delta, s_0, F)$ where $d' \in \delta'$ iff $d \in \delta$ and $\#pr_2(d) = 1$.

**Step 5.1.5** (Transforming the generalized Büchi automaton into a trimmed Büchi automaton)**.** The first of two transformations is simply the application of the algorithm described in Proposition 1.2.1 or a more optimized version of it. The second transformation is the trimming operation of the Büchi automaton, using Definition 1.2.3.

**Step 5.1.6** (Transforming the Büchi automaton into a Petri net)**.** In order to transform $BA'_\phi$ into a Petri net, the designer should use the transformation described in Proposition 2.2.4.

**Step 5.1.7** (Intersecting $PN_\phi/\overline{PN_\phi}$ and $PN_{system}$ and testing the output)**.** The designer should now test his new system for emptiness; he can test emptiness whether the intersection of $PN_{system}$ and $PN'_\phi$ is a P-Type Petri net language or it is a G-Type Petri net language, although the latter case can only be tested using the reachability algorithm, which as discussed before is EXPSPACE-hard.

**Step 5.1.8** (Verification of controllability conditions)**.** The designer has now obtained a Petri net $PN_{Goal}$, whose P-Type language is equal to $L_P(PN_{Goal}) = L_P(PN_{system}) \cap L_P(PN'_\phi)$, or he has obtained a Petri net $PN_{Goal}$ whose G-Type language is equal to $L_G(PN_{Goal}) = L_G(PN_{system}) \cap L_P(PN'_\phi)$ or $L_G(PN_{Goal}) = L_P(PN_{system}) \cap L_G(PN'_\phi)$ or even $L_G(PN_{Goal}) = L_G(PN_{system}) \cap L_G(PN'_\phi)$, depending on what was the supervision problem he approached. In order to check whether $L_P(PN_{Goal})$ is controllable, the designer should run the algorithm described in Proposition 4.2.3. In order to check whether $L_G(PN_{Goal})$ is controllable, he must always check if $L_P(PN_{system}) \cap L_P(PN'_\phi)$ is controllable; he then has to check the specific conditions for his case; he can already count on $PN'_\phi$ being non-blocking.

## 5.2 In-depth analysis of each step

In this section we will show why does the method work, how well does it work, and we will give some hints on how to achieve better results with it.

In the first step, Step 5.1.1, we begin by defining our initial system $PN_{system}$ and what type of supervision is going to used. If the designer intends to use the controllability theorems from Chapter 4, he should design the Petri net in a way such that the controllability conditions are easy to check. Moreover, we will list the four types of supervision, providing hints on how can they be used effectively:

- He wishes to use the P-Type language of the system, and he wishes to use only the P-Type language of the supervisor. This is a very useful type of Supervision, restricting some of the possible sequences of events of the system according to the property specified; for instance, in the Three Dining Philosophers

Problem it can be used to prevent a thinking Philosopher who intends to eat and that has already grabbed a chopstick from releasing the grabbed chopstick before he grabs the other chopstick. The controllability conditions are described in 4.2.3.

- He wishes to use the G-Type language of the system, and he wishes to use only the P-Type language of the supervisor. This is also an useful type of supervision, allowing us to use objectives in the design of our system, using the supervisor to eliminate some unneeded subsequences of events. If he wishes to assure controllability of the final result he should take notice of Proposition 4.2.4. One of the sufficient conditions is unrelated to the supervisor, and so if he can assure that the system is non-blocking he can then try to fulfill the other conditions which will assure him controllability.
- He wishes to use the P-Type language of the system, and he wishes to use the G-Type language of the supervisor. This can be helpful in establishing goals that would otherwise be difficult to express with the G-Type language of the system. This time, there is nothing the designer could do right away to assure controllability, assuming he is using Proposition 4.2.5 with the final result, since our method will produce non-blocking supervisors.
- He wishes to use both G-Type languages. This type of supervision allows us to think of the objectives of both the system and the supervisor, while restricting some of the unwanted sequence of events. The controllability conditions will be very hard to prove, although if he can prove that the system is non-blocking, he only has to prove the non-conflicting property, since as we have already mentioned the supervisor is non-blocking by construction.

The next step, Step 5.1.2, definitely linked with the step already described, is the EQLTL expression of the property the Petri net designer wishes to enforce. Our properties will be based on events, not on transitions. This means that the set of propositional symbols, if the designer wishes to only use LTL, is going to be $\Sigma$, not $T$. The reason for this choice is quite simple, and although we could write LTL formulas based on both transitions and events, we will let that small modification to be formalized in a later project. Since the intersection algorithm of the system and the specification is based on events, and not on transitions, and since both the system and the supervisor can be nondeterministic, we might end up not fulfilling our restriction. For instance, if our restriction is $\phi = G(t_1 \Rightarrow (X(\neg t_1)))$ and our system has three transitions $t_1, t_2, t_3$, labeled as $\sigma(t_1) = \sigma(t_2) = a, \sigma(t_3) = b$, then if $\alpha \in T^\omega$ has a subsequence like $t_1t_1$ then $\alpha \notin L_{LTL}(\phi)$, but since the intersection is based on events, $aa$ might be a valid sequence on the P-Type language of the final system, and yet, $aa$ can be performed by $t_1t_1$. Moreover, it makes more sense that the outside system, the specification, can only act upon the expressed behaviour of the system, which are sequences of events, not sequences of transitions.

If the user wishes to use EQLTL to better express some desired property, he can enrich the set of propositional symbols with the symbols needed for quantification.

The designer should be aware that the fact he is working with discrete event systems implies that many formulas are completely obsolete. One of the characteristics inherent to discrete event system is the fact

that events occur instantaneously and asynchronously, which means that for instance $\phi = (a \wedge b)$ will never be fulfilled. He also should be careful that if he is aiming to assure controllability of the desired behaviours, many formulas will discard uncontrollable events almost as a side effect. For instance, a formula such as $\phi = G(a \implies (X(b)))$ may not lead to uncontrollable systems, but it probably will. If there is an uncontrollable active event besides $b$ after $a$ fires, we will not obtain a controllable language. It might be wise to switch to a *weaker* version of $\phi$ if we are not sure that the uncontrollable events are disabled, for instance $\phi' = G(a \implies (X((c \vee d \vee e \ldots)Ub)))$, where $c, d, e, \ldots$ are the uncontrollable events, assuring that $b$ occurs as soon as it can.

The next step, Step 5.1.3 is the translation of the EQLTL specification $\phi$ into a generalized Büchi automaton. This step is quite difficult computationally speaking. We will obtain a generalized Büchi automaton $GBA_\phi$ such that $L^\omega_{\forall inf\cap}(GBA_\phi) = L_{EQLTL}(\phi)$. If $\phi \in LTL(\Sigma)$, there are quite good algorithms. As we already mentioned, [11, 38] are good introductions to fast translation algorithms. Note that although Theorem 3.4.3 is stated as an EQLTL-to-Büchi automata translation, it can easily be restated for an EQLTL to generalized Büchi automata translation; in fact we never directly used the accepting condition in the proof of the theorem. There exists here only one small difficulty: Theorem 3.3.1 uses a set of initial states, a feature that is not contemplated neither on our definition of generalized Büchi automata nor on our definition of finite state automata. Fortunately, generalized Büchi automata, like finite state automata with Büchi accepting criterion are closed for union, which allows us to eliminate this small problem.

The following step, 5.1.4, is the transformation of the generalized Büchi automaton. The first transformation described, only applied if $\phi \in EQLTL(\Sigma \cup Q) - LTL(\Sigma)$, allows us to say that $\sigma \in L^\omega_{\forall inf\cap}(GBA_\phi)$ iff $\sigma|_{2^\Sigma} \in L^\omega_{\forall inf\cap}(GBA'_\phi)$. We can conclude two facts after applying this transformation:

- if $\sigma \in L^\omega_{\forall inf\cap}(GBA'_\phi)$ then $\sigma \models \phi$,
- if $\sigma \models \phi$, then $\sigma|_{2^\Sigma} \in L^\omega_{\forall inf\cap}(GBA'_\phi)$.

The second transformation heavily restricts the relation set $\delta$ outputting $GBA''_\phi$, although we still obtain $L^\omega_{\forall inf\cap}(GBA''_\phi) \subseteq L^\omega_{\forall inf\cap}(GBA'_\phi)$, therefore, if $\sigma \in L^\omega_{\forall inf\cap}(GBA''_\phi)$ then $\sigma \in L^\omega_{\forall inf\cap}(GBA'_\phi)$ and then $\sigma \models \phi$

We now have to translate the generalized Büchi automaton obtained into a Büchi automaton, which corresponds to Step 5.1.5. We will obtain a finite state automaton $BA_\phi$ such that $L^\omega_{inf\cap}(BA_\phi) = L^\omega_{\forall inf\cap}(GBA''_\phi)$. Another important property related to the language of finite sequences of $BA_\phi$ that originates in the transformation algorithm is fairly interesting. When we defined the final states in the transformation of Proposition 1.2.1, we have set $F = F_k \times \{k\}$, where $F_k$ is the last member of $\mathcal{F}$ in $GBA''_\phi$ and $k = \#\mathcal{F}$. Consider exactly the similar finite state automata $BA^i_\phi$ where the set of accepting states is substituted by $F_i \times \{i\}$. It is clear, from the proof of the proposition, that $L^\omega_{inf\cap}(BA^i_\phi)$ are all equal. However, an important difference arises when considering $L_m(BA^i_\phi)$.

Let $L^\omega_{inf\cap}(BA_\phi) \neq \emptyset$ and let $\sigma \in L_m(BA_\phi = BA^k_\phi)$. Then, for all $i = 1, \ldots, k-1$ there exists a prefix of $\sigma$,

named $\sigma'_i$, such that $\sigma'_i \in L_m(BA^i_\phi)$. This effectively means that if $\sigma \in L_m(BA_\phi)$ all accepting conditions originated from $\mathcal{F}$ and ultimately from $\phi$ are fulfilled. We will reference this property as *completely fulfilling*.

The second transformation is the trimming operation of the Büchi automaton, which according to Proposition 1.2.2 allows us to obtain a finite state automaton $BA'_\phi$ such that $L^\omega_{inf\cap}(BA'_\phi) = L^\omega_{inf\cap}(BA_\phi)$, *completely fulfilling*, and with the extra property described in Proposition 1.2.3. Furthermore, $BA'_\phi$ is also non-blocking, because it was trimmed.

The step where we translate the finite state automaton into a Petri net, Step 5.1.6, is quite straightforward. We end up obtaining a Petri net $PN_\phi$ such that $L^\omega_{inf\cap}(PN_\phi) = L^\omega_{inf\cap}(BA'_\phi)$, $PN_\phi$ is *completely fulfilling* in respect with $\phi$, and if $L^\omega_{inf\cap}(PN_\phi) \neq \emptyset$, then if $\sigma \in L_G(PN_\phi)$ then there exists $\sigma' \in \Sigma^\omega$ such that $\sigma\sigma' \in L^\omega_{inf\cap}(PN_\phi)$; it is also non-blocking: $\overline{L_G(PN_\phi)} = L_P(PN_\phi)$.

In this phase the designer has transformed its original desired property in his natural language into a Petri net, $PN_\phi$ that expresses almost the desired property. He now can intersect the languages corresponding to the type of supervision he wants, but before he can concern himself with controllability, he should test the intersection obtained, which leads us to Step 5.1.7.

The designer possesses a Petri net which we will call $PN'_\phi$ that expresses an appropriate modification of the EQLTL formula specification. He is interested, however, in considering the finite sequences accepted, or fireable by $PN'_\phi$. Whether he is interested in $L_P(PN'_\phi)$ or $L_G(PN'_\phi)$ depends on what type of supervision he is looking for. Nevertheless, due to the properties described in the above steps, he know nows that if $\sigma \in L_P(PN'_\phi)$ there exists $\sigma' \in \Sigma^\omega$ such that $\sigma\sigma' \models \phi$; furthermore, since he knows that $L_G(PN'_\phi)$ is completely fulfilling, he knows that all $\sigma \in L_G(PN'_\phi)$ fulfill all sets of accepting conditions specified by $\phi$ at least once. He also knows that $PN'_\phi$ is non-blocking. He can now intersect $PN_\phi$ with $PN_{system}$, following the correct algorithm depending on the supervision problem. The intersection system has all the above properties, with the probable exception of non-blocking.

The designer should now test his new system for emptiness; in fact it is possible that after all his work the completed system will be empty. Possible reasons for the system to be unable to fulfill the desirable specifications are errors in the construction of the system, errors in the translation of the natural language specification to EQLTL and the formula itself might be unable to be fulfilled in a discrete event system.

Now, with the system in hand, the designer can worry himself about controllability, which corresponds to Step 5.1.8; if he was simply looking for a way of assuring that the system met some specifications, the work is done, and he can use the Petri net obtained, with the respective language, in his following work. If he wants to check controllability conditions he still has some calculations to do. In particular, the non-conflicting condition is the harder to check. It is possible to check it, in the particular case if $L_G(PN_{system})$ is a regular language; in the general case, the designer can always assure himself up to a certain sequence of events length, which will be more than enough on most cases.

# 6

# Examples and Method Analysis

## 6.1 Applications

This section will present three applications of the method described in the earlier section. The analysis presented in this section were developed with the use of TINA, a software for Petri net analysis[2], and some author made programs in Mathematica. Our programs allowed us to intersect Petri nets, to translate between FSA and Petri nets, and to solve the membership problem for P-Type and G-Type Petri net languages. The translation between (EQ)LTL and FSA with Büchi acceptance criterion was done using an algorithm provided with the tool TINA. The tool TINA also provided precious algorithms to detect deadlocks, which were particularly relevant in the first example.

The first example will be based on the second formulation given here of the Three Dining Philosophers problem. We will show two examples of the application of the methodology. Consider then the Petri net in Figure 4.2, representing the classical Three Dining Philosophers Problem. We are interested in eliminating deadlock.

Since we are just interested in restricting some sequences of events, with no interest whatsoever in marking additional strings, we will use the P-Type language created by the specification and the P-Type language of the system. Furthermore, the description of the discrete event system does not hint that any event is uncontrollable, so we will not need to worry ourselves about the controllability condition.

We then need to state our natural language specification and translate it to LTL($\Sigma$). One way of eliminating deadlock in the classical three dining philosophers problem is to atomize certain non atomic operations. In this case, the problem stems from the fact that the philosopher can not grab two chopsticks at once, so if he grabs the right chopstick, he has to hope that his left peer does not grab the left peer's right chopstick before he grabs it himself. So the designer proposes the formula $(\bigwedge_{i=1}^{3}(G((g_{i,i} \vee g_{i,i-1 mod\ 3}) \Rightarrow (X(\bigwedge_{j=1,k=1,j\neq i,k\neq i}^{3}((\neg g_{j,k})U(r_{i,i} \vee r_{i,i-1 mod\ 3}))))))$. This way after a grab by the philosopher $i$ the designer is assured that all the other philosophers are unable to grab related chopsticks.

Since we are using a $LTL(\Sigma)$ formula, we only have to transform the formula into a generalized Büchi automaton, trim it, and translate the result into a Büchi automaton, and finally into a Petri net. After computing the intersection of the P-Type language of both Petri nets we will obtain a Petri net with deadlock over sequences of fireable transitions, although the deadlock is inexistent if we consider sequences of events. We will discuss the reasons for this important and significantly restricting feature in the next section. However, we can use the resulting Petri net as a decision procedure to answer whether a certain behaviour is valid or not. For instance, if the given input is $g11.g13.r11.g21.g22.r22.g32.r13.g33.r21$ we can decide whether or not the input is valid, computing also the sequence of transitions that would output such behaviour. For instance, our system accepts $g11.g13.r11.g21.g22.r22.g32.r13.g33.r21$ and rejects for instance $g11.g22.g33$, since after $g11$ it is now impossible to make a *grab action* that is not $g12$. The resulting Petri net has 35 places and 1156 transitions.

We now propose a different version of the three dining philosophers problem, referenced in Figure 6.1. This time, the philosophers announce their intentions; if philosopher $i$ is thinking and he wants to eat, he outputs an event called $w_i$. Furthermore, when he grabs all his chopsticks, he announces that he will start eating by outputting the event $s_i$. These new events are shown the Petri net below; they allows us to easily specify the property described above with a simple formula like $(\bigwedge_{i=1}^{3}(G(w_i \Rightarrow (X(((\neg s_{i-1}) \wedge (\neg s_{i+1}))Us_i)))))$. Nevertheless, we are now interested in a different problem: we wish to add objectives to our Petri net. We wish that eventually all philosophers should eat, at least once. Now, there is not a direct way of creating an adequate final set for the Petri net presented representing this objective. So what if we used our proposed method?

The first step is again to translate our goals into LTL($\Sigma$) form. We proposed $(F(s_1)) \wedge (F(s_2)) \wedge (F(s_3))$. Again, we transform the generalized Büchi automaton into a Büchi automaton. The Petri net representing the specification has 8 places and 156 transitions. Finally, the intersection of the specification with the Petri net described above outputs a system with 32 places and 252 transitions. The resulting system accepts $w1.g11.g13.\underline{s1}.w2.r11.g21.g22.\underline{s2}.w3.r22.g32.r13.g33.r21.\underline{s3}$, rejecting for instance $w1.g11.g13.\underline{s1}.w2.r11.g21.g22.\underline{s2}.w3.r22.g32.r13.g33.r21$. It is interesting to note that if we had used $((G(F(s_1))) \wedge (G(F(s_2))) \wedge (G(F(s_3))))$ as specification, then for instance the following sequence would be accepted in the prior resulting system but it would be rejected by the system we had just built: $w1.g11.g13.\underline{s1}.w2.w3.g32.r13.g33.\underline{s3}.r32.g22.r11.g21.\underline{s2}$. The reason for this discrepancy is the simple fact that this new specification requires $s_1$, $s_2$ and $s_3$ to be fired infinitely often, in opposition to the earlier specification which required the events to be fired at least once. The fact that in this new specification we require the philosophers to eat infinitely often implies that we can expect them to eat orderly, just by "forgetting" some of the events announcing they wish to eat. The difference is easily seen in the Büchi automaton version of both specifications. Nevertheless, considering our original specification we obtain a Petri net with objectives, whose G-Type language has only words that describe behaviours of the system of philosophers where all of them eat at least once.

If we wished to establish some objectives and restrict some sequences of transitions we could merge the two

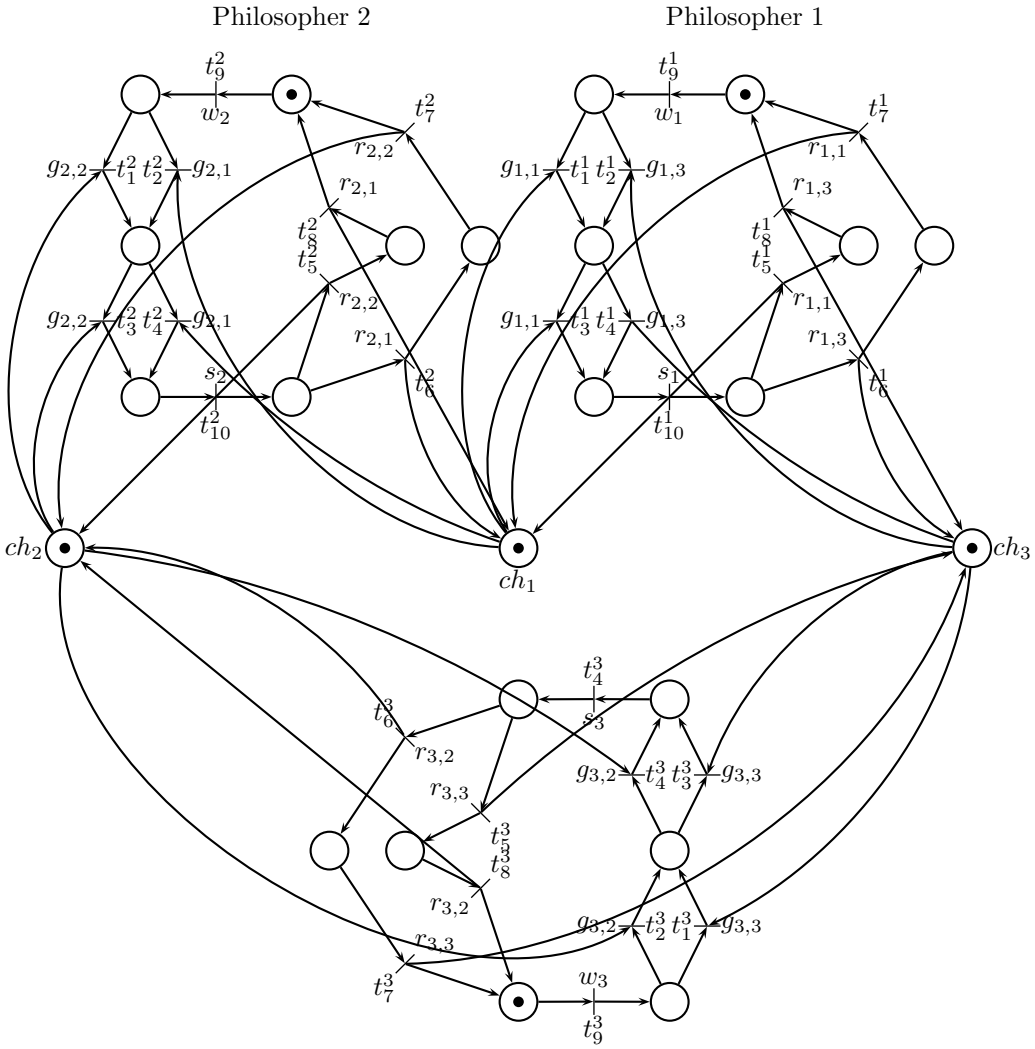


Figure 6.1: In this adaptation of the Three Dining Philosophers Problem, the Philosophers announce when they wish to eat (events $w_i$), and when they start eating (events $s_i$). There is still deadlock, as the philosophers are not allowed to put down chopsticks without eating first.

specifications forming the formula $(\bigwedge_{i=1}^{3}(G(w_i \Rightarrow (X(((\neg s_{i-1})\wedge(\neg s_{i+1}))U s_i)))))\wedge(F(s_1))\wedge(F(s_2))\wedge(F(s_3))$. With this formula we would avoid deadlock of sequences of events, and furthermore, any word accepted into the G-Type language of the resulting system would describe a sequence of behaviours where all of the philosophers would eat at least once. Unfortunately, our implementation of the algorithms could not cope with a specification this big. The Petri net representing the specification had 136 places and 6901 transitions. The Petri net of the resulting system has 161 places and 11533 transitions. Again, these objects are not minimal; they can and should be optimized, by eliminating dead transitions and places, eliminating also redundant transitions.

The second, smaller, example is inspired by a robotic soccer situation. We wish to model how to triangulate the ball properly between two robots. The Petri net consists primarily of four places for each robot ($p_j^i$ for robot $i$), a start place ($p_1$), representing how many triangulations will the robots perform, and a place

representing a ball not in the possession of the two robots ($p_2$). The basic idea is that the robot $i$ with the ball ($p_3^i$ or $p_4^i$) has to pass it to the other robot $j$, using the event $p_i$, preferably when the other robot is in position to catch the ball ($p_2^j$). If the robot $j$ is in position to catch the pass, it can either catch the ball ($p_3^j$) or fail to catch it, depending of course in external factors. If it is not in position, the robots lose possession of the ball ($p_2$). Either way, the robots can only perform this operation if there are tokens in the start place; furthermore if robot $i$ passed the ball, it will not be in position to receive it ($p_1^i$). Furthermore, if the ball is lost, they cannot repossess it. There are two additional events, one for each robot $pp_1$ and $pp_2$, that should be interpreted as *preparing to pass the ball*. A robot with the ball ($p_3^i$), before he can initiate the pass in ($p_4^i$) has to announce that it will pass the ball. It does so by outputting event $pp_i$. The following figure represents a model of the system outlined here.

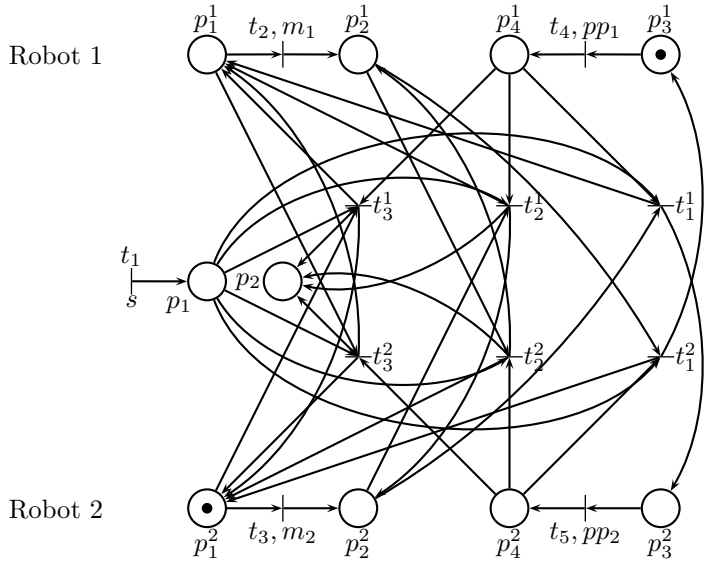


Figure 6.2: We shall name this Petri net $PN_{Tri}$. The transitions $t_j^i$ are labeled as the events $p_i$, which represent the act of robot $i$ passing the ball. The $j$-th robot is represented by the places $p_i^j$. The place $p_1$ represents how many triangulations will be made, and the place $p_2$ represents a lost ball.

Now, while this is a very simplistic model of a not so simple soccer action, it works for our intended purposes. We will add this time a termination condition, setting $F = \{p_3^1, p_3^2\}$, representing possession of the ball by the robots. We then consider the G-Type language $L_G(PN_{Tri})$, whose prefix-closure is not $L_P(PN_{Tri})$. Furthermore, in this case there are uncontrollable events: both events $p_1$ and $p_2$ are ideally uncontrollable, since the robot and ourselves cannot really know beforehand whether the pass is a success or not; even if a robot has the ball and the other robot is in the position to receive it, the pass can still fail, due to unexpected circumstances.

We are now ready to apply the method purposed in the earlier section. First, we begin by choosing the property we wish to enforce. We will start by a simple property that the net does not enforce, but it is

helpful. We wish that before any pass occurs, we have to set the maximum number of passes the robots will try to perform; this number cannot be altered during the execution of the actual movement. Now, a formula that expresses this property is $(G((\neg s) \Rightarrow (G(\neg s))))$. Since no triangulation has less than two passes, we also add an extra condition obtaining $s \wedge (Xs) \wedge (G((\neg s) \Rightarrow (G(\neg s))))$. Since this type of specification does not involve any type of objectives or goals, we will be concerned with the P-Type language of the Petri net representing the specification. Moreover, concerning controllability, it is easy to see that $\overline{L_G(PN_{Tri}) \cap L_P(Spec_1)}$ is controllable, since no uncontrollable active events are ever restricted. Following the method outlined in the earlier chapter, we will obtain a Petri net, which we will call $Spec_1$, which we can then intersect with $PN_{Tri}$ and obtain a system with the desired property. While this system is smaller than the final system of the earlier example, with only 14 places and 23 transitions, we will not present it here, since it has plenty of ramifications, which render any representation quite unintelligible. We can ask this system whether the sequence of events $\underline{s.s.s.s}.m2.pp1.p1.m1.pp2.p2.m2.pp1.p1.m1.pp2.p2$ belongs to the G-Type language of the intersection; the system will verify that it does belong to the language, while any sequence with a $s$ event anywhere but at the start will be rejected.

We will now suggest a different specification to apply at $PN_{Tri}$. This specification intends to minimize the loss of possession of the ball. For that, we have to ensure that a robot only passes the ball when the other robot is in position to receive it. This specification again does not set goals or objectives and it can be achieved using the formula $(G(m_2 \Leftrightarrow (Xpp_1))) \wedge (G(m_1 \Leftrightarrow (Xpp_2)))$. Interestingly enough, again $L_G(PN_{Tri}) \cap L_P(Spec_2)$ is controllable, and the same could be said if we had considered the P-Type language of $PN_{Tri}$. Note however, that we can not act upon the events $p_1$ and $p_2$; this means that if we had not designed $PN_{Tri}$ with the places $p_3^i$ we could not hope to construct a specification fulfilling the same objective while guaranteeing controllability. The intersection Petri net obtained in the end has 15 places and 22 transitions.

Finally, we applied the proposed method to the formula $(G(m_2 \Leftrightarrow (Xpp_1))) \wedge (G(m_1 \Leftrightarrow (Xpp_2))) \wedge s \wedge (Xs) \wedge (G((\neg s) \Rightarrow (G(\neg s))))$ and the intersection between the system and the specification had only 23 places and 23 transitions, with some evident spurious places. Furthermore, an interesting phenomenon was detected: we could actually use the system directly as map of possible actions, unlike the Dining Philosophers end result. The reason for this extra feature will be discussed in the following section, and it is one of the possible and very useful modifications of the method proposed; nevertheless we will hint that the reason for the existence of this extra feature in this end result is the fact that the specification is much more *deterministic* than the specification in the Dining Philosophers problem.

Our third example is quite easy to understand. There are four robots, placed in the vertices of a square, and they have to circulate the ball between adjacent robots. Furthermore, we will set an objective, which will be satisfied if the ball is in possession of a certain robot, named $Robot1$. While the initial model is simple, we will try to present some interesting supervisions.

Now, with this simple model $PRobots$, we are interested in considering the sequences such that the objective is reached. Now, it is easy to see that $L_G(PRobots)$ has plenty of non-optimal sequences, where non-

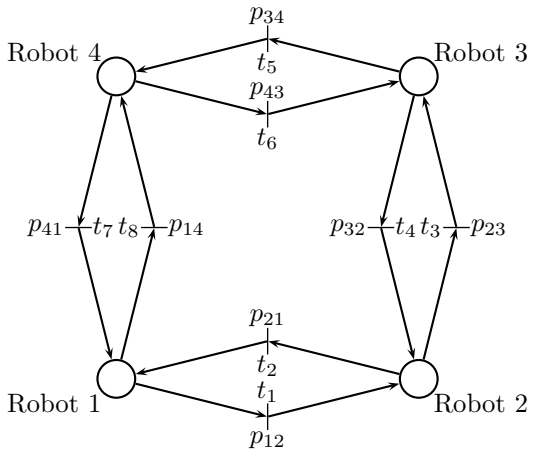


Figure 6.3: If a token is present in a certain place, we interpret it as the robot is in possession of the ball; furthermore, the set of final markings is $F = \{\text{Robot 1}\}$

optimality in this case means that there are internal useless loops like for instance $p_{12}, p_{23}, p_{32}, p_{23}, p_{34}, p_{41}$. We are interested in eliminating these useless loops. Another way of justifying the following specification would be state that we wish only to allow switches of the direction the ball is circulating when it is in the possession of a specific robot, in this case robot 1.

In order to formalize our specification, we chose the formula $\phi = (\phi_1 \wedge \phi_2)$, where $\phi_1 = (G(p_{14} \Rightarrow (X(((\neg p_{41}) \wedge (\neg p_{34}) \wedge (\neg p_{23}) \wedge (\neg p_{21}))Up_{21}))))$ and $\phi_2 = (G(p_{12} \Rightarrow (X(((\neg p_{21}) \wedge (\neg p_{32}) \wedge (\neg p_{43}) \wedge (\neg p_{14}))Up_{41}))))$. This specification is a little more expressive than what we initially described, since we are promising that if $p_{14}$ is fired then $p_{21}$ will be fired too, however, it is unimportant due to G-Type accepting condition. We then construct a generalized Büchi automaton, and after performing the usual operations we obtain a Petri net with 12 places and 76 transitions. It is interesting to check that the words accepted are precisely what we intended for. For instance, the resulting Petri net accepts the sequence of events $p_{14}p_{43}p_{32}p_{21}p_{14}p_{43}p_{32}p_{21}p_{12}p_{23}p_{34}p_{41}$. On the other side, it rejects for instance, $p_{14}p_{43}\underline{p_{32}p_{p23}}p_{34}p_{41}\underline{p_{12}p_{21}}p_{14}p_{43}p_{32}p_{21}p_{12}p_{23}p_{34}p_{41}$, due to fact that it is non-optimal and $p_{14}p_{43}p_{32}p_{21}p_{14}p_{43}p_{32}p_{21}p_{12}p_{23}p_{34}p_{41}\underline{p_{12}}$, because the original Petri net would already reject the word.

We are now interested, given the Petri net above, to introduce an extra specification. We are interested in circulating the ball first in one direction and after the passing is complete, we would wish to circulate the ball in the opposite direction. In order to do so, we have chosen the formula $\psi = \psi_1 \wedge \psi_2$, where $\psi_1 = (G(p_{12} \Rightarrow (X((G(\neg p_{12})) \vee ((\neg p_{12})Up_{14})))))$ and $\psi_2 = (G(p_{14} \Rightarrow (X((G(\neg p_{14})) \vee ((\neg p_{14})Up_{12})))))$. Merging both specifications $(\phi \wedge \psi)$, and following the method proposed we again obtain a sizable Petri net with 113 places and 1372 transitions. This Petri net exhibits quite restrictive behaviour, only accepting sequences such that after a sequence of passes if a new sequence begins, then it must go in the opposite direction, which is compliant with our initial specification. For instance, $p_{14}p_{43}p_{32}p_{21}p_{12}p_{23}p_{34}p_{41}p_{14}p_{43}p_{32}p_{21}p_{12}p_{23}p_{34}p_{41}$ is an accepted sequence, just like $p_{12}p_{23}p_{34}p_{41}$ or even $p_{14}p_{43}p_{32}p_{21}$. However, the earlier acceptable sequence $\underline{p_{14}}p_{43}p_{32}p_{21}\underline{p_{14}}p_{43}p_{32}p_{21}p_{12}p_{23}p_{34}p_{41}$ is no longer recognizable by this Petri net.

## 6.2 Method Analysis

In this section we will analyze some of the strong and weak points of the method proposed, suggesting also some possible adaptations. The analysis will be taking into account not only our initial goals, but also comparing the methodology proposed with some supervisory control of discrete event systems methods known by us.

First of all, we claim to have successfully created a method that allows us to guarantee that the event behaviours of a Petri net conform to a certain desired property. Moreover, our method definitely outputs a Petri net that can be further optimized. Assuming that the first statement is accepted, we have to agree that this method may relieve some designer workload when designing robotic tasks, which was our primary goal. We say **may** relieve, because not only our method does not output the smaller Petri net possible, but also in some cases it is evident what should the designer do to his Petri net plant in order for it to comply to his specifications. While this earlier objective was more of a declaration of principles than a concrete goal, we also have stated that we wished to extend the work by Lacerda and Lima[20], which we effectively extend, since our method used the more general formalization of Petri nets and the more powerful logic of QPLTL, again assuming the claim in the first sentence of this paragraph; furthermore, we have also given conditions to assure controllability of the final system event sequences, showing even that we can algorithmically test whether the P-Type language of the final system is controllable or not. We will then, in the following paragraph, try to convince the readers that the claim in this first paragraph is indeed reasonable.

Whatever the type of Petri net languages of the final system $FS$, we are assured that if $\sigma \in L(FS)$ implies that there exists $\sigma' \in \Sigma^\omega$ such that $\sigma\sigma' \models \phi$, where $\phi$ is the specification. However, this does not mean that $\sigma\sigma'$ is indeed fireable in $FS$. We could ask that $FS$ verifies this condition, however, we believe that the condition asked is strong enough. What it means is that $\sigma$ does not render impossible the fulfillment of $\phi$, or in other words, any sequence of events accepted by $FS$ does not contradict $\phi$. Of course, as we discussed earlier, if using G-Type languages, any word accepted by $FS$ fulfills some part of $\phi$, since all the conditions imposed by $\phi$, further ahead translated into sets of accepting conditions, are fulfilled at least once.

Now we will discuss the weak points of our method, and discuss possible solutions and adaptations:

- The method acts upon on sequences of events, unlike for instance, the method in [24] outlined in Section 4.2.2; this is not actually a disadvantage, it is simply a feature. It would be a disadvantage, however, if we compared the proposed method to a putative method capable of acting upon both markings and transitions directly.
- The controllability conditions expressed in Theorems 4.2.4, 4.2.5 and 4.2.6 are hard to check.
- There are properties difficult to express even using QPLTL, specially those indirectly concerning markings.
- As discussed in the earlier paragraph, we do not assure that we can effectively satisfy $\phi$ in the final

system; we can simply state that any accepted word does not contradict $\phi$.

- The latent non-determinism existing in both the plant and the specification implies that it is difficult to use the obtained system as a task plan, as in the typical applications [5, 26, 1]; this problem is evident in the Dining Philosophers problem, where even though we used a specification to eliminate deadlock of events, there was still deadlock in transitions.
- Unlike FSA, it is not trivial to eliminate spurious states and transitions; this problem is aggravated by the earlier point, since the non-determinism of the specification may generate *useless* states and transitions.

The first weak point is in fact a future work idea. It would be extremely useful if we could combine specifications using a temporal logic about transitions, and specifications using perhaps place invariants into one formalized system. Another possibility would be to use a more complex temporal logic with propositional symbols meaning not only *the event $a$ is going to happen now* but also *the marking is bigger than $\mu$*. Now, while we did not investigate this possibility throughly, we believe according to some issues [8], that this possibility may not be very successful.

Regarding the second point, for practical purposes it is simple to create an algorithm that tests controllability up to an arbitrary length of the sequences of events. While this is not very satisfying from a mathematician's point of view, the controllability conditions can be hand proven to be present in certain Petri nets. However, since the proof is on case-by-case basis, it is advisable to have either uncontrollable events only active in very specific situations, or if there are not many uncontrollable events at all.

Concerning the third point presented, we have to remind the reader that even if no other weak points existed, this method is not, by its intrinsic nature, designed to detect and restrict properties related to markings. Many other methods of supervisory control exist, probably more adequate to deal with these kinds of problems.

Finally, as we have already dealt with the fourth point presented, we will discuss a little more throughly the impact of non-determinism. Classically, the theory of supervisory control assumes that both the plant and the supervisor are deterministic. We chose, nevertheless, to present our results without particular concerns about whether the systems were deterministic or not, expecting that even using non-deterministic systems we could still end up obtaining an output that could be somehow useful. Moreover, it also would save the computational problem of the powerset construction needed to transform a NFSA into a DFSA. While, as argued above, and in fact shown in the Dining Philosophers example, we cannot use the final system as a plan, we can still use it as a decision procedure to ask whether or not certain behaviour is allowed, and so our effort was not completely in vain. Nevertheless, if the designer really wishes to use the method in order to obtain a system capable of generating behaviours we will discuss in the following paragraphs some changes to our method in order for it to address the real goals of the designer.

First of all, the main reason for the inability of non-deterministic systems to work as generators of behaviours

will be presented. Consider $PN = (P, T, I, O, \mu, \sigma, F)$. Let $\tau_1 \in L(PN)$, for any type of languages of Petri nets. Consider also $\tau \in \Sigma^*$, such that $\tau_1\tau \in L(PN)$, too. We know that there exists $\alpha_1 \in T^*$ such that $\mu\,|\alpha_1\rangle_{PN}$ and $\sigma(\alpha_1) = \tau_1$; we also know that there exists $\alpha_2 \in T^*$ such that $\mu\,|\alpha_2\rangle_{PN}$ with $\sigma(\alpha_2) = \tau_1\tau$. However, we can not prove or disprove the existence of $\alpha_3 \in T^*$ such that $\sigma(\alpha_1\alpha_3) = \tau_1\tau$ and $\mu\,|\alpha_1\alpha_3\rangle_{PN}$, because $\alpha_1$ is not necessarily a prefix of $\alpha_2$! This means for instance, that a robot, after firing a sequence of correct and desirable transitions, could eventually encounter itself in a position where it can not do anything allowable, and being required to effectively do something still; this situation would happen a lot for instance in the Dining Philosophers example.

To solve this situation, we first need the definition of a *deterministic* Petri net. This definition was adapted from [12]:

**Definition 6.2.1.** A *deterministic* Petri net $DPN = (P, T, I, O, \mu, \sigma, F)$ is a Petri net where for all reachable markings $\mu$, and for all transitions $t_1, t_2 \in T$, if $\mu\,|t_1\rangle_{DPN}$ and $\mu\,|t_2\rangle_{DPN}$ and $\sigma(t_1) = \sigma(t_2)$ then $t_1 = t_2$.

Now the basic idea is the following: we ignore the labeling function of the Petri net plant $PN_{system}$, setting it to be $\sigma_{system} = id$. We write a formula over $QPLTL(id(T)) = QPLTL(T)$. As usual, we transform the specification into a generalized Büchi automaton and further into a finite state automaton with Büchi accepting condition. Now, before we transform the finite state automaton into a Petri net, we will use the powerset construction and transform the finite state automaton with Büchi accepting condition into a deterministic version of the finite state automaton. Unfortunately, this deterministic version accepts more infinite sequences of events than the original version; for more information see [25]. However, the languages of finite sequences of events are precisely the same. When we transform the deterministic finite state automaton into a Petri net, it is easy to see that we obtain a deterministic Petri net $PN_{Spec}$, whose finite language (whether is P-Type or G-Type based) is precisely equal to the Petri net obtained if we had directly translated the possibly non deterministic finite state automaton.

Now, our claim is that the Petri net $PN_{Final}$ obtained from the intersecting $PN_{Spec}$ with $PN_{system}$ is a deterministic Petri net. Using this fact, we will then claim that we can use $PN_{Final}$ as a generator of behaviours.

**Proposition 6.2.1.** *Let $PN_{system} = (P_{system}, T_{system}, I_{system}, O_{system}, \mu_{system}, id, F_{system})$ be a Petri net. Let $PN_{Spec} = (P_{Spec}, T_{Spec}, I_{Spec}, O_{Spec}, \mu_{Spec}, \sigma_{Spec}, F_{Spec})$ be the deterministic Petri net obtained after following the method outlined above. Then, the Petri net obtained from the intersection of $PN_{system}$ with $PN_{Spec}$ is also a deterministic Petri net.*

**Proof.** Let $PN_{Final} = (P_{Final}, T_{Final}, I_{Final}, O_{Final}, \mu_{Final}, \sigma_{Final}, F_{Final})$ be the Petri net obtained from the intersection, as in Proposition 2.2.2. We have to prove that for all reachable markings $\mu$ of $PN_{Final}$, and for all transitions $t_1, t_2 \in T_{Final}$ if $\mu\,|t_1\rangle_{PN_{Final}}$, $\mu\,|t_2\rangle_{PN_{Final}}$ and $\sigma_{Final}(t_1) = \sigma_{Final}(t_2)$ then $t_1 = t_2$. Now, looking into the algorithm of the intersection, we know that $t_1$ was formed from some particular transitions $t_i \in T_{system}$ and $t_j \in T_{Spec}$ verifying $\sigma_{system}(t_i) = \sigma_{Spec}(t_j)$. The same can be said about $t_2$: there exists some transitions $t_k \in T_{system}$ and $t_l \in T_{Spec}$ such that $\sigma_{system}(t_k) = \sigma_{Spec}(t_l)$. Furthermore, since $\sigma_{system} =$

$id$, $\sigma_{Spec}(t_j) = t_i$ and $\sigma_{Spec}(t_l) = t_k$. Now, we have to use the fact that $\sigma_{Final}(t_1) = \sigma_{system}(t_i) = \sigma_{Spec}(t_j)$, $\sigma_{Final}(t_2) = \sigma_{system}(t_k) = \sigma_{Spec}(t_l)$ and the fact that the labels of $t_1$ and $t_2$ in the intersection to conclude that $t_i = t_k$.

Now, we know that $\mu\,|t_1\rangle_{PN_{Final}}$ and $\mu\,|t_2\rangle_{PN_{Final}}$. This information, taking into account the algorithm of the intersection, is equivalent to $pr_{system}(\mu)\,|t_i\rangle_{PN_{system}}$, $pr_{Spec}(\mu)\,|t_j\rangle_{PN_{Spec}}$ and $pr_{system}(\mu)\,|t_k\rangle_{PN_{system}}$, $pr_{Spec}(\mu)\,|t_l\rangle_{PN_{Spec}}$. Then, $pr_{Spec}(\mu)\,|t_j\rangle_{PN_{Spec}}$, $pr_{Spec}(\mu)\,|t_l\rangle_{PN_{Spec}}$ and $\sigma_{Spec}(t_j) = \sigma_{Spec}(t_l)$ allows us to conclude, since $PN_{Spec}$ is deterministic, that $t_j = t_l$. We can then conclude that since $t_i = t_k$ and $t_j = t_l$, that $t_1 = t_2$.

Now, it is simple to see that if $\tau \in L(PN)$, for some deterministic Petri net $PN = (P, T, I, O, \mu, \sigma, F)$, then there exists only one $\alpha \in T^*$ such that $\mu\,|\alpha\rangle_{PN}$ with $\sigma(\alpha) = \tau$. With this property in mind, it is simple to see that we can use a deterministic Petri net as a plan, because if $s \in L(PN)$ and $s.a \in L(PN)$ and if we used a certain sequence of transitions labeled as $s$, then we are certain that precisely that sequence of transitions has one continuation labeled as $s.a$. In fact, we could easily prove the same results only admitting that $PN_{system}$ is deterministic, which is clearly weaker than requiring $\sigma = id$.

So, we can then obtain a final system with similar properties to the system proposed in the firstly proposed method, but now able to be used as a generator of behaviours. Unfortunately, this method will probably output bigger systems, since we had to use the powerset construction upon the specification given in FSA form.

# Part III

# Considerations on Petri Nets $\omega$-Languages

# 7

# Petri net $\omega$-Languages

This chapter is intended as an extension of Section 2.3. We will discuss various topics: in the first section we will present some appropriate definitions for infinite sequences accepting conditions of Petri nets, from earlier literature. In the second section, we will discuss the claim made on Section 2.3: we claimed that our Petri net definitions were as far as we know more useful than those in [39], paving the road for a powerful characterization theorem, which will be the thematic of the third section.

## 7.1 Petri net $\omega$-languages definitions

As we discussed earlier, given a formalism like Petri nets or a finite state automata, in order to associate a $\omega$-language with a Petri net we have to create suitable accepting conditions. Just like in FSA, we should base our search for suitable accepting conditions of infinite sequences in the already existing ones for finite sequences. While it is hard to think of a suitable extension of the T-Type accepting condition, it is quite easy to generalize both P, G and L Type accepting criteria. Carstensen and Valk [3] proposed the following accepting conditions, based on L-Type accepting conditions. We will tag them with a $V$ as a subscript in order to remember their origin.

**Definition 7.1.1.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net. Let $\sigma : T \mapsto \Sigma$ be the $\lambda$-free labeling function. The $\omega$-languages accepted by $PN$, according to Valk's definitions, will be defined:

$$L^{\omega}_{V,True}(PN) = \{\sigma(\alpha) : \alpha \in T^{\omega}, \mu_0 \left|\alpha\right\rangle_{PN}\},$$

$$L^{\omega}_{V,ran\cap}(PN) = \{\sigma(\alpha) : \alpha \in T^{\omega}, \mu_0 \left|\alpha\right\rangle_{PN}, ran(PN(\alpha)) \cap F \neq \emptyset\},$$

$$L^{\omega}_{V,ran\subseteq}(PN) = \{\sigma(\alpha) : \alpha \in T^{\omega}, \mu_0 \left|\alpha\right\rangle_{PN}, ran(PN(\alpha)) \subseteq F\},$$

$$L^{\omega}_{V,inf\cap}(PN) = \{\sigma(\alpha) : \alpha \in T^{\omega}, \mu_0 \left|\alpha\right\rangle_{PN}, inf(PN(\alpha)) \cap F \neq \emptyset\},$$

$$L^{\omega}_{V,inf\subseteq}(PN) = \{\sigma(\alpha) : \alpha \in T^{\omega}, \mu_0 \left|\alpha\right\rangle_{PN}, inf(PN(\alpha)) \subseteq F\}.$$

While the conditions proposed in [3] are not exactly the ones presented here, these ones are enough for our future purposes. Unfortunately, in [3], the hierarchy of Petri net languages did not seem directly related with the hierarchy already presented in FSA (Theorem 1.2.1). However, in that article, the author proposes an alternative in order to have related classes of languages fulfilling a hierarchy similar to the FSA one. In [39], the author proposes another set of accepting conditions, proving that these accepting conditions originate precisely the classes of languages proposed as an alternative by Carstensen and Valk. Furthermore, these accepting conditions allowed Yamasaki to prove a powerful characterization theorem of the Petri net $\omega$-languages, relating a Petri net $\omega$-language with an appropriate FSA $\omega$-language.

Unfortunately, the definitions of some of the accepting conditions proposed by Yamasaki contained an imprecision. The intuitive definition given by Yamasaki in the introduction of [39] did not match the formal definition he proposed. Our definitions, proposed in 2.3, correct his imprecisions, allowing us to correctly prove his characterization theorem.

We will finally present Yamasaki's definitions. We will discuss the three sets of accepting conditions, comparing them with the basic FSA ones; we will pay, of course, special attention to the differences between our proposed definitions in Section 2.3, and Yamasaki's.

**Definition 7.1.2.** Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net. Let $\sigma : T \mapsto \Sigma$ be the $\lambda$-free labeling function. The $\omega$-languages accepted by $PN$, according to Yamasaki's definitions, will be defined:

$$L^{\omega}_{Y,True}(PN) = \{\sigma(\alpha) : \alpha \in T^{\omega}, \mu_0 \left|\alpha\right\rangle_{PN}\},$$

$$L^{\omega}_{Y,ran\cap}(PN) = \{\sigma(\alpha) : \alpha \in T^{\omega}, \mu_0 \left|\alpha\right\rangle_{PN}, ran(PN(\alpha)) \cap \uparrow F \neq \emptyset\},$$

$$L^{\omega}_{Y,ran\subseteq}(PN) = \{\sigma(\alpha) : \alpha \in T^{\omega}, \mu_0 \left|\alpha\right\rangle_{PN}, ran(PN(\alpha)) \subseteq \uparrow F\},$$

$$L^{\omega}_{Y,inf\cap}(PN) = \{\sigma(\alpha) : \alpha \in T^{\omega}, \mu_0 \left|\alpha\right\rangle_{PN}, inf(PN(\alpha)) \cap \uparrow F \neq \emptyset\},$$

$$L^{\omega}_{Y,inf\subseteq}(PN) = \{\sigma(\alpha) : \alpha \in T^{\omega}, \mu_0 \left|\alpha\right\rangle_{PN}, inf(PN(\alpha)) \subseteq \uparrow F\}.$$

The definitions introduced by Yamasaki [39], are exactly like the ones introduced by Valk, but with the filter set $\uparrow F$ instead of $F$. Again, the tag $Y$ serves as a reminder of who the author was.

We can now construct the families of languages, for $\beta \in \{Y, V\}$ and $\gamma \in \{True, ran\cap, ran\subseteq, inf\cap, inf\subseteq\}$:

$$P_{\beta,\gamma} = \left\{L \subseteq \Sigma^{\omega} : \text{there exists a Petri net } PN \text{ such that } L = L^{\omega}_{\beta,\gamma}(PN)\right\}.$$

## 7.2 Analysis of the accepting conditions

Notice that all Petri net languages $P_{True} = P_{V,True} = P_{Y,True}$. Furthermore, their motivation is straightforward. The list of accepted infinite behaviours are those who can effectively fire in the Petri net $PN$.

We shall now focus on the languages' definitions $L^{\omega}_{*,ran\cap}$. These types of language force the existence of at least one element of their acceptance set in the set of markings produced by a fireable infinite behaviour, either $F$ or $\uparrow F$. Again, the idea is also similar to the one present in FSA.

The third type of languages definitions, $L^{\omega}_{*,ran\subseteq}$, force all markings passed by an infinite behaviour to be present in the acceptance set, be it either $F$ or $\uparrow F$. The same idea is also present in FSA.

We arrive at the last two more interesting language definitions. Firstly, we shall compare the motivations behind $L^{\omega}_{*,inf\cap}$. We will first analyze the FSA one. The FSA one states that if an infinite behaviour is to be accepted, there will have to be a state that is passed infinitely often. Equivalently, the acceptance set $F$ is visited infinitely often. These conditions are equivalent since $F$ is finite. When we discuss the $L^{\omega}_{V,inf\cap}$ proposal, we notice that again the acceptance set $F$ is finite, and there should be a marking that visits it in order for some infinite behaviour to be accepted. We can conclude then that the meaning is preserved. Unfortunately, when we use infinite sets like $\uparrow F$, we cannot hope to preserve the equivalence. Notice the subtle difference between our proposal, $L^{\omega}_{inf\cap}(PN)$, and its inspiration, $L^{\omega}_{Y,inf\cap}(PN)$. In $L^{\omega}_{inf\cap}(PN)$, we are requiring the infinite behaviour to be fireable, and that it should pass infinitely often by the set $\uparrow F$. On the contrary, in $L^{\omega}_{Y,inf\cap}(PN)$, we are requiring something stronger the previous condition; we are requiring that there should be a marking in $\uparrow F$ that is passed by infinitely often. Since $\uparrow F$ is an infinite set, the notions are not equivalent,

**Proposition 7.2.1.** *Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be any Petri net. Then, $L^{\omega}_{Y,inf\cap}(PN) \subseteq L^{\omega}_{inf\cap}(PN)$. Moreover, there exists a Petri net $PN'$ such that $L^{\omega}_{inf\cap}(PN') \neq L^{\omega}_{Y,inf\cap}(PN')$.*

**Proof.** Let $\sigma \in L^{\omega}_{Y,inf\cap}(PN)$. Then there exists $\alpha \in T^{\omega}$ such that $inf(PN(\alpha))\cap \uparrow F \neq \emptyset$. But then, there exists a marking, $\mu$, in $\uparrow F$ such that is passed by infinitely often when firing $\alpha$. But then $PN(\alpha)|_{\mu}$ is an infinite sequence, and so $\omega Q(PN(\alpha)|_{\uparrow F}) = True$.

Consider the Petri net in Figure 7.1. Clearly, $a^{\omega} \in L^{\omega}_{inf\cap}(PN')$. The sequence of transitions $t_1^{\omega}$ originates the sequence of markings $PN'(t_1^{\omega}) = (1)(2)\dots(n)\dots$. Clearly, $PN'(t_1^{\omega})|_{\uparrow F} = PN'(t_1^{\omega})$ and so $\omega Q(PN'(t_1^{\omega})|_{\uparrow F}) = True$. However, $inf(PN'(t_1^{\omega})) = \emptyset$ and so $inf(PN'(t_1^{\omega}))\cap \uparrow F = \emptyset$, since after all there are no repetitions of markings. □

Finally, let us analyze $L^{\omega}_{*,inf\subseteq}$. When we analyze the FSA accepting criterion related we conclude that in order to accept a certain infinite behaviour, we will have to, after a certain point, only pass by elements of the acceptance set $F$. There is also an additional property that might be more or less irrelevant: $inf(pr_1(\alpha)) \neq \emptyset$, for all $s_0\,|\alpha\rangle$. In Petri nets, since we have an infinite state space, this property might not be true. Nevertheless, in $L^{\omega}_{V,inf\subseteq}$, if an infinite behaviour is accepted, not only it has to be fireable, but also after a certain point,

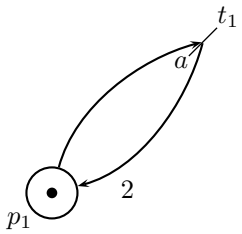


Figure 7.1: The Petri net $PN'$ represented above with $F = \{(1)\}$ is such that $L^{\omega}_{Y,inf\cap}(PN') \neq L^{\omega}_{inf\cap}(PN')$.

the only possible markings are those in $F$, assuming $inf(PN(\alpha)) \neq \emptyset$. This statement is true, since $F$ is finite, and consequently, there is a finite number of markings that can lead to an element of $F$, and so, it is impossible to have an infinite subsequence of different markings that lead to elements in $F$. It is also clear from the definition of $L^{\omega}_{inf\subseteq}$ that the accepting set $\uparrow F$ is eventually enclosing, or in other words, that any acceptable infinite transition sequence will eventually only pass by markings in the accepting set, $\uparrow F$. We will now show that $L^{\omega}_{inf\subseteq}(PN) \subseteq L^{\omega}_{Y,inf\subseteq}(PN)$, while the converse inclusion is not true.

**Proposition 7.2.2.** *Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net. Then $L^{\omega}_{inf\subseteq}(PN) \subseteq L^{\omega}_{Y,inf\subseteq}(PN)$, and there exists a Petri net $PN'$ such that $L^{\omega}_{Y,inf\subseteq}(PN') \not\subseteq L^{\omega}_{inf\subseteq}(PN')$.*

**Proof.** Let $s \in L^{\omega}_{inf\subseteq}(PN)$. Then, there exists $\alpha \in T^{\omega}$ such that $\mu_0 \left|\alpha\right\rangle_{PN}$ and $\omega Q(PN(\alpha)|_{C-\uparrow F}) = False$. The last condition means that the cardinality of the bag of reached markings that do not belong to $\uparrow F$ is finite. There are two possibilities: either $inf(PN(\alpha))$ is empty or it is not empty. If $inf(PN(\alpha)) = \emptyset$ then $inf(PN(\alpha)) \subseteq\uparrow F$. If it is not empty, then there has to be a marking that is repeated infinitely often. However, the bag of reached markings that do not belong to $\uparrow F$ is finite, so, it can not be one of those. It has to belong to $\uparrow F$. And so the inclusion is proved. We will now provide a counterexample for the converse inclusion. Clearly in the Petri net in Figure 7.2, $(ab)^{\omega} \in L^{\omega}_{Y,inf\subseteq}(PN')$, as the only infinite sequence of transitions that can produce such an output is $\alpha = (t_1t_2)^{\omega}$, and $PN'(\alpha) = (1,0)(0,1)(2,0)(1,1)(3,0)\ldots$ fulfilling $inf(PN'(\alpha)) = \emptyset \subseteq\uparrow F$. However, $PN'(\alpha)|_{C-\uparrow F} = (1,0)(2,0)(3,0)\ldots$, which is not a finite sequence. We can then conclude that $(ab)^{\omega} \notin L^{\omega}_{inf\subseteq}(PN')$. □

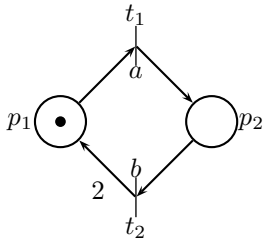


Figure 7.2: The Petri net $PN'$ represented above with $F = \{(0,1)\}$ is such that $L^{\omega}_{Y,inf\subseteq}(PN') \neq L^{\omega}_{inf\subseteq}(PN')$.

As a synthesis of this small discussion we have shown that the concepts behind the FSA accepting conditions definitions are maintained throughout the definitions of the earlier introduced accepting criteria and the

now-introduced, after adaptation, accepting criteria. This small discussion leaves, of course many interesting question left unanswered. For instance, we can not conclude any relationships between classes of languages, from this discussion. In fact we cannot either state $P_\gamma = P_{Y,\gamma}$ or $P_\gamma \neq P_{Y,\gamma}$, for $\gamma = \{inf\cap, inf\subseteq\}$. We also cannot establish relationships between $P_\gamma$ and $P_{V,\gamma}$, although they are probably incomparable, following statements in [39] and a theorem in [3].

## 7.3 Petri net $\omega$-language characterization theorem

We will now characterize each class of Petri net $\omega$-languages, presenting also the hierarchy of Petri net $\omega$-languages, as presented in [39, 3], using our proposed definitions. This characterization theorem, as we have already discussed, was proposed by Yamasaki, but as the definitions used were not appropriate, the proof of the theorem in [39] is not correct. First of all, let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net; if $R \subseteq T^\omega$ then we will define:

$$L^\omega_{True}(PN, R) = \{\sigma(\alpha) : \mu_0 \left|\alpha\right\rangle_{PN}, \alpha \in R\}.$$

We will introduce some notation in order to simplify the proof of the following theorem. If $f \in Z^X$ and $g \in Z^Y$, then $f \oplus g \in Z^{X\cup Y}$ is defined in the following way, where the $+$ assumes that there exists a sum operation over $Z$:

$$f \oplus g(x) = \begin{cases} f(x) + g(x) & \text{if } x \in X \cap Y \\ f(x) & \text{if } x \in X \\ g(x) & \text{if } x \in Y \end{cases}$$

For $n \in \mathbb{N}$ and a set $X$, $\mathbf{n}^X$ denotes the constant function in $\mathbb{N}^X$ such that $\mathbf{n}^X(x) = n$ for all $x \in X$. If $X$ is a singleton, we will write $\mathbf{n}^x$ instead of $\mathbf{n}^{\{x\}}$. The following theorem was found in [39], but as we already discussed that some of the definitions contained a small error that would not allow the proof of this theorem to be correct. We will provide the counterexamples, although they are just the applications of examples shown earlier in Proposition 7.2.2 and Proposition 7.2.1.

**Theorem 7.3.1.** *For any accepting condition* $\gamma \in \{True, ran\cap, ran\subseteq, inf\cap, inf\subseteq\}$, $P_\gamma = \{L^\omega_{True}(PN, R) : \text{PN is a Petri net}, R \in E_\gamma\}$.

**Proof.** This proof will have two main parts. We will not concern ourselves with $\gamma = \{True, ran\cap, ran\subseteq\}$, since these were already proven in [39]. Furthermore, the proof will follow the same lines as the article cited, but will be a bit more careful. The first part of the proof will show that if we consider a Petri net $PN = (P, T, I, O, \mu_0, \sigma, \emptyset)$ and a language $R$ over $T^\omega$ decided by a FSA with accepting condition $\gamma$, then there exists a Petri net $PN'$ such that $L^\omega_\gamma(PN') = L^\omega_{True}(PN, R)$. Let $M = (S, T, \delta, s_0, F)$ such that $L^\omega_\gamma(M) = R$. Then $PN' = (P', T', I', O', \mu'_0, \sigma', F')$ will be defined in the following way:

$$P' = P \cup S,$$

$$T' = \delta,$$

$$I'((s,t,s')) = I(t) \oplus \mathbf{0}^S \oplus \mathbf{1}^s,$$

$$O'((s,t,s')) = O(t) \oplus \mathbf{0}^S \oplus \mathbf{1}^{s'},$$

$$\mu'_0 = \mu_0 \oplus \mathbf{0}^S \oplus \mathbf{1}^{s_0},$$

$$\sigma'((s,t,s')) = \sigma(t) \text{ and finally,}$$

$$F' = \left\{ \mathbf{0}^P \oplus \mathbf{0}^S \oplus \mathbf{1}^q : \quad q \in F \right\}.$$

This is a possible way of intersecting these two different objects. It is easy to see that for any $\tau \in T'^*$, $m \oplus \mathbf{0}^P \oplus \mathbf{1}^s \left|\tau\right\rangle_{PN'} = m' \oplus \mathbf{0}^P \oplus \mathbf{1}^{s'}$ if and only if $m \left|pr_2(\tau)\right\rangle_{PN} = m'$ and $s \left|\tau\right\rangle_M = s'$. We just have to prove that $L^\omega_\gamma(PN') = L^\omega_{True}(PN, L^\omega_\gamma(M))$. Let us show first the inclusion $L^\omega_{True}(PN, L^\omega_\gamma(M)) \subseteq L^\omega_\gamma(PN')$ for the two remaining accepting conditions.

Consider $\gamma = inf\cap$. If $s \in L^\omega_{True}(PN, L^\omega_\gamma(M))$, then there exists a sequence of transitions $\alpha \in L^\omega_{inf\cap}(M)$ such that $\mu_0 \left|\alpha\right\rangle_{PN}$ and $\sigma(\alpha) = s$. Equivalently, there exists $\alpha' \in \delta^\omega$ such that $\mu_0 \left|pr_2(\alpha')\right\rangle_{PN}$, $s_0 \left|\alpha'\right\rangle_M$, $inf(M(\alpha')) \cap F \neq \emptyset$ and $\sigma'(\alpha') = s$.

From the first two conditions in the last sentence we conclude that $\mu'_0 \left|\alpha'\right\rangle_{PN'}$. We just need to check if the accepting condition $\omega Q(PN'(\alpha')|_{\uparrow F'}) = True$ is fulfilled. The accepting condition in $PN'$ is indeed fulfilled, since we have to pass infinitely often by markings in $\uparrow F'$ since in the automaton we have to pass infinitely often by states in $F$. Again, for the other accepting criterion, the proof is almost the same; the only change will be in the treatment of the accepting condition. But, remembering the discussion had after we introduced the $\omega$-languages, we now know that after a certain instant in time all markings in $PN'$ will be in $\uparrow F'$, since in the automaton after a certain moment we will always be in $F$.

In order to prove the converse inclusion let $s \in L_{inf\cap}(PN')$. Using the definition we obtain that there exists $\tau \in \delta^\omega$ such that $\mu'_0 \left|\tau\right\rangle_{PN'}$ and $\omega Q(PN'(\tau)|_{\uparrow F'}) = True$. Using the equivalence from above we obtain that $\mu \left|pr_2(\tau)\right\rangle_{PN}$, and $s_0 \left|\tau\right\rangle_M$. Furthermore, we know that the $\uparrow F'$ is visited infinitely often. This implies that states in $F$ are visited infinitely often, which concludes the proof for this accepting condition. Almost the same can be said about $inf \subseteq$ in the proof of both inclusions, and so we will not present it here.

The second part of this proof will show the opposite inclusion. We claim that if $PN = (P, T, I, O, \mu_0, \sigma, F)$ is a Petri net, then the $\omega$-language with accepting condition $\gamma$ is such that there exists a Petri net $PN'$ and a $\omega$-language $R$ acceptable by a FSA with accepting condition $\gamma$ over the transitions of $PN'$ such $L^\omega_\gamma(PN) = L^\omega_{True}(PN', R)$.

We will first define the Petri net $PN' = (P', T', I', O', \mu'_0, \sigma', \emptyset)$. Consider the following set $T_F = \{t_i^\mu : t_i \in T, \mu \in F\}$ and these definitions:

$$P' = P,$$

$$T' = T \cup T_F,$$

$$I'(t_i) = I(t_i), I'(t_i^\mu) = I(t_i) \cup \mu,$$

$$O'(t_i) = O(t_i), O'(t_i^\mu) = I(t_i) \cup \mu - I(t_i) + O(t_i),$$

$$\mu' = \mu,$$

$$\sigma'(t_i^\mu) = \sigma'(t_i) = \sigma(t_i).$$

The new transitions added have a interesting property: $m_1 \left|t_i^\mu\right\rangle_{PN'} = m_2$ if and only if $m_1 \in \uparrow F$ and $m_1 \left|t_i\right\rangle_{PN} = m_2$. The introduced transitions do the same job as the original ones, but they can also detect whether the current marking is in $\uparrow F$. Let us now show then that $L^\omega_{inf\cap}(PN) = L^\omega_{True}(PN', (T^*.T_F)^\omega)$.

Note that $(T^*.T_F)^\omega$ belongs to $E_{inf\cap}$. Assume $s \in L^\omega_{True}(PN', (T^*.T_F)^\omega)$. Then there exists $\alpha \in (T^*.T_F)^\omega$ such that $\mu_0 \left|\alpha\right\rangle_{PN'}$ with $\sigma'(\alpha) = s$. Consider $l : T \cup T_F \mapsto T$ defined as follows: $l(t_i) = l(t_i^\mu) = t_i$. We will then prove that $\mu_0 \left|l(\alpha)\right\rangle_{PN}$ and $\omega Q(PN(l(\alpha))|_{\uparrow F}) = True$. The first condition is obvious, we can simply use the property discussed when we introduced the new transitions. For the second condition, note that we know that transitions of the type $T_F$ fire infinitely often. Furthermore, we also know that when they fire, we must have been in a marking in $\uparrow F$. We can then conclude that $\uparrow F$ is touched infinitely often, which is enough to prove this inclusion.

For the converse inclusion, let $s \in L^\omega_{inf\cap}(PN)$. Then there exists $\alpha \in T^\omega$ such that $\mu_0 \left|\alpha\right\rangle_{PN}$, $\sigma(\alpha) = s$ and $\omega Q(N(\alpha)|_{\uparrow F}) = True$. We shall now consider the infinite sequence of moments where $PN(\alpha)$ passed by $\uparrow F$. We can now construct an inverse function $l^{-1} : T \mapsto T \cup T_F$ such that if a transition $t \in T$ fires with input marking $\mu$ verifying $\mu' \subseteq \mu$ with $\mu' \in F$ then we replace $t_i$ by $t_i^{\mu'}$. Now we are ready to prove that $\mu_0 \left|l^{-1}(\alpha)\right\rangle_{PN'}$ and that $l^{-1}(\alpha) \in (T^*.T_F)^\omega$. The first condition is simple to check, since all transitions $t_i^\mu$ were constructed to emulate the correspondent transitions $t_i$. The truth of the second condition is due to the fact that we know that the transitions in $T_F$ will have to fire infinitely often, since after all in $l^{-1}(\alpha)$ there are infinite transitions of the type $T_F$.

We will just have to discuss the final accepting criteria, $inf \subseteq$, proving $L^\omega_{inf\subseteq}(PN) = L^\omega_{True}(PN', T^*T_F^\omega)$. Note that $T^*T_F^\omega$ is recognizable by some FSA with accepting condition $inf \subseteq$.

Assume now that $s \in L^\omega_{True}(PN', T^*T_F^\omega)$. There exists $\alpha \in T^*T_F^\omega$, such that $\sigma'(\alpha) = s$ and $\mu_0 \left|\alpha\right\rangle_{PN'}$. Consider the function $l : T \cup T_F \mapsto T$, defined as above. We will show that $\mu_0 \left|l(\alpha)\right\rangle_{PN}$ and $\omega Q(PN(l(\alpha))|_{C-\uparrow F}) = False$. The first part is exactly the same as above. Furthermore, to prove the second part, we just have to state that if after a certain moment all transitions that fire are in $T_F$, this means that we are permanently in $\uparrow F$. So we can conclude that the markings that do not belong to $\uparrow F$ are in a finite number.

To prove the other inclusion, let $s \in L^\omega_{inf\subseteq}(PN)$. Then, there exists $\alpha \in T^\omega$ satisfying $\sigma(\alpha) = s$, $\mu_0 \left|\alpha\right\rangle_{PN}$ and $\omega Q(PN(\alpha)|_{C-\uparrow F}) = False$. Let $i$ be the index of the last element of $PN(\alpha)|_{C-\uparrow F}$ in $PN(\alpha)$. Consider then $\alpha' = \alpha(1), \alpha(2), \ldots, \alpha(i), l(\alpha(i+1), PN(\alpha), i+1), l(\alpha(i+2), PN(\alpha), i+2), \ldots \in T \cup T_F$. The function

$l : T \times C^\omega \times \mathbb{N} :\mapsto T \cup T_F$ is defined in the following way for $t_i \in T$.

$$l(t_i, \beta, n) = t_i^\mu, \text{ if } \mu \subseteq \beta(n) \text{ with } \mu \in F$$

If there are two or more choices of $\mu$, choose any one of them. Clearly, $\alpha' \in T^*T_F^\omega$. We just have to check that $\mu_0 \,|\alpha'\rangle_{PN'}$. Let the prefix of the $i$ first transitions of $\alpha'$ be called $\alpha'_{ini}$. Then it is clear that $\mu_0 \,|\alpha'_{ini}\rangle_{PN'}$. We will now check that $\mu_0 \,|\alpha'_{ini} l(\alpha(i+1), PN(\alpha), i+1)\rangle_{PN'}$. Since $\mu_i \in\uparrow F$, in particular there is at least one marking $\mu$ in $F$ such $\mu \subseteq \mu_i$. This means that the conditions for $l(\alpha(i+1), PN(\alpha), i+1)$ are satisfied and so it can fire. Furthermore, as we have seen, we can repeat this idea, since $l(\alpha(i+1), PN(\alpha), i+1)$ will give conditions for $l(\alpha(i+2), PN(\alpha), i+2)$ to fire. □

This final theorem characterizes Petri net $\omega$-languages in a clear way. It states that all infinite behaviour accepted by a Petri are in fact the result of the sieving of a similar kind FSA infinite language through a Petri net. This theorem can also be used to prove important closure properties about the Petri nets $\omega$-languages. In fact in [39] we can check the proof for the closure for union and intersection. We have the following hierarchy, which is now clear after the last theorem.

**Proposition 7.3.1.** $P_{True} = P_{ran\subseteq} \subset P_{ran\cap} = P_{inf\subseteq} \subset P_{inf\cap}$.

We will now present two counter-examples to Yamasaki's proof of the characterization theorem.

**Example 7.3.1.** We will develop on already presented example to show that if we consider $L^\omega_{Y,inf\cap}(PN)$, then $L^\omega_{Y,inf\cap}(PN) \neq L^\omega_{True}(PN', (T^*.T_F)^\omega)$, and hence our need for presenting new definitions. Notice

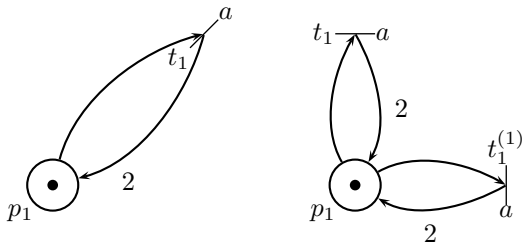


Figure 7.3: The Petri net at the left will be called $PN$. Its accepting set is $F = \{p_1\}$. The Petri net at the right is the Petri obtained by the construction described in the second part of Theorem 7.3.1.

in Figure 7.3, that $a^\omega \in L^\omega_{True}(PN', (T^*T_F)^\omega)$, just by picking the fireable sequence $(t_1 t_1^{(1)})^\omega$. However, $L^\omega_{Y,inf\cap}(PN) = \emptyset$, since $inf(PN(t_1^\omega)) = \emptyset$.

**Example 7.3.2.** We will develop an already given example in order to show that $L^\omega_{Y,inf\subseteq}(PN) \neq L^\omega_{True}(PN', T^*T_F^\omega)$, further compelling us to present new definitions. It is easy to see that, in Figure 7.4, that $(ab)^\omega \in L^\omega_{Y,inf\subseteq}(PN)$. However, we will show that $(ab)^\omega \notin L^\omega_{True}(PN', T.T_F^\omega)$. The elements of $T.T_F^\omega$ that are mapped by $\sigma'$ to $(ab)^\omega$ are $(t_1t_2)^*(t_1^{(0,1)}t_2^{(0,1)})^\omega$ and $(t_1t_2)^*t_1t_2^{(0,1)}(t_1^{(0,1)}t_2^{(0,1)})^\omega$. It is easy to see that neither element can fire. The first element cannot fire in $PN'$ since after $(t_1t_2)^*$ the marking will be $(n, 0)$, for some $n$, and so $t_1^{(0,1)}$ cannot fire. Furthermore, after $(t_1t_2)^*t_1t_2^{(0,1)}$ the marking will be $(n, 0)$ and, again, $t_1^{(0,1)}$ cannot fire.

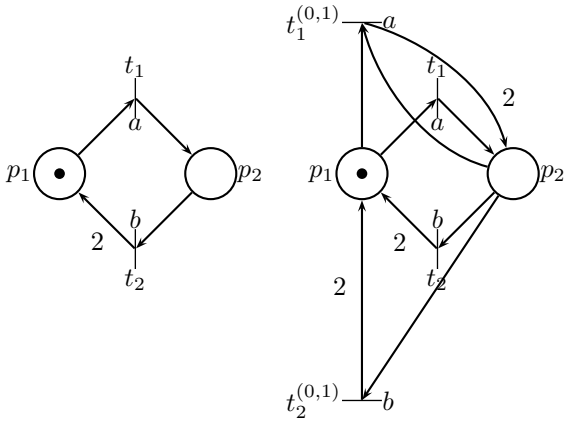


Figure 7.4: The Petri net at the left will be called $PN$. Its accepting set is $F = \{(0,1)\}$. The Petri net at the right, $PN'$, is the Petri net obtained by the construction described in the latter part of Theorem 7.3.1.

While these last propositions do not have a direct impact into our work, we felt that they offer a small but important contribution to the study of Petri net $\omega$-Language Theory; while browsing through Yamasaki's paper, his work seemed to capitalize mainly on these now fully proven propositions, and so, it is probable that all his results are correct, with these new definitions. Furthermore, the proof in the next chapter draws heavily upon our proposed definitions of $L^{\omega}_{inf\cap}$.

# 8

# Multiple accepting conditions Petri nets concept, and their impact to our work

As the earlier chapter, this chapter is intended as an extension of Section 2.3. In this chapter, we will present in the first section a generalization of the concept of the generalized Büchi automaton to Petri nets. This generalization can be applied to our proposed methodology of Chapter 5, although only in some cases it is profitable to use it; this will be the issue of the second section of this chapter.

## 8.1 Multiple accepting conditions Petri nets and their equivalence to Büchi Petri nets

Despite the discussions in the earlier chapter, we now turn our attention to another problem. We have found our corresponding Büchi Petri net acceptance criterion. But is there a natural extension in Petri nets of the generalized Büchi automaton concept? Ideally, this extension should preserve the existing equivalence between generalized Büchi automata and Büchi automata into Petri nets. If such an extension existed, instead of translating a generalized Büchi automaton into a Büchi automaton, as in Step 5.1.5, we could transform the generalized Büchi automaton into this extension, and then use the preserved equivalence to obtain a Büchi Petri net which would accept precisely the same language as the Büchi Petri net obtained through the proposed methodology. Of course, this alternative method is only worth considering if somehow we are able to transform the extension of a generalized Büchi automaton into a Büchi Petri net *faster* than what we can achieve with the proposed methodology.

**Remark 8.1.1.** This new definition and its possible equivalence could have a direct practical application in our thesis. Consider the Figure 8.1. If the application of dashed transformations provide us with a smaller Petri net than what we would have obtained by following the more obvious route with the full lines, then, for efficiency sake, the new route is preferable, since in the end they represent the same language, but within a smaller representation.

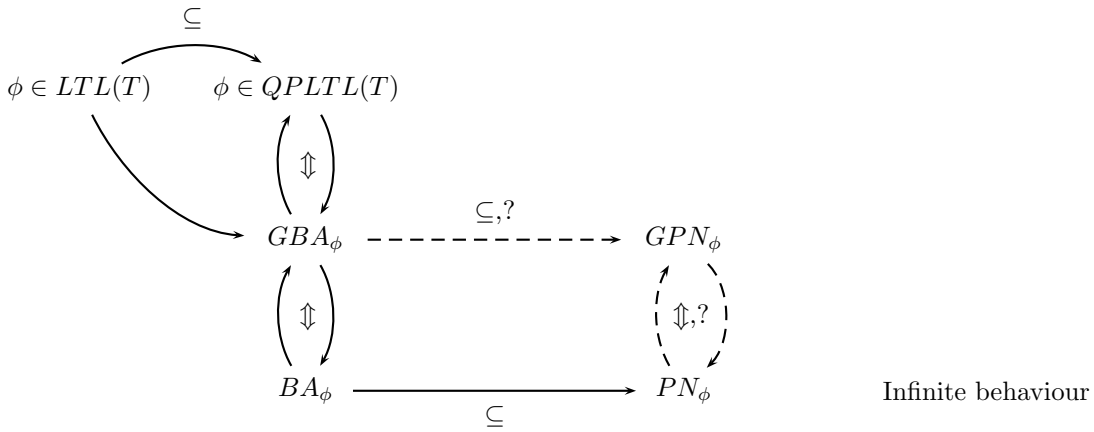


Figure 8.1: Motivation for the concept of generalized Büchi Petri nets. The question is whether this concept exists with good properties relating to Büchi Petri nets, and whether the dashed path is "faster" than the already known path, written with full lines.

So, we shall define the analog of generalized Büchi automata to Petri nets.

**Definition 8.1.1.** Let $PN = (P, T, I, O, \mu_0, \sigma)$ be a $\lambda$-free marked Petri net structure and $\mathcal{F} = \{F_1, \ldots, F_n\}$ be a finite set of sets of accepting markings. $MACPN = (P, T, I, O, \mu_0, \sigma, \mathcal{F})$ is called a *multiple accepting conditions Petri net* (MACPN). The language accepted by a multiple accepting conditions Petri net $MACPN$, represented by $L^\omega_{\forall inf \cap}(MACPN)$, is defined as follows:

$$L^\omega_{\forall inf \cap}(MACPN) = \{\sigma(\alpha) : \alpha \in T^\omega, \mu_0 \left|\alpha\right\rangle_{MACPN}, \forall_{F_i \in \mathcal{F}} \quad \omega Q(MACPN(\alpha)|_{\uparrow F_i}) = True\}.$$

If some multiple accepting conditions Petri net language recognized by $MACPN$ is in fact recognizable by a generalized Büchi automaton $B$, it is simple to see that there is a regular Petri net $PN'$, with only one set of accepting conditions, such that $L^\omega_{\forall inf \cap}(MACPN) = L^\omega_{inf \cap}(PN')$, just by using the equivalence between generalized Büchi automata and Büchi automata. In fact, for the practical purposes of this thesis, this would be all that is strictly required. Nevertheless, we will prove the general equivalence. The general equivalence allows us to be much more *synthetic* in our definitions of Petri net $\omega$-languages, just like in Büchi automata, as we can use a strictly more powerful way of expressing a condition, obtaining automatically the algorithm to transform a Petri net equipped with this new accepting condition into a Büchi Petri net.

This proof could be shorter and more clear, however, our goal with this theorem was to show constructively how to transform MACPNs into PNs, paying special attention on how many new places and transition are introduced. Clearly, this is not the best algorithm, however, it can already be used to as a benchmark in order to see if the newly introduced route is faster than the more obvious one, as discussed in Remark 8.1.1.

The main idea consists on detecting when elements of $F_i \in \mathcal{F}$ are reached, and only when all of them are reached will we start to detect them again; furthermore, since they happen infinitely often, we can ignore some of them. We set also the accepting markings of the regular Petri net as the filter set of the marking that detects when all of them are reached. Furthermore, the mechanism that detects that when each $F_i$ is

reached is reset when all of them are reached.

**Definition 8.1.2.** Let $MACPN = (P, T, I, O, \mu_0, \sigma, \mathcal{F})$ be a multiple accepting conditions Petri net with $\sigma : T \mapsto \Sigma$ the $\lambda$-free labeling function and $\mathcal{F} = \{F_1, \ldots, F_n\}$ with $n > 1$. Then, the Petri net $PN_{MACPN} = (P', T', I', O', \mu_0', \sigma', F')$ is called a *witness* of $MACPN$, and it is defined as follows:

$$P' = P \cup \{p_{RF_i} : 1 \leq i \leq n\} \cup \{p_R\} \cup \{p_F\}.$$

$$T' = \{t_i' : t_i \in T\} \cup \left\{t_{i,R}' : t_i \in T\right\} \cup \left\{t_i'^{\mu,j} : t_i \in T, 1 \leq j \leq n, \mu \in F_j\right\}.$$

$$I'(t_i') = I(t_i) + \{p_R\}.$$

$$I'(t_{i,R}') = I(t_i) + \sum_{i=1}^{n} \{p_F\}.$$

$$I'(t_i'^{\mu,j}) = (\mu - O(t_i)) + I(t_i) + \{p_R\} + \left\{p_{RF_j}\right\}.$$

$$O'(t_i^{'}) = O(t_i) + \{p_R\}.$$

$$O'(t_{i,R}^{'}) = O(t_i) + \sum_{i=1}^{n} \{p_R\} + \sum_{i=1}^{n} \{p_{RF_i}\}.$$

$$O'(t_i'^{\mu,j}) = (\mu - O(t_i)) + O(t_i) + \{p_F\}.$$

$$\mu_0' = (\mu_0, \underbrace{1, \ldots, 1}_{n}, n, 0).$$

$$\sigma'(t_i') = \sigma'(t_{i,R}') = \sigma'(t_i'^{\mu,j}) = \sigma(t_i).$$

$$F = \left\{ (\underbrace{0, \ldots, 0}_{m}, \underbrace{0, \ldots, 0}_{n}, 0, n) \right\}.$$

We will call the places of the witness, $PN_{MACPN}$, $\{p_R\}$,$\{p_{RF_i}\}$ and $\{p_F\}$ supervisor places. The places $p_{RF_j}$ will be the ones responsible for detecting reached $F_j$. The places $p_R$ and $p_F$ will be responsible for detecting if all of the $F_j$ were reached and reseting the detection mechanism. The places corresponding to $MACPN$ are called original places. The projection $pr_{Ori}$ projects a marking of the witness into the original places, and $pr_{Sup}$ projects the marking into the supervisor places. The Parikh mapping considered sets the $p_i$ by the index order at the start, followed by the $p_{RF_i}$, again by index order, and finally, $p_R$ and $p_F$. Another important remark for the future is that any symbol with the form $(.)'$ means that we are writing about the witness.

We will first prove that if the witness of a certain multiple accepting conditions Petri net $MACPN$ accepts a certain infinite sequence of events, then $MACPN$ also accepts that sequence of events.

**Definition 8.1.3.** Let $MACPN = (P, T, I, O, \mu_0, \sigma, \mathcal{F})$ be a multiple accepting conditions Petri net with $\sigma : T \mapsto \Sigma$ the $\lambda$-free labeling function and $\mathcal{F} = \{F_1, \ldots, F_n\}$ with $n > 1$ and $PN_{MACPN} =$

$(P', T', I', O', \mu'_0, \sigma', F')$ the witness of $MACPN$. Consider $l : T' \mapsto T$ defined as follows:

$$l(t'_i) = l(t'_{i,R}) = l(t'^{\mu,j}_i) = t_i.$$

We shall call $l$ the *linking* function, as it links the witness to the original multiple accepting conditions Petri net.

It is easy to see that $l$ is a function that preserves labeling functions, meaning that if $\alpha \in T'^{\omega}$, then $\sigma'(\alpha) = \sigma(l(\alpha))$.

**Proposition 8.1.1.** *Let $MACPN = (P, T, I, O, \mu_0, \sigma, \mathcal{F})$ be a multiple accepting conditions Petri net with $\sigma : T \mapsto \Sigma$ the $\lambda$-free labeling function and $\mathcal{F} = \{F_1, \ldots, F_n\}$ with $n > 1$ and $PN_{MACPN} = (P', T', I', O', \mu'_0, \sigma', F')$ the witness of $MACPN$. Then $L^{\omega}_{inf\cap}(PN_{MACPN}) \subseteq L^{\omega}_{\forall inf\cap}(MACPN)$.*

**Proof.** Set $\#P = m$ and $\#\mathcal{F} = n$.

We will first start by noticing that the net $PN_{MACPN}$ has a place invariant given by $p_R + \sum_{i=1}^{n} p_{RF_i} + 2p_F = 2n$. This equation is a place invariant since all transitions preserve the place invariant, and the initial marking $\mu'_0$ obeys the place invariant.

Let $\tau \in \Sigma^{\omega}$ belong to $L^{\omega}_{inf\cap}(PN_{MACPN})$. Then, there exists $\alpha' \in T'^{\omega}$ such that $\sigma'(\alpha') = \tau$, $\mu'_0 \,|\alpha'\rangle_{PN_{MACPN}}$ and $\omega Q(PN_{MACPN}(\alpha')|_{\uparrow(\underbrace{0,\ldots,0}_{m},\underbrace{0,\ldots,0}_{n},0,n)}) = True$.

We will prove that $l(\alpha')$ is such that $\mu_0 \,|l(\alpha')\rangle_{MACPN}$ and that $\forall_{F_i \in \mathcal{F}} \quad \omega Q(MACPN(l(\alpha'))|_{\uparrow F_i}) = True$.

First, we will now prove that the two nets are related by $pr_{Ori}(\mu'_j) = \mu_j$ for any $j$, where $\mu'_j$ is the $j^{th}$ marking of $PN_{MACPN}$ following $\alpha'$, and $\mu_j$ is the $j^{th}$ marking in $MACPN$ if we could follow $l(\alpha')$.

Clearly, $pr_{Ori}(\mu'_0) = \mu_0$. Assume now that $pr_{Ori}(\mu'_j) = \mu_j$. We shall prove that $pr_{Ori}(\mu'_{j+1}) = \mu_{j+1}$. There are three cases depending on which type of transition fired. Let say that $t'_{k_{j+1}} = t'_r$, where the $t'_{k_{j+1}}$ is the $(j+1)^{th}$ transition of $\alpha'$, for some $t'_r \in T'$.

$$\begin{aligned}
\mu'_{j+1} &= \mu'_j - I'(t'_r) + O'(t'_r) \\
&= \mu'_j - I(t_r) - \{p_R\} + O(t_r) + \{p_R\} \\
&= \mu'_j - I(t_r) + O(t_r) \\
pr_{Ori}(\mu'_{j+1}) &= pr_{Ori}(\mu'_j) - I(t_r) + O(t_r) \\
pr_{Ori}(\mu'_{j+1}) &= \mu_j - I(l(t'_r)) + O(l(t'_r)) \\
&= \mu_{j+1}.
\end{aligned}$$

Let us say that $t'_{k_{j+1}} = t'^{\mu,s}_r$ for some $\mu \in F_s$. Well, then $\mu'_j$ has to permit the firing of $t'^{\mu,s}_r$ and $I'(t'^{\mu,s}_r) = (\mu - O(t_r)) + I(t_r) + \{p_R\} + \{p_{RF_s}\}$ so $\mu'_j = B + (\mu - O(t_r)) + I(t_r) + \{p_R\} + \{p_{RF_s}\}$. We also know that

$\mu_j = pr_{Ori}(B) + (\mu - O(t_r)) + I(t_r)$; let us then calculate $\mu'_{j+1}$.

$$\begin{aligned}
\mu'_{j+1} &= B + (\mu - O(t_r)) + I(t_r) + \{p_R\} + \{p_{RF_s}\} - ((\mu - O(t_r)) + I(t_r) + \{p_R\} + \{p_{RF_s}\}) \\
&\quad + (\mu - O(t_r)) + O(t_r) + \{p_F\} \\
&= B + (\mu - O(t_r)) + O(t_r) + \{p_F\} \\
pr_{Ori}(\mu'_{j+1}) &= pr_{Ori}(B) + (\mu - O(t_r)) + O(t_r)
\end{aligned}$$

Let us calculate finally $\mu_{j+1}$

$$\begin{aligned}
\mu_{j+1} &= pr_{Ori}(B) + (\mu - O(t_r))) + I(t_r) - I(l(t'^{\mu,s}_r)) + O(l(t'^{\mu,s}_r)) \\
&= pr_{Ori}(B) + (\mu - O(t_r)) + I(t_r) - I(t_r) + O(t_r) \\
&= pr_{Ori}(B) + (\mu - O(t_r)) + O(t_r)
\end{aligned}$$

We conclude finally $\mu_{j+1} = pr_{Ori}(\mu'_{j+1})$. Before we move on to the next case it is important to note that $\mu \subseteq \mu_{j+1}$ with $\mu$ being a accepting marking belonging to $F_s$.

The final case is if $t'_{k_{j+1}} = t'_{r,R}$; let say that $t'_{k_{j+1}} = t'_{r,R}$, for some $t'_{r,R} \in T'$.

$$\begin{aligned}
\mu'_{j+1} &= \mu'_j - I'(t'_{r,R}) + O'(t'_{r,R}) \\
&= \mu'_j - (I(t_r) + \sum_{i=1}^{n} \{p_F\}) + O(t_r) + \sum_{i=1}^{n} \{p_R\} + \sum_{i=1}^{n} \{p_{RF_i}\} \\
pr_{Ori}(\mu'_{j+1}) &= pr_{Ori}(\mu'_j) - I(t_r) + O(t_r) \\
&= \mu_j - I(l(t'_{r,R})) + O(l(t'_{r,R})) \\
&= \mu_{j+1}
\end{aligned}$$

So, we have proven that for $\mu_0 \, |l(\alpha')\rangle_{MACPN}$, since we have shown that the markings in the original places in $PN_{MACPN}$ are the same as in $MACPN$ and it can be easily seen in the definition of witness that the conditions for transitions to fire in $MACPN$ are precisely the same as the conditions for transitions to fire in $PN_{MACPN}$, regarding original places, of course. Let us see that $\forall_{F_i \in \mathcal{F}} \quad \omega Q(MACPN(l(\alpha'))|_{\uparrow F_i}) = True$.

As $\alpha'$ reaches $\uparrow F$ infinitely often, this will mean that there will be infinitely many firings of transitions of the type $t'_{r,R}$. But, for a transition like $t'_{r,R}$ to fire, $n$ tokens must exist in $p_F$; as the initial marking has no token in $p_F$, and when $t'_{r,R}$ fires, all tokens are taken from $p_F$, this implies that for each firing of $t'_{r,R}$, $n$ firings of transitions of the type $t'^{\mu,j}_r$ must occur. Furthermore, the $n$ firings have to be of different $j$, which means that for each firing of $t'_{r,R}$ there will be a set of $n$ transitions of all with different $j$. But, as we have just seen, when $t'_{r,R}$ fires, a marking in $\uparrow F$ is reached meaning that between each firing of $t'_{r,R}$ all sets $\uparrow F_j$ are reached. This, of course concludes the proof of the inclusion. □

The proof of the other inclusion is quite more cumbersome. Our idea is simply to guess an appropriate

inverse of $l$, and show that the inverse of $l$ is fireable in the witness of $MACPN$. We will first define the appropriate inverse of $l$.

**Definition 8.1.4.** Let $MACPN = (P, T, I, O, \mu_0, \sigma, \mathcal{F})$ be a multiple accepting conditions Petri net with $\sigma : T \mapsto \Sigma$ the $\lambda$-free labeling function and $\mathcal{F} = \{F_1, \ldots, F_n\}$ with $n > 1$ and $PN_{MACPN} = (P', T', I', O', \mu_0', \sigma', F')$ the witness of $MACPN$. Let $l : T' \mapsto T$ defined as above. Let $s \in L^{\omega}_{\forall inf \cap}(MACPN)$, and $\alpha \in T^{\omega}$ such that $\sigma(\alpha) = s$, $\mu_0 \left|\alpha\right\rangle_{MACPN}$ and $\forall_{F_i \in \mathcal{F}} \quad \omega Q(MACPN(\alpha)|_{\uparrow F_i}) = True$. Then, $\Omega \in l^{-1}(\alpha)$ is defined in the following discussion.

It is possible to build $n = \#\mathcal{F}$ sequences of natural numbers such that each element is the instant when each $\uparrow F_i$ is reached when firing $\alpha$, while saving at the same time which filter set of which marking was effectively reached. The sequence corresponding to $F_j$ is called $i^{F_j}$. For example, $\uparrow F_1$ was reached in $i^{F_1} = (i^{F_1}_{\mu_1,1}, i^{F_1}_{\mu_2,2}, \ldots, i^{F_1}_{\mu_n,n}, \ldots), i^{F_1}_{\mu_j,j} < i^{F_1}_{\mu_j,j+1}$. It is possible to choose a subsequence, called $i^{\mathcal{F}}$, from all these $n = \#F$ sequences, such that each element belongs to $i^{F_j}$ and the element next to an element from $i^{F_j}$ is an element from $i^{F_{(j+1)modn}}$. It is also possible to enforce the restriction that $i^{\mathcal{F}}_k + 2 < i^{\mathcal{F}}_{k+1}$. This last restriction serves only the purpose of simplifying the proof, further ahead. All these choices are possible due to the fact that all $\uparrow F_i$ are reached infinitely often. We will now choose an element of $l^{-1}(\alpha)$ based on the sequence of indexes $i^{\mathcal{F}}$. We will now rename $i^{\mathcal{F}}$ allowing notation to be more clear.

$$i^{\mathcal{F}} = (i^{\mathcal{F}}_1, i^{\mathcal{F}}_2, \ldots, i^{\mathcal{F}}_n, \ldots).$$

Do not forget that $i^{\mathcal{F}}_i = i^{F_{i(modn)}}_{\mu_s,s}$ for some $\mu_s \in F_{i(modn)}$, in the $s^{th}$ time $\uparrow F_{i(modn)}$ was reached.

Consider then that $\alpha = t_{k_1} t_{k_2} \ldots t_{k_n} \ldots \in T^{\omega}$, with $k_i \in \{1, \ldots, \#T\}$, and the corresponding run:

$$\mu_0 \left|\alpha\right\rangle_{MACPN} = (\mu_0, t_{k_1}, \mu_1) \ldots (\mu_{i^{\mathcal{F}}_1 - 1}, t_{k_{i^{\mathcal{F}}_1}}, \mu_{i^{\mathcal{F}}_1}) \ldots (\mu_{i^{\mathcal{F}}_n - 1}, t_{k_{i^{\mathcal{F}}_n}}, \mu_{i^{\mathcal{F}}_n}) \ldots$$

We will define then $\Omega$ to be one of the inverses of $\alpha$. Clearly, we can associate to the same transition $t_i \in T$ different transitions in $T'$ provided that they are not assigned in the same instant. We will then use the notation $l^{-1}(t, s)$ to choose to which transition in the instant $s$ is transition $t$ mapped into. Assume that $t_{k_j} = t_r, t_r \in T$, for some $r$.

$$l^{-1}(t_r, s) = \begin{cases} t'^{\tau,m}_r & \text{if } s = i^{\mathcal{F}}_m, m \geq 1, i^{\mathcal{F}}_m = i^{F_{m(modn)}}_{\tau,s} \\ t'_{r,R} & \text{if } s = i^{\mathcal{F}}_{yn} + 1, y \geq 1 \\ t'_r & \text{otherwise} \end{cases}$$

Then set $\Omega = l^{-1}(t_{k_1}, 1) l^{-1}(t_{k_2}, 2) \ldots l^{-1}(t_{k_n}, n) \ldots$.

Our definition of $\Omega$ already allows us to state that if it is possible to fire $\Omega$ in $PN_{MACPN}$, then $\uparrow F'$ is reached infinitely often.

**Lemma 8.1.1.** *Let* $MACPN = (P, T, I, O, \mu_0, \sigma, \mathcal{F})$ *be a multiple accepting conditions Petri net with* $\sigma : T \mapsto \Sigma$ *the* $\lambda$*-free labeling function and* $\mathcal{F} = \{F_1, \ldots, F_n\}$ *with* $n > 1$ *and* $PN_{MACPN} = (P', T', I', O', \mu'_0, \sigma', F')$ *the witness of* $MACPN$*. Let* $l : T' \mapsto T$ *defined as above. Let* $s \in L^{\omega}_{\forall inf \cap}(MACPN)$*, and* $\alpha \in T^{\omega}$ *such that* $\sigma(\alpha) = s$*,* $\mu_0 \left|\alpha\right\rangle_{MACPN}$ *and* $\forall_{F_i \in \mathcal{F}} \ \omega Q(MACPN(\alpha)|_{\uparrow F_i}) = True$*, and let* $\Omega$ *defined as above. Then, if* $\mu'_0 \left|\Omega\right\rangle_{PN_{MACPN}}$*, then* $\omega Q(PN_{MACPN}(\Omega)|_{\uparrow F'}) = True$*.*

**Proof.** As $I'(t'_{i,R}) = I(t_i) + \sum_{i=1}^{n} \{p_F\}$, this means that if transitions of the type $t'_{i,R}$ exist in $\Omega$ and fire infinitely often, then the accepting markings $\uparrow (\underbrace{0, \ldots, 0}_{m}, \underbrace{0, \ldots, 0}_{n}, 0, n)$ are reached infinitely often too, because otherwise $t'_{i,R}$ could not fire. Why do transitions of the type $t'_{i,R}$ fire infinitely often? Because the set of indexes $\left\{i^{\mathcal{F}}_{yn} + 1\right\}_{y=1}^{+\infty}$ is infinite, so $\Omega$ has infinitely many transitions of the type $t'_{i,R}$. They can fire because that was our assumption. □

We now have to tackle the most difficult part. We need to prove that $\Omega$ can fire in $PN_{MACPN}$. In order to prove this statement, we will have to divide $\Omega$ into its small constituents. We then shall prove that its small constituents can fire, with some assumptions, and that the transitions linking the small constituents can also be fired, also assuming some needed properties. Then, it is easy to see that the all assumptions are verified.

**Lemma 8.1.2.** *Let* $MACPN = (P, T, I, O, \mu_0, \sigma, \mathcal{F})$ *be a multiple accepting conditions Petri net with* $\sigma : T \mapsto \Sigma$ *the* $\lambda$*-free labeling function and* $\mathcal{F} = \{F_1, \ldots, F_n\}$ *with* $n > 1$ *and* $PN_{MACPN} = (P', T', I', O', \mu'_0, \sigma', F')$ *the witness of* $MACPN$*.et* $s \in L^{\omega}_{\forall inf \cap}(MACPN)$*, and* $\alpha \in T^{\omega}$ *such that* $\sigma(\alpha) = s$*,* $\mu_0 \left|\alpha\right\rangle_{MACPN}$ *and* $\forall_{F_i \in \mathcal{F}} \ \omega Q(MACPN(\alpha)|_{\uparrow F_i}) = True$*, and let* $\Omega$ *defined as above. Then* $\Omega = \Omega_0 t'_{r_1,R} \ldots \Omega_n t'_{r_n,R} \ldots$*, where* $t'_{r_n,R}$ *is a infinite sequence of transitions, with each belonging to* $T'$*, and with* $\Omega_i \in (T' - A)^*$*, where* $A = \left\{t'_{r,R} : t'_{r,R} \in T'\right\}$*.*

**Proof.** The proof of this lemma is obvious from the definition of $\Omega$.

**Lemma 8.1.3.** *Consider the partition of* $\Omega$ *as above. Then, each* $\Omega_i$ *can be written as* $\Omega_i = \Omega_{i,1} t'^{\mu_{i,1},1}_{r_{i,1}} \ldots \Omega_{i,n} t'^{\mu_{i,n},n}_{r_{i,n}}$ *with* $\Omega_{i,j} \in B^*$*, where* $B = \{t'_r : t'_r \in T'\}$*.*

**Proof.** The proof of this lemma is slightly less obvious than the earlier one; we just have to notice that the definition of $\Omega$ allows us to admit that the markings in each of the $F_j$ are reached in numerical order.

**Lemma 8.1.4.** *Consider the partition of a given* $\Omega_i = \Omega_{i,1} t'^{\mu_{i,1},1}_{r_{i,1}} \ldots \Omega_{i,n} t'^{\mu_{i,n},n}_{r_{i,n}}$*. Assume that the marking at the start of each* $\Omega_{i,j}$ *in* $PN_{MACPN}$ *is* $\mu'_{i,j,ini}$*. Assume also that* $pr_{Ori}(\mu'_{i,j,ini}) = \mu_{i,j,ini}$*, where* $\mu_{i,j,ini}$ *is the marking in* $MACPN$ *at the start of* $\Omega_{i,j}$*. Finally assume that at the start of each* $\Omega_{i,j}$ *there is at least one token in* $p_R$*. Then* $\mu'_{i,j,ini} \left|\Omega_{i,j}\right\rangle_{PN_{MACPN}} = \mu'_{i,j,end}$ *is such that:*

- $pr_{Ori}(\mu'_{i,j,end}) = \mu_{i,j,end} = pr_{Ori}(\mu'_{i,j,ini}) \left|l(\Omega_{i,j})\right\rangle_{MACPN}$,
- $pr_{Sup}(\mu'_{i,j,ini}) = pr_{Sup}(\mu'_{i,j,end})$.

**Proof.** Let us assume that the length of $\Omega_{i,j}$ is at least one, without loss of generality, since otherwise it is obvious. The following proof will be done by induction on the length of $\Omega_{i,j}$.

In the basis case, $\Omega_{i,j} = t'_r$ for some $t'_r \in T'$. Can $t'_r$ fire? Since $I'(t'_r) = I(t_r) + \{p_R\}$, and we know that $pr_{Ori}(\mu'_{i,j,ini}) = \mu_{i,j,ini}$, $t_r$ is fireable in $MACPN$ and there is at least one token in $p_R$, we can conclude that $I'(t'_r) \subseteq \mu'_{i,j,ini}$, and so it can fire. As it fires:

$$\begin{aligned}
\mu'_{i,j,end} &= \mu'_{i,j,ini} - I'(t'_r) + O'(t'_r) \\
&= \mu'_{i,j,ini} - I(t_r) - \{p_R\} + O(t_r) + \{p_R\} \\
&= \mu'_{i,j,ini} - I(t_r) + O(t_r) \\
pr_{Ori}(\mu'_{i,j,end}) &= pr_{Ori}(\mu'_{i,j,ini}) - I(t_r) + O(t_r) \\
&= \mu_{i,j,end} \\
pr_{Sup}(\mu'_{i,j,end}) &= pr_{Sup}(\mu'_{i,j,ini})
\end{aligned}$$

For the step case, $\Omega_{i,j} = \Pi_{i,j} t'_r$, where $\Pi_{i,j}$ has one transition less than $\Omega_{i,j}$. Now $\mu'_{i,j,ini} |\Omega_{i,j}\rangle_{PN_{MACPN}} = (\mu'_{i,j,ini} |\Pi_{i,j}\rangle_{PN_{MACPN}}) |t'_r\rangle_{PN_{MACPN}}$. With this equality in mind, it is simple to see that we effectively can fire $\Omega_{i,j}$ repeating the process used for the basis case, since we can state exactly the same assumptions about $\mu'_{i,j,ini}$ as about $(\mu'_{i,j,ini} |\Pi_{i,j}\rangle_{PN_{MACPN}})$. □

We will now prove that all $\Omega_i$ can fire, assuming that the markings where they begin obey $pr_{Ori}(\mu') = \mu$, and that $pr_{Sup}(\mu') = \sum_{i=1}^{n} \{pr_R\} + \bigcup_{i=1}^{n} \{p_{RF_i}\}$, or in other words, the marking $\mu'$ has supervisor places correctly marked.

**Lemma 8.1.5.** *Consider the partition of $\Omega$ as above. Let $\mu'_{i,ini}$ be the initial marking before the firing of $\Omega_i$. Assume that $pr_{Ori}(\mu'_{i,ini}) = \mu_{i,ini}$, where $\mu_{i,ini}$ is the marking before the firing of $l(\Omega_i)$. Assume also that $pr_{Sup}(\mu'_{i,ini}) = \sum_{i=1}^{n} \{pr_R\} + \bigcup_{i=1}^{n} \{p_{RF_i}\}$. Then, $\mu'_{i,ini} |\Omega_i\rangle_{PN_{MACPN}}$ and the marking $\mu'_{i,end}$ obtained verifies $pr_{Ori}(\mu'_{i,end}) = \mu_{i,end}$ and $pr_{Sup}(\mu'_{i,end}) = \sum_{i=1}^{n} \{p_F\}$.*

**Proof.** Consider the partition of a general $\Omega_i = \Omega_{i,1} t'^{\mu_{i,1},1}_{r_{i,1}} \ldots \Omega_{i,n} t'^{\mu_{i,n},n}_{r_{i,n}}$. We will prove that $\mu'_{i,ini} |\Omega_i\rangle_{PN_{MACPN}}$ by induction on $n$. The basis case for $n = 1$ contains again almost all the information needed, and it is very similar to the proof for $n > 1$, with some small considerations regarding the supervisor places.

Consider then that $\Omega_i = \Omega_{i,1} t'^{\mu_{i,1},1}_{r_{i,1}}$. Now, we are in the conditions of the above lemma, and so, we can conclude that $\mu'_{i,ini} |\Omega_{i,1}\rangle_{PN_{MACPN}}$. We will call the marking reached, $\mu'_{i,ini} |\Omega_{i,1}\rangle_{PN_{MACPN}}$, $\mu'_{i,1,mid}$. From the above Lemma 8.1.4, we can also conclude that $pr_{Ori}(\mu'_{i,1,mid}) = \mu_{i,1,mid}$, the corresponding marking in $MACPN$, and that $pr_{Sup}(\mu'_{i,1,mid}) = pr_{Sup}(\mu'_{i,ini})$. We will admit that $t'^{\mu_{i,1},1}_{r_{i,1}}$ is $t'^{\mu,1}_r \in T'$ for simplicity of notation. Can $t'^{\mu,1}_r$ fire in $PN_{MACPN}$? First, we need to realize some conditions about $\mu'_{i,1,mid}$. We know that $pr_{Ori}(\mu'_{i,1,mid}) = \mu_{i,1,mid}$. We now need to relate this information with what happens in $MACPN$. Firstly, we know that $\mu$ is reached in $\mu_{i,1,end}$, where $\mu_{i,1,end}$ is the marking reached after firing $l(\Omega_{i,1} t'^{\mu,1}_r)$ in

$MACPN$. Secondly, we know that $t_r$ fires, and so this means that $\mu_{i,1,end} = B + (\mu - O(t_r)) + O(t_r)$.

$$\begin{aligned}
\mu_{i,1,end} &= \mu_{i,1,mid} - I(t_r) + O(t_r) \\
B + (\mu - O(t_r)) + O(t_r) &= \mu_{i,1,mid} - I(t_r) + O(t_r) \\
B + (\mu - O(t_r)) &= \mu_{i,1,mid} - I(t_r) \\
B + (\mu - O(t_r)) + I(t_r) &= \mu_{i,1,mid}
\end{aligned}$$

Now, we know that there exists only one token in $p_R$, and that there also exists one token in $p_{RF_1}$. This is information enough to conclude that $t_r'^{\mu,1}$ can fire, since $I'(t_r'^{\mu,1}) = (\mu - O(t_r)) + I(t_r) + \{p_R\} + \{p_{RF_1}\} \subseteq \mu'_{i,1,mid}$.

$$\begin{aligned}
\mu'_{i,1,end} &= pr_{Ori}(\mu'_{i,1,mid}) + pr_{Sup}(\mu'_{i,1,mid}) - I'(t_1'^{\mu,1}) + O'(t_1'^{\mu,1}) \\
&= pr_{Sup}(\mu'_{i,1,mid}) + B + (\mu - O(t_r)) + I(t_r) - ((\mu - O(t_r)) + I(t_r) + \{p_R\} + \{p_{RF_1}\}) \\
&\quad + (\mu - O(t_r)) + O(t_r) + \{p_F\} \\
&= (pr_{Sup}(\mu'_{i,1,mid}) - \{p_R\} - \{p_{RF_1}\}) + B + (\mu - O(t_r)) + O(t_r) \\
pr_{Ori}(\mu'_{i,1,end}) &= B + (\mu - O(t_r)) + O(t_r) \\
&= \mu_{i,1,end} \\
pr_{Sup}(\mu'_{i,1,end}) &= pr_{Sup}(\mu'_{i,1,mid}) - \{p_R\} - \{p_{RF_i}\} + \{p_F\}
\end{aligned}$$

Notice now we can conclude that both conditions of the lemma are satisfied, since $pr_{Sup}(\mu'_{i,1,mid}) = \{p_R\} + \{p_{RF_1}\}$.

For the proof of the step, we just have to repeat $n$ times this same argument, noticing that we will also start with $n$ tokens in $\{p_R\}$ and one token in each of the $p_{RF_i}$, meaning that will end up loosing, after firing all $\Omega_{i,j}$ with the linking $t_r'^{\mu,j}$ transitions, the $n$ tokens in $p_R$ and all of the tokens in each of the $p_{RF_i}$, gaining however $n$ tokens in $p_F$.

We can conclude then that each of the $\Omega_i$ will be able to fire, assuming the conditions of the lemma. □

After all these lemmas, we just need to prove that the links between each of the $\Omega_i$ can fire. After proving that they can effectively fire, we show that, after all, $\Omega$ can fire, and so $L^{\omega}_{\forall inf \cap}(MACPN) \subseteq L^{\omega}_{inf \cap}(PN_{MACPN})$.

**Lemma 8.1.6.** *Let $MACPN = (P, T, I, O, \mu_0, \sigma, \mathcal{F})$ be a multiple accepting conditions Petri net with $\sigma : T \mapsto \Sigma$ the $\lambda$-free labeling function and $\mathcal{F} = \{F_1, \ldots, F_n\}$ with $n > 1$ and $PN_{MACPN} = (P', T', I', O', \mu'_0, \sigma', F')$ the witness of $MACPN$. Let $l : T' \mapsto T$ defined as above. Let $s \in L^{\omega}_{\forall inf \cap}(MACPN)$, and $\alpha \in T^{\omega}$ such that $\sigma(\alpha) = s$, $\mu_0 \left|\alpha\right\rangle_{MACPN}$ and $\forall_{F_i \in \mathcal{F}}$ $\omega Q(MACPN(\alpha)|_{\uparrow F_i}) = True$, and let $\Omega$ defined as above. Then $\mu'_0 \left|\Omega\right\rangle_{PN_{MACPN}}$.*

**Proof.** First, consider the partitioning of $\Omega$, $\Omega = \Omega_0 t'_{r_1,R} \ldots \Omega_n t'_{r_n,R} \ldots$. We will show that $\Omega^i = \Omega_0 t'_{r_1,R} \ldots \Omega_i t'_{r_i,R}$ can fire, with $pr_{Ori}(\mu'_0 \left|\Omega^i\right\rangle_{PN_{MACPN}}) = \mu_0 \left|l(\Omega^i)\right\rangle_{MACPN}$ and $pr_{Sup}(\mu'_0 \left|\Omega^i\right\rangle_{PN_{MACPN}}) =$

$\sum_{i=1}^{n}\{p_R\}+\bigcup_{i=1}^{n}\{p_{RF_i}\}$ by induction on $i$, for all $i$. Moreover, the markings $\mu'_{i,ini}$ are the markings before the firing of $\Omega_i$. Either way, $\mu'_0$ coincides with $\mu'_{0,ini}$. First, consider the basis case, with $i=1$.

Notice that $\mu'_{0,ini}$ satisfies the condition of Lemma 8.1.5, so we can assume that $\mu'_{0,ini}\left|\Omega_0\right\rangle_{PN_{MACPN}}=\mu'_{0,mid}$. Now, can $t'_{r_1,R}$ fire? We do know that $pr_{Ori}(\mu'_{0,mid})=\mu_{0,mid}$ allows the firing of $t_r$, where $\mu_{0,mid}$ is the marking reached after firing $l(\Omega_0)$; furthermore we also know, due to the conclusions of Lemma 8.1.5, that $pr_{Sup}(\mu'_{0,mid})=\sum_{i=1}^{n}\{p_F\}$. Since $I'(t'_{r,R})=I(t_r)+\sum_{i=1}^{n}\{p_F\}$, we conclude that $t'_{r,R}$ can effectively fire. Then:

$$\begin{aligned}
\mu'_{0,end} &= \mu'_{0,mid}-I'(t'_{r,R})+O'(t'_{r,R})\\
&= pr_{Ori}(\mu'_{0,mid})+pr_{Sup}(\mu'_{0,mid})-(I(t_r)+\sum_{i=1}^{n}\{p_F\})+\\
&+O(t_r)+\sum_{i=1}^{n}\{p_R\}+\bigcup_{i=1}^{n}\{p_{RF_i}\}\\
&= pr_{Ori}(\mu'_{0,mid})-I(t_r)+O(t_r)+\sum_{i=1}^{n}\{p_R\}+\bigcup_{i=1}^{n}\{p_{RF_i}\}\\
pr_{Ori}(\mu'_{0,end}) &= pr_{Ori}(\mu'_{0,mid})-I(t_r)+O(t_r)\\
&= \mu_{0,mid}-I(t_r)+O(t_r)\\
&= \mu_{0,end}\\
pr_{Sup}(\mu'_{0,end}) &= \sum_{i=1}^{n}\{p_R\}+\bigcup_{i=1}^{n}\{p_{RF_i}\}
\end{aligned}$$

We conclude now the basis case, proving that $\mu'_0\left|\Omega^1\right\rangle_{PN_{MACPN}}$, with the correct characterizations of the markings. Also notice, to dispel even more confusions, that $\mu'_{0,end}=\mu'_{1,ini}$.

Let us now assume that $\mu'_0\left|\Omega^k\right\rangle_{PN_{MACPN}}=\mu'_{k+1,ini}$, that $pr_{Ori}(\mu'_{k+1,ini})=\mu_{k+1,ini}$ and that $pr_{Sup}(\mu'_{k+1,ini})=\sum_{i=1}^{n}\{p_R\}+\bigcup_{i=1}^{n}\{p_{RF_i}\}$. We are again in the conditions to apply Lemma 8.1.5, and so we conclude that we can effectively fire from $\mu'_{k+1,ini}\left|\Omega_{k+1}\right\rangle_{PN_{MACPN}}=\mu'_{k+1,mid}$. Again, using the conclusions of Lemma 8.1.5, we know that $pr_{Ori}(\mu'_{k+1,mid})=\mu_{k+1,ini}\left|l(\Omega_{k+1})\right\rangle_{MACPN}$, and that $pr_{Sup}(\mu'_{k+1,mid})=\sum_{i=1}^{n}\{p_F\}$. Let $t'_{r_k,R}=t'_{r,R}$, for some $t'_{r,R}\in T'$. Can $t'_{r,R}$ fire in $\mu'_{k+1,mid}$? Since

$I'(t'_{r,R}) = I(t_r) + \sum_{i=1}^{n} \{p_F\}$, we conclude that $t'_{r,R}$ can effectively fire. Then:

$$\begin{aligned}
\mu'_{k+1,end} &= \mu'_{k+1,mid} - I'(t'_{r,R}) + O'(t'_{r,R}) \\
&= pr_{Ori}(\mu'_{k+1,mid}) + pr_{Sup}(\mu'_{k+1,mid}) - (I(t_r) + \sum_{i=1}^{n} \{p_F\}) + \\
&+ O(t_r) + \sum_{i=1}^{n} \{p_R\} + \bigcup_{i=1}^{n} \{p_{RF_i}\} \\
&= pr_{Ori}(\mu'_{k+1,mid}) - I(t_r) + O(t_r) + \sum_{i=1}^{n} \{p_R\} + \bigcup_{i=1}^{n} \{p_{RF_i}\} \\
pr_{Ori}(\mu'_{k+1,end}) &= pr_{Ori}(\mu'_{k+1,mid}) - I(t_r) + O(t_r) \\
&= \mu_{k+1,mid} - I(t_r) + O(t_r) \\
&= \mu_{k+1,end} \\
pr_{Sup}(\mu'_{k+1,end}) &= \sum_{i=1}^{n} \{p_R\} + \bigcup_{i=1}^{n} \{p_{RF_i}\}
\end{aligned}$$

So we end up concluding that $\Omega^{k+1}$ can fire, again preserving the basic properties of the markings. We can finally conclude that $\Omega$ can fire in $PN_{MACPN}$. □

**Proposition 8.1.2.** *Let $MACPN = (P, T, I, O, \mu_0, \sigma, \mathcal{F})$ be a multiple accepting conditions Petri net with $\sigma : T \mapsto \Sigma$ the $\lambda$-free labeling function and $\mathcal{F} = \{F_1, \ldots, F_n\}$ with $n > 1$ and $PN_{MACPN} = (P', T', I', O', \mu'_0, \sigma', F')$ the witness of $MACPN$. Then $L^{\omega}_{\forall inf\cap}(MACPN) \subseteq L^{\omega}_{inf\cap}(PN_{MACPN})$.*

**Proof.** Let $s \in L^{\omega}_{\forall inf\cap}(MACPN)$. Then there exists $\alpha \in T^{\omega}$ such that $\mu_0 \,|\alpha\rangle_{MACPN}$ and $\forall_{F_i \in \mathcal{F}} \ \omega Q(MACPN(\alpha)|_{\uparrow F_i}) = True$. Consider $\Omega$ defined as in Definition 8.1.4. Then, $\Omega \in T'^{\omega}$, by definition; $\sigma'(\Omega) = s$, since $\sigma'(\Omega) = \sigma(l(\Omega)) = \sigma(\alpha) = s$. Furthermore $\mu'_0 \,|\Omega\rangle_{PN_{MACPN}}$, and using Lemma 8.1.1, we can conclude that $\omega Q(PN_{MACPN}(\Omega)|_{\uparrow F'}) = True$, and so $s \in L^{\omega}_{inf\cap}(PN_{MACPN})$. □

And finally:

**Theorem 8.1.1.** *Let $MACPN = (P, T, I, O, \mu_0, \sigma, \mathcal{F})$ be a multiple accepting conditions Petri net with $\sigma : T \mapsto \Sigma$ the $\lambda$-free labeling function and $\mathcal{F} = \{F_1, \ldots, F_n\}$ with $n > 1$ and $PN_{MACPN} = (P', T', I', O', \mu'_0, \sigma', F')$ the witness of $MACPN$. Then $L^{\omega}_{\forall inf\cap}(MACPN) = L^{\omega}_{inf\cap}(PN_{MACPN})$ and so $P_{inf\cap} = P_{\forall inf\cap}$.*

It is then clear that the class of Petri net $\omega$-languages with a Büchi-like accepting criterion is equivalent to the class of multiple accepting conditions Petri net $\omega$-languages, since it is clear that any Petri net is a multiple accepting conditions Petri net with $\#\mathcal{F} = 1$.

## 8.2 Adaptation of the proposed methodology

We can then use our new concept to adapt the proposed methodology in Chapter 5. We will avoid transforming a generalized Büchi automaton into a finite state automaton with Büchi acceptance criterion, transforming it instead into a multiple accepting conditions Petri net. The first adaptation to the methodology to be made is in Step 5.1.4; since we will not perform Step 5.1.5, we need to trim the automaton in this phase. Use Definition 1.2.4. Afterwards, use the two following steps.

**Step 8.2.1** (Transformation of a generalized Büchi automaton into a multiple accepting conditions Petri net)**.** As described above, the designer should make good use of Proposition 2.2.4 to create a structure defined in Definition 8.1.1. Assume that the structure produced as output is the multiple accepting conditions Petri net $MACPN_\phi$. This structure is such that $L^\omega_{\forall inf\cap}(MACPN_\phi) = L^\omega_{\forall inf\cap}(GBA''_\phi)$, and it is trimmed; while we have not defined what is a trimmed Petri net, much less for MACPNs, the concept is still the same: all places can eventually be marked, all transitions can eventually fire, and from all reachable markings it is possible to reach at least a marking in each of the sets of the accepting markings.

**Step 8.2.2** (From a multiple accepting conditions Petri net to a Petri net)**.** The designer can now apply Theorem 8.1.1; although it contemplates all possible Petri nets, it is far easier to apply to a finite state automaton disguised as a Petri net. Let $\overline{PN_\phi}$ be the result of the application of Theorem 8.1.1. Then $L^\omega_{inf\cap}(\overline{PN_\phi}) = L^\omega_{\forall inf\cap}(MACPN_\phi)$. Let us prove that $\overline{PN_\phi}$ possesses properties similar to $PN_\phi$, so that the designer is assured that the only effective difference between the two routes is the size of the Petri net model.

First of all, let us show that if $L^\omega_{inf\cap}(\overline{PN_\phi}) \neq \emptyset$, then if $\sigma \in L_G(\overline{PN_\phi})$, there exists $\sigma' \in \Sigma^\omega$ such that $\sigma\sigma' \in L^\omega_{inf\cap}(\overline{PN_\phi})$. Consider $s \in L_G(\overline{PN_\phi})$ and let $\overline{PN_\phi} = (P, T, I, O, \mu_0, \sigma, F)$, given accordingly to Theorem 8.1.1. Then there exists $\alpha \in T^*$ such that $\sigma(\alpha) = s$, $\mu_0 \,|\alpha\rangle_{\overline{PN_\phi}} \in\uparrow (\underbrace{0, \ldots, 0}_{\#P-n-2}, \underbrace{0, \ldots, 0}_{n}, 0, n)$, where $n = \#\mathcal{F}$. Please recall from the discussion of Theorem 8.1.1 that the first $\#P - n - 2$ are the places from $MACPN_\phi$, the $n$ places following them are supervisor places that control whether each accepting condition was reached; the following place controls how many accepting conditions have yet to be reached, and the last place controls when should the supervisor places be reset.

Now, using the information that $MACPN_\phi$ was built from a trimmed generalized Büchi automaton we can conclude several things:

- The first $\#P - n - 2$ places always have one token.
- Since each of these $\#P - n - 2$ places represents a state in the generalized Büchi automaton, and the generalized Büchi automaton was trimmed, all places have at least one transition as input and one transition as output.
- All transitions are such that if they fire the markings in the original $MACPN_\phi$ are exactly the same as the markings in $\overline{PN_\phi}$, if we project these last markings to the first $\#P - n - 2$ places.

- Since all states were co-reachable in one or more steps to each of the sets of accepting states, we can conclude that from any reachable marking the set of markings $\uparrow (0, \ldots, 0, n)$ is reachable.

We can then conclude that if $s \in L_G(\overline{PN_\phi})$ there exists $s' \in \Sigma^\omega$ such that $s.s' \in L^\omega_{inf\cap}(PN_\phi)$. We now have to prove that $\overline{PN_\phi}$ is completely fulfilling. Just looking at the transformation from Theorem 8.1.1 we know that we only reach the set of accepting markings when all the sets of accepting markings originating from $MACPN_\phi$ and consequently from $GBA'''_\phi$ and ultimately from $\phi$ are reached. And so, $\overline{PN_\phi}$ is completely fulfilling. It is also non-blocking, a conclusion reached from the fact that all reachable markings are co-reachable.

We are now ready to continue into Step 5.1.7.

We will make a small efficiency analysis in order to test the efficiency of the alternative proposed. This technique assumes that the designer has constructed the generalized Büchi automaton $GBA''_\phi$ and he now has to choose between both techniques. In most cases, as we will see, transforming the generalized Büchi automaton into a Büchi automaton and then into a Petri net provides better results. However, there are some rare circumstances that may favor the translation of the generalized Büchi automaton into a multiple accepting conditions Petri net, specially if the number of places is the relevant factor in future transformations. Our efficiency analysis will try to calculate the worst case scenarios of the size of the structures involved. Notice, furthermore, that we have not presented optimized algorithms; the following should be only used as a basic reference.

**Performance Analysis.** We will initially concentrate our analysis in the number of places and transitions. The total number of places and transitions are an important size measurement, since many Petri net algorithms depend directly on this measurement of size. We will then analyze a less used measure that deals with ramification. Ramification is the total number of arcs ($n$-weighted arcs will count as one) present in the Petri net; although it is less used than the number of places and transitions, it is a very useful measurement of the complexity of a Petri net graph; a Petri net intended to be used as a graphical modeling tool should have low ramification, or otherwise it is difficult to understand it.

Consider the generalized Büchi automaton $GBA''_\phi = (S, \Sigma, \delta, s_0, \mathcal{F})$, with $\mathcal{F} = \{F_1, \ldots, F_n\}$. We will assume the worst case scenario where the trimming operation that transform $GBA''_\phi$ into $GBA'''_\phi$ outputs the same generalized Büchi automaton. And so when applying Step 8.2.1, if the result is $MACPN_\phi = (P_1, T_1, I_1, O_1, \mu_0^1, \sigma_1, \mathcal{F}_1)$ it is easy to see that $\#P_1 = \#S$ and $\#T_1 = \#\delta$. When translating the $MACPN_\phi$ into $\overline{PN_\phi}$, calculating the output Petri net $\overline{PN_\phi} = (P_2, T_2, I_2, O_2, \mu_0^2, \sigma_2, F^2)$ we can then see that $\#P_2 = \#P_1 + \#\mathcal{F}_1 + 2$. The total number of transitions is $\#T_2 = \#T_1 + \#T_1 + \sum_{i=1}^{\#\mathcal{F}_1} \#T_1 \times \#(\mathcal{F}_1)_i = \#T_1(2 + \sum_{i=1}^{\#\mathcal{F}_1} \#(\mathcal{F}_1)_i)$.

Let us then consider again $GBA''_\phi$. Following Step 5.1.5, the first operation transforms $GBA''_\phi$ into $BA_\phi = (S_1, \Sigma, \delta_1, s_0^1, F_1)$; we can then calculate $\#S_1 = \#S \times \#\mathcal{F}$ and $\#\delta_1 = \#\mathcal{F} \times \#\delta$. Now, we will again assume that the trimming operation is useless, outputing $BA'_\phi = BA_\phi$. Finally, following Step 5.1.6, we obtain a

Petri net $PN_\phi = (P_3, T_3, I_3, O_3, \mu_0^3, \sigma_3, F_3)$ such that $\#P_3 = \#S_1$ and $\#T_3 = \#\delta_1$.

Assuming a place occupies the same size as a transition, the size of $\overline{PN_\phi}$ and $PN_\phi$ is:

$$\begin{aligned}
\#\overline{PN_\phi} &= \#P_1 + \#\mathcal{F}_1 + 2 + \#T_1(2 + \sum_{i=1}^{\#\mathcal{F}_1} \#(\mathcal{F}_1)_i) \\
&= \#S + \#\mathcal{F} + 2 + \#\delta(2 + \sum_{i=1}^{\#\mathcal{F}} \#(\mathcal{F})_i) \\
\#PN_\phi &= \#S_1 + \#\delta_1 \\
&= \#S \times \#\mathcal{F} + \#\mathcal{F} \times \#\delta
\end{aligned} \tag{8.1}$$

If we assume $\#(\mathcal{F}_i) = 1, \forall_i$, then the difference in size is $\#S \times \#\mathcal{F} - \#S - \#\mathcal{F} - 2 - 2 \times \#\delta$. While the difference can be negative, some special conditions must occur for that to happen: $\#\mathcal{F}$ has to be very large, and the ramification in the original generalization Büchi automaton cannot be too large.

When analyzing the ramification created by the transformations, it is easy to see that the ramification in $PN_\phi$ is simply $2 \times \#T_3 = 2 \times \#\delta_1 = 2 \times \#\mathcal{F} \times \#\delta$. The ramification of $\overline{PN_\phi}$ is much larger, not only because $\overline{PN_\phi}$ has many more transitions, but also because the extra arcs introduced linking the supervisor places to the transitions. The ramification in $\overline{PN_\phi}$ is less than:

$$\#\delta \times 4 + \#\delta \times 5 + 7\#\delta \sum_{i=1}^{\#\mathcal{F}} \times \#(\mathcal{F})_i$$

This estimate of the ramification was done by directly using the algorithm described in 8.1.1 and bounding the terms $\#(\mu - O(t_i) \leq 1$. There is no doubt whatsoever that if ramification is the primary concern of the designer, he should transform the generalized Büchi automaton into a Büchi automaton.

So, as a conclusion of this simple analysis we can say there are three different circumstances that counsel different routes:

- if the user is only worried about the number of places of final system, for instance, if he needs to run further algorithms more heavily dependent on the number of places than on transitions, then he should choose the MACPN route,
- if the user uses the typical approach of attributing the same degree of importance to places and transitions, then he almost surely should use the GBA to BA route,
- and finally, if the user is also interested in the ramification produced, he should definitely choose the GBA to BA algorithms, since the MACPN algorithm regarding transitions, at least in the current state, output too much ramification.

# Conclusions and Future Work

The dissertation main objective was the extension of the work by Lacerda and Lima [20] to Petri nets. This extension was achieved in two different directions: not only we were able to achieve similar results for a more expressive class, unbounded Petri nets, but also our specifications can be richer, since we can now use QPLTL to express our desirable properties.

Concerning the main line of work, we presented a procedure capable of enforcing QPLTL specifications into a possibly unbounded Petri net that still outputs a Petri net. Furthermore, we discussed a small algorithm to test controllability of the P-Type Language of the obtained Petri net, while giving conditions to relate controllability conditions of the various G-Type Languages to the basic controllability test of the P-Type Language. Our initial line of work allow us to deal with non-deterministic systems and specifications; since in some cases the designer may not be interested in non-deterministic systems, we have suggested a way of adapting the proposed method in order to obtain a deterministic system. This small adaptation can be useful in cases where we wish to use the obtained system as a plan.

Furthermore, when analyzing $\omega$-Languages accepted by Petri nets, we extended the concept of a generalized Büchi automaton, and proved that the extension is equally expressive to appropriate Petri net $\omega$-Languages, a property shared by the generalized Büchi automaton concept. Even if the new concept is equally expressive, we should not forget that it is much more synthetic, as some properties would be too cumbersome to write using classical definitions of Petri net $\omega$-Languages. We also improved some results of Yamasaki[39], using some proposed definitions more appropriate to his results. This extension could have had more usefulness in the main line of work, if we had chosen to optimize the construction that translates the new extension into a regular Petri net.

This work has left us with many questions that we would like to see answered in the future. Many of the open questions are about relationships of classes of Petri net $\omega$-Languages. Are all the results developed by Yamasaki [39] correct with the new definition of $L^{\omega}_{inf\cap}(PN)$? What are the relationships between the classes of Languages Yamasaki unwittingly described, our proposals, and Valk's initial definitions? Are the proposed definitions still related with the Borel Hierarchy? Can multiple accepting conditions Petri nets be defined with a L-Type accepting condition in mind?

Other questions raised are of a more practical nature. The proposed controllability conditions are hard to check. Can they be altered? Is the proposed method applicable to real world cases, since after all, we only presented academic examples? If not, why is it so? Is it because it depends too much on a good algorithm to dispose of uninteresting places and transitions, since the intersection algorithm may produce many of these places and transitions? How easy is it to use the effective extra expressibility brought by the use of QPLTL specifications? And, finally, is it possible to develop a system capable of directly acting upon both markings and transitions sufficiently close to our natural thought processes?

# Bibliography


[1] T. Antunes. Comportamentos relacionais para robots futebolistas. Master's thesis, I.S.T, 2007.

[2] B. Berthomieu, P. Ribet, and F. Vernadat. The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14), July 2004.

[3] H. Carstensen and R. Valk. Infinite behaviour and fairness in Petri nets. In G. Rozenberg, H. J. Genrich, and G. Roucairol, editors, *European Workshop on Applications and Theory in Petri Nets*, volume 188 of *Lecture Notes in Computer Science*, pages 83–100. Springer, 1984.

[4] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[5] H. Costelha and P. Lima. Modelling, analysis and execution of robotic tasks using Petri nets. In *IROS*, pages 1449–1454. IEEE, 2007.

[6] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1:1115–138, 1971.

[7] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.

[8] J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994.

[9] K. Etessami. Stutter-invariant languages, $\omega$-automata, and temporal logic. In *CAV'99: Computer Aided Verification, 11th International Conference*, pages 236–248, 1999.

[10] T. French and M. Reynolds. A sound and complete proof system for QPTL. In P. Balbiani, N.-Y. Suzuki, F. Wolter, and M. Zakharyaschev, editors, *Advances in Modal Logic*, pages 127–148. King's College Publications, 2002.

[11] D. Giannakopoulou and F. Lerda. Efficient translation of LTL formulae into Büchi automata. Technical report, Research Institute for Advanced Computer Science, June 01 2001.

[12] A. Giua and F. DiCesare. On the existence of Petri net supervisors. In *Proc. 31st IEEE Conf. on Decision and Control*, pages 3380–3385. IEEE Control Systems Society, Dec. 1992.

[13] M. Hack. Decidability questions for Petri nets. Technical Report TR-161, MIT Lab. for Comp. Sci., June 1976.

[14] A. Holt and F. Commoner. Events and conditions. In *ACM Conference Record*, New York, 1970. ACM Press.

[15] A. W. Holt et al. Final report of the information system theory project. Technical Report RADC-TR-68-305, Rome Air Development Center, 1968.

[16] J. Hopcroft and J. Ullman. *Introduction to Automata and Formal Languages*. Addison-Wesley, Reading, MA, 2001.

[17] J.R. Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *Proceedings of the 1960 International Congress of Logic, Methodology and Philosophy of Science*, pages 1–12. Stanford University Press, 1960. June.

[18] Y. Kesten and A. Pnueli. Complete proof system for QPTL. *JLC: Journal of Logic and Computation*, 12, 2002.

[19] R. Kumar and L. E. Holloway. Supervisory control of deterministic petri nets with regular specification languages, 1996.

[20] B. Lacerda and P. Lima. Linear-time temporal logic control of discrete event models of cooperative robots. *Journal of Physical Agents*, 2(1), 2008.

[21] L. Landweber. Decision problems for $\omega$-automata. *Math. Systems Theory*, 3:376–384, 1969.

[22] M. Parigot and M. Pelz. A logical approach of Petri net languages. *Theoretical Computer Science*, 39:155–169, 1985.

[23] E. W. Mayr. An algorithm for the general Petri net reachability problem. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 238–246, New York, NY, USA, 1981. ACM.

[24] J. Moody and P. Antsaklis. *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[25] M. Mukund. Finite-state automata on infinite inputs, July 09 1996.

[26] P. Palamara, V. Ziparo, L. Iocchi, D. Nardi, and P. Lima. Teamwork design based on Petri net plans, Aug. 31 2009.

[27] R. J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.

[28] D. Perrin and J.-É. Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, Amsterdam, 2004.

[29] J. Peterson. *Petri Net Theory and the Modelling of Systems*. Prentice-Hall, Englewoods Cliffs, New Jersey, 1981.

[30] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, Bonn, Germany, 1962. (In German).

[31] A. Pnueli. The temporal logic of programs. *Symposium on Foundations of Computer Science*, 0:46–57, 1977.

[32] P. Ramadge and W. Wonham. Supervisory control of a class of discrete event processes. *Siam J. Control and Optimization*, 25(1), 1987.

[33] P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[34] A. Sistla. *Theoretical Issues in the Design and Verification of Distributed Systems*. PhD thesis, Harvard University, 1983.

[35] A. Sistla, M. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[36] K. Wagner. On $\omega$-regular sets. *Information and Control*, 43(2):123–177, Nov. 1979.

[37] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.

[38] P. Wolper. Constructing automata from temporal logic formulas: A tutorial. *Lecture Notes in Computer Science*, 2090:261–277, 2001.

[39] H. Yamasaki. Logical characterization of Petri net $\omega$-languages. Technical report, Universität Halle-Wittenber, 1999.

# Appendix

This proposition is discussed in page 14.

**Proposition 8.2.1.** *Let $GN = (S, \Sigma, s_0, \delta, \mathcal{F})$ be a generalized Büchi automaton, where $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$. Then there exists a Büchi automaton $N' = (S', \Sigma, \delta', s'_0, F')$ such that $L^{\omega}_{\forall inf \cap}(GN) = L^{\omega}_{inf \cap}(N')$.*

**Proof.** Let $S'$, $\delta'$,$s'_0$ and $F'$ defined as below.

$$S' = S \times \{1, 2, \ldots, k\}.$$

$$s'_0 = (s_0, 1).$$

$$F' = F_k \times \{k\}.$$

$$((s,j), a, (t,i)) \in \delta' \text{if } (s,a,t) \in \delta \text{ and } \begin{cases} i = j & \text{if } s \notin F_j \\ i = (j \ mod \ k) + 1 & \text{if } s \in F_j \end{cases}$$

The states of $N'$ are the states of $GN$ marked by an integer in the range $[1, k]$. The markings do not change unless one goes through a state in $F_j$, where $j$ is the current value of the mark. The mark will be incremented ($j \leq k$) or reset($j = k$). If one repeatedly cycles through all the marks, which is the only way to reach $F'$ infinitely often, then, necessarily, all the sets from $\mathcal{F}$ are visited infinitely often. On the other way, if it is possible to visit infinitely often all the sets of $\mathcal{F}$ in $GN$, it is possible, as discussed above, to visit them in order, and hence to infinitely often go through $F^{'}$ in $N^{'}$. □

This proposition is discussed in page 24.

**Proposition 8.2.2.** *Let $L_1$ and $L_2$ be two G-type languages with $\lambda$-free labeling functions. Then $L_1 \cap L_2$ is a G-Type language with a $\lambda$-free labeling function.*

**Proof.** Let $PN_1 = (P_1, T_1, I_1, O_1, \mu_1, \sigma_1, F_1)$ and $PN_2 = (P_2, T_2, I_2, O_2, \mu_2, \sigma_2, F_2)$ such that $L_G(PN_1) = L_1$ and $L_G(PN_2) = L_2$. We shall prove that there exists $PN' = (P', T', I', O', \mu', \sigma', F')$ where $\sigma'$ is a $\lambda$-free labeling function, such that $L_G(PN') = L_1 \cap L_2$. Consider $PN' = (P', T', I', O', \mu', \sigma', F')$ defined as follows:

$$P' = P_1 \cup P_2$$

$$T' = \{t_{i,j} : \quad t_i \in T_1, t_j \in T_2, \sigma_1(t_i) = \sigma_2(t_j)\}$$

$$I'(t_{i,j}) = I_1(t_i) + I_2(t_j)$$

$$O'(t_{i,j}) = O_1(t_i) + O_2(t_j)$$

$$\mu' = (\mu_1, \mu_2)$$

$$\sigma'(t_{i,j}) = \sigma_1(t_i) = \sigma_2(t_j)$$

$$F' = \left\{(\mu^i, \mu^j) : \mu^i \in F_1, \mu^j \in F_2\right\}$$

First, it is important to notice that even though the definition sets $P' = P_1 \cup P_2$, many of these places are spurious. A place is spurious if for all possible executions, no token will ever be assigned to the place. This means, for instance, that possibly many of the elements of $F'$ are unreachable. We will prove both inclusions.

The inclusion $L_G(PN') \subseteq L_1 \cap L_2$ is a bit troublesome to prove. If $\sigma \in L_G(PN')$, then there exists a sequence of transitions $\alpha \in T'^*$ such that $\exists_{\rho \in \uparrow F'} \quad (\mu_1, \mu_2) \left|\alpha\right\rangle_{PN'} = \rho = (\underbrace{\rho_1}_{\#P_1}, \underbrace{\rho_2}_{\#P_2})$. Let $l_1 : T' \mapsto T_1$ and $l_2 : T' \mapsto T_2$. We can then state that $\mu_1 \left|l_1(\alpha)\right\rangle_{PN_1} = \rho_1$ and $\mu_2 \left|l_2(\alpha)\right\rangle_{PN_2} = \rho_2$, where $\rho_1 \in \uparrow F_1$ and $\rho_2 \in \uparrow F_2$; this is enough to prove the inclusion, due to the way $\sigma'$ was defined. This last statement involves checking that all transitions of $l_j(\alpha)$ can fire. They can fire due to the following facts: $\mu_1^k = pr_{P_1}(\mu'^k)$ and $\mu_2^k = pr_{P_2}(\mu'^k)$ for all $k > 1$ and so the firing conditions are always satisfied.

The inclusion $L_1 \cap L_2 \subseteq L_G(PN')$ is easier. If there is a word $\sigma \in L_1 \cap L_2$ then there are $\alpha_1 \in T_1^*$, $\alpha_2 \in T_2^*$ such that $\exists_{\rho_1 \in \uparrow F_1} \quad \mu_1 \left|\alpha_1\right\rangle_{PN_1} = \rho_1$ and $\exists_{\rho_2 \in \uparrow F_2} \quad \mu_2 \left|\alpha_2\right\rangle_{PN_2} = \rho_2$. Let $l : T_1 \times T_2 \mapsto T'$ be such that $l(t_i, t_j) = t_{i,j}$ if $t_i \in T_1$ and $t_j \in T_2$.

Assuming $\mu_1^{k+1} = \mu_1^k - I_1(t_{i_k}) + O_1(t_{i_k})$ and $\mu_2^{k+1} = \mu_2^k - I_2(t_{j_k}) + O_1(t_{j_k})$ it is very easy to prove that $(\mu_1^{k+1}, \mu_2^{k+1}) = (\mu_1^k, \mu_2^k) - I(t_{i,j_k}) + O(t_{i,j_k})$ where $t_{i,j_k}$ is the $k^{th}$ transition to be fired. It is then possible to state that $(\mu_1, \mu_2) \left|l(\alpha_1, \alpha_2)\right\rangle_{N'}$ and it will reach a marking in $\uparrow F'$. □