Supervisory Control of Petri Nets using Linear Temporal Logic

Manuel Biscaia Martins

Extended Abstract

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
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
1 Introduction

We should recognize that most of our actions are extremely complicated, if we analyze each subroutine. 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 [16], 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 [24, 23]. 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

## Contents

1 Introduction .................................................. 1
2 Preliminaries .................................................. 2
   2.1 Language theoretic and bag theory notations ................. 2
   2.2 Finite state automata ......................................... 4
   2.3 Petri nets .................................................. 5
   2.4 Linear Temporal Logic and the QPLTL extension ............ 5
   2.5 Supervisors and controllability conditions .................. 7
3 Proposed methodology ......................................... 8
   3.1 Proposed methodology steps .................................. 8
   3.2 Examples and Modifications .................................. 9
4 Considerations on Petri net $\omega$-languages .................... 9
   4.1 Petri net $\omega$-languages definitions ...................... 9
   4.2 Generalization of the generalized Büchi automaton concept .. 10
5 Conclusion and future work ...................................... 11
as a Petri net, a formalization with a higher degree of expressibility [22], 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)[25, 26], 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's 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.

Furthermore, during our investigations, we had to approach the subject of the Petri net ω-languages. Our research was initially focused on [29] and then on earlier works [2, 18]. In the hope of using a Petri net characterization theorem in [29], we discovered a small mistake by Yamasaki, critical to the proof of the wanted theorem; in fact to correct the mistake, we had to rewrite two of Yamasaki’s ω-languages accepting conditions for Petri nets. With these rewritten definitions we are able to reprove Yamasaki’s theorem; we are also able to prove the equivalence between the Petri net analogous of generalized Büchi automata and Petri nets with a Büchi-like accepting condition. The Petri net analogous of generalized Büchi automata is, as far as we know, an unmentioned concept in the literature, and so its equivalence to regular Petri nets is also a new contribution.

2 Preliminaries

2.1 Language theoretic and bag theory notations

In this subsection 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 Σ be a set called the alphabet. Then, a non-empty finite sequence s over Σ is a function s : \{1, 2, \ldots, n\} → Σ for some n ≥ 1. The empty sequence is represented as λ. An infinite sequence s over Σ is a function s : N → Σ.

There is a basic operation over finite sequences called concatenation. The concatenation of two finite sequences s₁, s₂, represented by s₁.s₂ (sometimes the . is omitted), is a finite sequence s₁.s₂ : \{1, \ldots, n₁, n₁ + 1, \ldots, n₁ + n₂\} where s₁.s₂(n) = s₁(n), if 1 ≤ n ≤ n₁, and s₁.s₂(n) = s₂(n) if n₁ < n ≤ n₁ + n₂. Note that the concatenation operation is not commutative, but it is associative. The set Σ⁺ is the set of all finite sequences, empty or not. The set Σω is the set of all infinite sequences. A language over Σ is a subset of Σ⁺. A ω-language is a subset of Σω. The operation of concatenation can be extended to be executed between a finite sequence s₁ and an infinite sequence s₂. Note, however, that s₁.s₂ is a valid operation, while s₂.s₁ is not. The function inf : Σω → Σ is defined as inf(α) = \{a ∈ Σ : α(n) = a for infinitely many n\}. The function ran : Σω → Σ is defined as ran(α) = \{a ∈ Σ : α(n) = a for some n\}. The function ωQ : Σ⁺ ∪ Σω → \{True, False\} returns True if the argument is an infinite sequence and False otherwise. The function pref(α) : Σ⁺ ∪ Σω → 2Σ⁺ 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 \((\cdot)^c\), 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 \(\cdot\) be two languages \( L_1, L_2 \), 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 \( a.x_1 | | b.x_2 = a.(x_1 | | b.x_2) \cup b.(a.x_1 | | x_2) \) 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 \((\cdot)^*\) and \((\cdot)^\omega\), related with concatenation. The operator \((\cdot)^*\) is called Kleene star operator. The Kleene star operator applied to a word \( \alpha \in \Sigma^* \) is defined as follows: \( \alpha^* = \bigcup_{i=0}^{\infty} \alpha^i \). 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 \( L^\omega = \{ \alpha_1 \ldots \alpha_n : \alpha_1, \ldots, \alpha_n, \ldots \in L - \{ \lambda \} \} \). While not directly related to languages, the projection function \( pr_i : A_1 \times \ldots \times A_n \mapsto A_i \) is such that \( pr_i((a_1, \ldots, a_n)) = a_i \). 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.

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 \( x \in B \) 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:

\[
A \cup B = #(x, A \cup B) = \max(#(x, A), #(x, B)),
\]

\[
A \cap B = #(x, A \cap B) = \min(#(x, A), #(x, B)),
\]

\[
A + B = #(x, A + B) = #(x, A) + #(x, B),
\]

\[
A - B = #(x, A - B) = #(x, A) - #(x, A \cap B).
\]

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 all 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 \( D = \{ d_1, d_2, \ldots, d_n \} \); then, there is a natural correspondence between each bag \( B \) over \( D \) and the \( n \)-vector \( f = (f_1, f_2, \ldots, f_n), f_i = #(d_i, B) \). This correspondence is known as the **Parikh mapping**, introduced in [20]. When using Parikh mappings we tend to
use the index order of the domain.

2.2 Finite state automata

In this subsection will give some basic knowledge on finite state automata. The main sources used were [13, 19, 21, 3, 17]. A finite state automaton $N$ is a tuple $N = (S, \Sigma, \delta, s_0, F)$, with a finite set $S$ of states, the alphabet $\Sigma$, the transition relation $\delta \subseteq S \times \Sigma \times S$, the initial state $s_0 \in S$, and the set $F \subseteq S$ of accepting states. Any $\alpha = (q_0, e_0, p_0)(q_1, e_1, p_1) \ldots (q_n, e_n, p_n) \in \delta^*$ is called a run of $N$, if $q_0 = s_0$ and $p_i = q_{i+1}$ for any $i < n$. This definition is easily generalized to infinite sequences. We write $s_0 |\alpha\rangle_N$ in both the finite and the infinite case.

We associate two types of languages over finite sequences:

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

$$L_m(N) = \{pr_2(\alpha) : \alpha \in \delta^*, \exists s \in F \ s_0 |\alpha\rangle_N = s\}.$$

We can associate five types of languages over infinite sequences; although many other interesting definitions exist, these are in some way primitive [17]:

$$L_{\text{True}}^\omega(N) = \{pr_2(\alpha) : \alpha \in \delta^*, s_0 |\alpha\rangle_N\},$$

$$L_{\text{ran}}^\omega(N) = \{pr_2(\alpha) : \alpha \in \delta^*, s_0 |\alpha\rangle_N, \text{ran}(pr_1(\alpha)) \cap F \neq \emptyset\},$$

$$L_{\text{ran} \subseteq}^\omega(N) = \{pr_2(\alpha) : \alpha \in \delta^*, s_0 |\alpha\rangle_N, \text{ran}(pr_1(\alpha)) \subseteq F\},$$

$$L_{\text{inf} \subseteq}^\omega(N) = \{pr_2(\alpha) : \alpha \in \delta^*, s_0 |\alpha\rangle_N, \text{inf}(pr_1(\alpha)) \cap F \neq \emptyset\},$$

$$L_{\text{inf} \subseteq}^\omega(N) = \{pr_2(\alpha) : \alpha \in \delta^*, s_0 |\alpha\rangle_N, \text{inf}(pr_1(\alpha)) \subseteq F\}.$$

If $L$ is an $\omega$-language and $L = L_{\text{inf} \subseteq}^\omega(N)$, then $L$ is said to be recognized by a finite state automaton with a B"uchi acceptance condition. An interesting generalization of this concept is the generalized B"uchi acceptance criterion. A generalized B"uchi automaton $\mathcal{G}N$ is a tuple similar to a FSA $\mathcal{G}N = (S, \Sigma, \delta, s_0, \mathcal{F})$, where $\mathcal{F} = \{F_1, \ldots, F_n\}$, with $F_i \subseteq S$. The generalized B"uchi acceptance criterion is:

$$L_{\text{inf} \subseteq}^\omega(\mathcal{G}N) = \{pr_2(\alpha) : \alpha \in \delta^*, s_0 |\alpha\rangle_{\mathcal{G}N}, \forall F_i \in \mathcal{F} \ \text{inf}(pr_1(\alpha)) \cap F_i \neq \emptyset\}.$$

It was proven that this extension does not add any expressiveness.

Furthermore, we need to discuss trimming operations. A state $s \in S$ is said to be reachable in $N = (S, \Sigma, \delta, s_0, F)$ if there exists $\alpha \in \delta^*$ such that $s_0 |\alpha\rangle_N = s$. A state $s$ is said to be co-reachable if there exists $\alpha$ such that $s |\alpha\rangle_N \in F$. A state $s$ is said to be co-accessible if there exists $\alpha \in \delta^*$, such that $s |\alpha\rangle_N$ and $\text{inf}(N(\text{alpha})) \cap F \neq \emptyset$. The trimming operation $(\text{trim}(\cdot))$ over a finite state automaton intended to accept infinite sequences simply restricts the set $S$ to all reachable and co-reachable states. The trimming operation $(\text{trim}^\omega(\cdot))$ over a finite state automaton intended to accept infinite sequences restricts the set $S$ to all reachable and co-accessible states. The trimming operation $\text{trim}^\omega$ can easily be extended to be applied to a generalized B"uchi automaton.
2.3 Petri nets

Petri nets are a formalism with more expressive power than FSA. We will give some basic definitions on Petri nets, Petri languages and ω-languages. For more information, consult [22], or other works referenced by it [12, 11]; for more information regarding decidability of problems in Petri nets consult [6].

A Petri net $PN$ is a tuple $PN = (P, T, I, O, \mu_0, \sigma, F)$, where $P$ is a finite set of places, $T$ is a finite set of transitions, $I : T \mapsto P^\infty$ is the input function, $O : T \mapsto P^\infty$ is the output function, $\mu_0$ is an element of $P^\infty$, called the initial marking; the function $\sigma : T \mapsto \Sigma$ is a λ-free labeling function, and $F$ is a set of elements of $P^\infty$.

A marking is an element of $P^\infty$. A marking is the analogous of a state in FSA; unlike FSA however, the state space can be infinite. The filter set of $F$, where $F$ is a set of markings is defined as follows: $\uparrow F = \{ \mu : \mu \in P^\infty, \exists \mu' \in F \ \mu' \subseteq \mu \}$. Now we have to define how the markings evolve.

If $t$ is a transition, $t$ is said to be fireable in $PN$ at marking $\mu$ if $I(t) \subseteq \mu$. Assuming $t$ is fireable, if $t$ fires at marking $\mu$, the resulting marking will be given by the firing rule $\mu' = \mu - I(t) + O(t)$, and it is written $\mu \upharpoonright _t PN = \mu'$. It is easy to generalize the firing rule for finite or infinite sequences of transitions. We now are able to associate languages to a Petri net. There are four main types of accepting conditions in Petri nets:

- $L_L(PN) = \{ \sigma(\alpha) : \alpha \in T^* \ \text{and} \ \exists \mu_f \in F \ \mu_0 | \alpha \}_{PN} = \mu_f \}$,
- $L_G(PN) = \{ \sigma(\alpha) : \alpha \in T^* \ \text{and} \ \exists \mu_f \in F \ \mu_0 | \alpha \}_{PN} = \mu_f \}$,
- $L_T(PN) = \{ \sigma(\alpha) : \alpha \in T^* \ \text{and} \ \mu_0 | \alpha \}_{PN} = \mu_t \ \text{with} \ \mu_t \ \text{as terminating state} \}$,
- $L_P(PN) = \{ \sigma(\alpha) : \alpha \in T^* \ \text{and} \ \mu_0 | \alpha \}_{PN} \}$.

A terminating state is a marking such that all transitions are not fireable. An important fact about Petri net languages is that they are closed for union and intersection. They are not closed for complement, though.

When considering Petri net ω-languages, the accepting conditions we will present are based on [29], although as discussed in the introduction they had to be adapted. Consult Section 4 for more information. The definitions presented by us are as follows, where $C$ is the set of all markings:

- $L^\omega_{\text{True}}(PN) = \{ \sigma(\alpha) : \alpha \in T^\omega, \mu_0 | \alpha \}_{PN} \}$,
- $L^\omega_{\text{ran} \cap}(PN) = \{ \sigma(\alpha) : \alpha \in T^\omega, \mu_0 | \alpha \}_{PN} \ \text{ran}(PN(\alpha)) \cap \uparrow F \neq \emptyset \}$,
- $L^\omega_{\text{ran} \subseteq}(PN) = \{ \sigma(\alpha) : \alpha \in T^\omega, \mu_0 | \alpha \}_{PN} \ \text{ran}(PN(\alpha)) \subseteq \uparrow F \}$,
- $L^\omega_{\text{inf} \cap}(PN) = \{ \sigma(\alpha) : \alpha \in T^\omega, \mu_0 | \alpha \}_{PN} \ \omega Q(PN(\alpha) | \cap F) = \text{True} \}$,
- $L^\omega_{\text{inf} \subseteq}(PN) = \{ \sigma(\alpha) : \alpha \in T^\omega, \mu_0 | \alpha \}_{PN} \ \omega Q(PN(\alpha) | \cap F) = \text{False} \}$.

It is quite easy to see that for all FSA $N$ there exist Petri nets $PN'$ such that $L^\omega_\gamma(PN') = L^\omega_\gamma(N)$, for all accepting conditions $\gamma = \{ \text{True, ran} \cap, \text{ran} \subseteq, \text{inf} \cap, \text{inf} \subseteq \}$.

2.4 Linear Temporal Logic and the QPLTL extension

Linear Temporal Logic syntax and semantics were taken from [5, 4]. Earlier sources exists but these two works present a good introduction to the more general area of Temporal Logic. After choosing a set $\Pi$ of propositional
symbols, in our case representing the firing of certain events in the original Petri net, the set $LTL(II)$ is the set of all well formed formulas created by the following rule:

$$\phi = True | p \in II | (\neg \phi) | (\phi \rightarrow \psi) | (X \phi) | (\phi U \psi)$$

This syntax is sufficient to generate all commonly agreed $LTL(II)$ formulas. It is common to introduce the other boolean connectives by abbreviation, and also the following temporal modalities:

- $(F \phi) \equiv_{abv} (True U \phi)$,
- $(\phi R \psi) \equiv_{abv} (\neg((\neg \phi) U (\neg \psi)))$,
- $(G \phi) \equiv_{abv} (False R \phi)$.

The $X$ modality should be read as *next time*, the $U$ modality should be read as *until*, the $F$ should be read as *sometime in future*, and finally $G$ should be read as *always*.

The semantics can be defined without using directly Kripke structures, using instead infinite sequences of valuations $\sigma: \mathbb{N} \mapsto 2^\Pi$. Let then $\sigma: \mathbb{N} \mapsto 2^\Pi$. Let $\phi \in LTL(II)$. 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 $\models$ inductively on the structure of $\phi$:

- $\sigma \models True$,
- $\sigma \models p$ if $p \in \sigma(0)$,
- $\sigma \models (\phi \rightarrow \psi)$ if $\sigma \not\models \phi$ or $\sigma \models \psi$,
- $\sigma \models (\neg \phi)$ if $\sigma \not\models \phi$,
- $\sigma \models (X \phi)$ if $\sigma^{1} \models \phi$,
- $\sigma \models (\phi U \psi)$ if $\exists j (\sigma^{j} \models \psi$ and $\forall k<j (\sigma^{k} \models \phi))$.

A formula $\phi \in LTL(II)$ 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 $\models \phi$.

It is possible to create a FSA such that the language recognized by it with Büchi’s accepting condition is precisely the set of all infinite sequences of valuations that can satisfy a certain $LTL$ formula. See [28, 9] for more information.

Unfortunately, there are $\omega$-languages accepted by finite state automata that are not expressible by a $LTL$ formula. This statement is obtained easily from Wolper’s work [27]. However, adding existential quantification to $LTL$ we are able to obtain a logic, called $EQLTL$, that is as powerful as FSA equipped with Büchi accepting condition. For more information see [26, 14, 25, 8, 5, 7].

We will not go into details, although we will show the syntax of $EQLTL$ and its semantics. A formula $\phi$ belongs in $EQLTL(II)$ if and only if $\phi \in LTL(II)$ or $\phi = \exists \pi_{1} \ldots \exists \pi_{k} \psi$, where $\psi \in LTL(II)$ and $\pi_{1}, \ldots, \pi_{k}$ occur in $\psi$. The semantics are also quite simple: $\sigma \models_{EQLTL} \psi, \psi \in LTL(II)$ iff $\sigma \models_{LTL} \psi$. Otherwise, $\sigma \models_{EQLTL} \exists \pi_{1} \psi$, iff there exists $\sigma': \mathbb{N} \mapsto 2^\Pi$, $\pi_{1}$-equivalent to $\sigma$. 6
2.5 Supervisors and controllability conditions

We are interested in, given an initial task plan represented in Petri net form, modifying and certifying the modification automatically. In order to do so, we will intersect the task plan Petri net with the Petri net representing our desired specifications. Sometimes, however, certain events cannot be restricted. So we will use the concepts initially developed by Ramadge and Wonham [24, 23] to guarantee that the intersections do not restrict uncontrollable events. Although supervisory control of Petri nets is much more difficult than supervisory control of FSA, we are not interested in obtaining classical results of supervisory control. We just want to check whether a condition called the controllability condition is verified in the intersection of the Petri net representations, since if the controllability condition is verified, we are assured that we are not restricting anything impossible to control. So we will first present the definition of a Petri net supervisor, according to [15, 10], making some adaptations for our specific problem.

Let $PN = (P,T,I,O,\mu_0,\sigma,F)$ be a Petri net. 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 $Sp : L_P(PN) \rightarrow 2^\Sigma$. 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 $e \in UAE_\tau$ if there exists $\alpha \in T^*$ and $t \in T$ such that $\sigma(\alpha) = \tau$, $\mu_0 |(\alpha,t)|_{PN}$, and $e = \sigma(t) \in \Sigma_uc$.

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 $\mu_0 |t|_{PN}$ and $\sigma(t) \in Sp(\lambda)$,
- if $\alpha = t_1t_2t_3 \ldots t_n$ and if $\alpha_1 = t_1t_2t_3 \ldots t_{n-1}$ is a run in $Sp/PN$ then $\alpha$ is a run in $Sp/PN$ if $\mu_0 |\alpha|_{PN}$ and if $\sigma(t_n) \in Sp(\sigma(\alpha_1))$.

The language accepted by this object, represented as $L(Sp/PN)$ is defined as follows:

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

Finally, a supervisor can be given by a Petri net representation, and so we obtain that $L(Sp/PN) = L_P(Sp) \cap L_P(PN)$. We can also be interested in the objectives set by the initial system, or even the objectives set by the specification in Petri net representation. This is not classical supervisory control, as normally the supervisor cannot set any objectives; on our case, however, we only wish to ensure that the intersection does not restrict uncontrollable events. We then can prove the following theorem for the case when both the system and the specifications set objectives:

**Proposition 2.1.** Let $PN = (P,T,I,O,\mu_0,\sigma,F)$ be a Petri net, the system, and let $Sp = (P_{Sp},I_{Sp},O_{Sp},\mu_0,\sigma_{Sp},F_{Sp})$ 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, $\forall_{\sigma \in L_P(Sp)} \exists_{\sigma'} \sigma' \in L_G(Sp)$,
• \( PN \) is non-blocking, \( \forall \sigma \in L_P(PN) \exists \sigma' \in \Sigma^* \quad \sigma' \in L_G(PN) \),

• \( L_G(Sp) \) and \( L_G(PN) \) are non-conflicting, \( L_G(PN) \cap L_G(Sp) = L_G(PN) \cap L_G(Sp) \).

If we are only interested in the P-Type language of the system/specification, we can eliminate the non-blocking requirement. Furthermore, we also provide an algorithm to decide whether \( L_P(Sp) \cap L_P(PN) \) is controllable when \( L_P(Sp) \) is a regular language (which is our case). The work [15] already suggests this algorithm, although we generalize it for the more general case of non-deterministic Petri nets systems and specifications.

3 Proposed methodology

3.1 Proposed methodology steps

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

Step 3.1 (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 3.2 (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 3.3 (Translation from EQLTL to generalized B"uchi automaton). Let \( \phi \in EQLTL(\Sigma \cup Q) \) be our specification. We can obtain a generalized B"uchi automaton \( GBA_\phi \) such that \( L_{EQLTL}(\phi) = L_{inF}(GBA_\phi) \).

Step 3.4 (Transforming \( GBA_\phi \)). The first transformation is only performed if \( \phi \in EQLTL(\Sigma \cup Q) \) \(-\) \( LTL(\Sigma) \). If so, consider the generalized B"uchi automaton \( GBA'_\phi = (S, \Sigma, \delta', s_0, F) \) obtained from \( GBA_\phi = (S, 2^{\Sigma \cup P}, \delta, s_0, F) \) 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 3.5 (Transforming the generalized B"uchi automaton into a trimmed B"uchi automaton). The first of two transformations is simply the application of the algorithm that shows the equivalence between the expressibility of generalized B"uchi automata and B"uchi automata [28]. The second transformation is the trimming operation of the B"uchi automaton, using \textit{trim}().

Step 3.6 (Transforming the B"uchi automaton into a Petri net). In order to transform \( BA'_\phi \) into a Petri net, the designer should use the transformation that shows that all \( (\omega) \)-regular languages are in fact Petri net languages.

Step 3.7 (Intersecting \( PN_\phi/\mathcal{PN}_\phi \) 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 is EXPSPACE-hard.
Step 3.8 (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 $L_G(PN_{Goal}) = L_P(PN_{system}) \cap
L_G(PN'_\phi)$, depending on what was the supervision problem he approached. 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.

3.2 Examples and Modifications

We successfully used all types of supervisory control proposed by us [1], in three initial systems. The well
known dining philosophers, a Petri net system representing a triangulation pass between two robots, whose
representation was strictly more powerful than what would be possible using FSA, and also a system of four
robots passing the ball between each other, where we enforced some complex specifications.

As discussed there, there are two main flaws in the method; while the nondeterminism factor forces us to
only use the obtained Petri net system as an oracle of whether a certain behaviour is acceptable or not, the
excess of places and transitions effectively reduce the complexity of specifications and systems we can use. In
order to not obtain non-deterministic task plans, we can assume that the initial system given is deterministic,
and then “determinize” the FSA representing the specification. The excess of places and transitions cannot be
eradicated completely, to our knowledge, since for instance it is impossible to trim a Petri net; it is one of the
future work ideas: find interesting algorithms to avoid producing so many unnecessary Petri net components.

4 Considerations on Petri net $\omega$-languages

This section is intended as an extension of Subsection 2.3. We will discuss various topics: in the first subsection
we will present Yamasaki’s definitions for infinite sequences accepting conditions of Petri nets and we will discuss
our claim that Petri net definitions given in 2.3 were as far as we know more useful than those in [29], paving
the road for a powerful characterization theorem. In the second subsection we will present our definition of
the analogous concept of generalized Büchi automaton and its equivalence to a petri net with Büchi accepting
criterion.

4.1 Petri net $\omega$-languages definitions

The definitions that inspired the Petri net accepting conditions of $\omega$-words in Subsection 2.3 will be given:

Let $PN = (P, T, I, O, \mu_0, \sigma, F)$ be a Petri net. The $\omega$-languages accepted by $PN$, according to Yamasaki’s
definitions, will be defined:

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

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

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

$L_{Y, inf \cap}(PN) = \{ \sigma(\alpha) : \alpha \in T^\omega, \mu_0 | \alpha \} _{PN}, inf(PN(\alpha)) \cap \uparrow F \neq \emptyset \}$,
\[ L_{Y,\infty}^\omega(PN) = \{ \sigma(\alpha) : \alpha \in T^\omega, \mu_0 | \alpha \rangle_{PN}, \text{inf}(PN(\alpha)) \subseteq \uparrow F \} . \]

Consider also \( P_{\beta, \gamma} = \{ L \subseteq \Sigma^\omega : \text{there exists a Petri net } PN \text{ such that } L = L_{\beta, \gamma}^\omega(PN) \} \), where \( \beta \) is left blank if we are considering our definitions.

We tagged the definitions of the accepting conditions with a \( Y \) in order to remind who the author was. In [29], the author proposes the following definition and theorem:

\[ L_{\text{True}}^\omega(PN, R) = \{ \sigma(\alpha) : R \subseteq T^\omega, \alpha \in R, \mu_0 | \alpha \rangle_{PN}, \} . \]

**Theorem 4.1.** For any accepting condition \( \gamma \in \{ \text{True, ran} \cap, \text{ran} \subseteq, \text{inf} \cap, \text{inf} \subseteq \} \), \( P_{Y, \gamma} = \{ L_{\text{True}}^\omega(PN, R) : PN \text{ is a Petri net, } R \in E_{\gamma} \} \).

We prove in [1] that the proof of the theorem is only valid if we consider \( P_{\gamma} \) instead of \( P_{Y, \gamma} \). It does not mean, however, that the theorem is false as stated by Yamasaki; our claim is simply that his proof is flawed. We will also present two examples, Figure 1 and Figure 2, showing that the accepting conditions proposed by us differ at first sight from the ones proposed by Yamasaki.

**Figure 1:** The Petri net \( PN \) represented above with \( F = \{(1)\} \) is such that \( L_{Y,\infty}^\omega(PN) \neq L_{\infty}^\omega(PN) \), if \( F = \{(1)\} \).

**Figure 2:** The Petri net \( PN' \) represented above with \( F = \{(0, 1)\} \) is such that \( L_{Y,\infty}^\omega(PN) \neq L_{\infty}^\omega(PN) \), as \((ab)^\omega \in L_{Y,\infty}^\omega(PN) \) and \((ab)^\omega \notin L_{\infty}^\omega(PN) \).

**4.2 Generalization of the generalized Büchi automaton concept**

The generalized Büchi automaton concept is extremely useful, as it allows us to request infinite passings by multiple states, unlike the regular Büchi accepting condition. Although, from the point of view of expressibility the generalized Büchi automaton concept is useless, it provides an easy algorithm to automatically convert conjunctive specifications in a Büchi automata. Its usefulness can in fact be seen during our work, as the
algorithm that transforms an EQLTL formula into a Büchi automaton, in fact transforms the formula into a
generalized Büchi automaton.

During our investigation, we considered the possibility of the existence of a Petri net analogous definition.
If the equivalence between generalized Büchi automata and regular Büchi automata was carried to Petri nets,
we could use the extra expressibility brought by Petri nets to accelerate the proposed methodology, avoiding
Step 3.5. And so, the multiple accepting conditions Petri net was born. We were also capable of proving its
equivalence to Petri nets equipped with the accepting condition \( \inf \cap \), or Büchi’s accepting condition.

**Definition 4.1.** Let \( PN = (P, T, I, O, \mu_0, \sigma) \) be a \( \lambda \)-free marked Petri net structure and \( F = \{ F_1, \ldots, F_n \} \) be a
finite set of sets of accepting markings. \( MACPN = (P, T, I, O, \mu_0, \sigma, F) \) is called a multiple accepting conditions
Petri net (MACPN). The language accepted by a multiple accepting conditions Petri net \( MACPN \), represented
by \( L_{\omega}^{\inf \cap} (MACPN) \), is defined as follows:

\[
L_{\omega}^{\inf \cap} (MACPN) = \{ \sigma(\alpha) : \alpha \in T_{\omega}, \mu_0 |_{MACPN}, \forall F_i \in F \quad \omega Q(MACPN(\alpha) | F_i) = True \}.
\]

**Theorem 4.2.** Let \( MACPN = (P, T, I, O, \mu_0, \sigma, F) \) be a multiple accepting conditions Petri net with \( \sigma : T \rightarrow \Sigma \)
the \( \lambda \)-free labeling function and \( F = \{ F_1, \ldots, F_n \} \) with \( n > 1 \) and \( PN_{MACPN} = (P', T', I', O', \mu'_0, \sigma', F') \) the
witness of \( MACPN \). Then \( L_{\omega}^{\inf \cap} (MACPN) = L_{\omega}^{\inf \cap} (PN_{MACPN}) \) and so \( P_{\inf \cap} = P_{\omega} \).

Unfortunately, given that we have proven the general case, the witness (see [1]) is not optimized for the case
when \( MACPN \) is in fact a generalized Büchi automaton in disguise. However, we have shown that in some very
specific conditions it is better to use this translation algorithm

### 5 Conclusion and future work

The dissertation main objective was the extension of the work by Lacerda and Lima [16] 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 determinstic 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 [29], 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.

Regarding future work, many interesting theoretic questions remain, specially in the domain of Petri net $\omega$-languages, and the hierarchy of Petri net $\omega$-languages, relating our newly introduced definitions with Valk’s [2], or even the (possibly) flawed ones from Yamasaki [29]. Moreover, concerning practical issues, the proposed controllability conditions are hard to check, and so investigation to weaken them would be useful. How does this method of modification and certification of a task plan interact with other methods, specially other from supervisory control? And even more importantly, can this methodology be applied to real world examples?
Bibliography


