UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Supervision of Discrete Event Systems Based on Temporal Logic Specifications

by

Bruno Filipe Araújo Lacerda

Supervisor: Prof. Dr. Pedro Manuel Urbano de Almeida Lima

Thesis specifically prepared to obtain the PhD Degree in Electrical and Computer Engineering

Draft

May 2013

Abstract

In this work, a methodology for performing supervisory control of discrete event systems based on linear temporal logic is presented. The discrete event system is modelled as a finite state automaton or a Petri net and, given a linear temporal logic safety specification written over the events and the states of the system, either a finite state automaton or a Petri net that realizes a supervisor that restricts the behaviour of the system so that it satisfies the specification is constructed. The first step of the methodology is based on defining semantics for evaluating the linear temporal logic formulas in both the finite state automaton and the Petri net models. These semantics are then used to compose the corresponding model with the Büchi automaton representing the linear temporal logic formula. The result of each composition is the minimal restriction of the uncontrolled behaviour that satisfies the formula. A strictly symbolic semantics is defined for both finite state automata and Petri nets, and a more general semantics for Petri nets that includes and takes advantage of the fact that in Petri nets the state can be described algebraically is also introduced. This generalized semantics also has the benefit of reducing the size of the supervisor realizations, when compared to the strictly symbolic approach. The structure obtained from the presented compositions represents a coarse realization for the supervisor, for which one can apply existing approaches in the literature to verify and enforce relevant properties in supervisory control theory, such as admissibility and deadlock-freeness. A discussion on how to apply these approaches for the results of the compositions presented in this work is also provided. For the case of Petri nets, the inherent limitations of the model with regards to supervisory control theory are also addressed. To illustrate and compare the models and semantics used in the methodology, three examples of application are described: a social robot, a team of simulated soccer robots and a real multi-robot surveillance scenario mock-up.

Keywords: Discrete Event Systems, Supervisory Control, Finite State Automata, Petri Nets, Linear Temporal Logic, Formal Methods, Task Specification, Robotics, Multi-Robot Coordination, Multiagent Systems

Resumo

Neste trabalho apresenta-se uma metodologia para realizar controlo por supervisão de sistemas de eventos discretos, baseada em lógica temporal linear. O sistema de eventos discretos é modelado como um autómato de estados finito ou uma rede de Petri e, dada uma especificação de segurança em lógica temporal linear escrita sobre os eventos e estados do sistema, é construído respectivamente um autómato de estados finito ou uma rede de Petri. Esta nova estrutura realiza um supervisor que restringe o comportamento do sistema, tal que este satisfaça a especificação. O primeiro passo da metodologia baseia-se em definir semânticas para avaliar as fórmulas de lógica temporal linear nos modelos autómato de estados finito e rede de Petri. Estas semânticas são então utilizada para compor o modelo correspondente com o autómato de Büchi que representa a fórmula. O resultado de cada composição representa a menor restrição do comportamento não-controlado que satisfaz a fórmula. Uma semântica estritamente simbólica é definida tanto para autómatos de estados finitos como para redes de Petri, e é também introduzida uma semântica mais geral para redes de Petri, que inclui e tira proveito do facto do estado de uma rede de Petri poder ser descrito algebricamente. Esta semântica generalizada apresenta também o benefício de reduzir o tamanho das realizações de supervisores, quando comparada com a abordagem estritamente simbólica. A estrutura obtida pelas composições apresentadas representa uma realização grosseira para o supervisor, para a qual se podem aplicar abordagens existentes na literatura para verificar e fazer cumprir propriedades relevantes para a teoria de controlo por supervisão, tais como admissibilidade e liberdade de *deadlocks*. Uma discussão focada na aplicação destas abordagens para os resultados das composições apresentadas neste trabalho é também fornecida. Para o caso das redes de Petri, as limitações inerentes do modelo em relação à teoria de controlo por supervisão são também abordadas. Com o objectivo de ilustar e comparar os modelos e semânticas utilizados na metodologia, três exemplos de aplicação são descritos: um robot social, uma equipa de robots futebolistas simulados e um cenário real de vigilância multi-robot.

Palavras-Chave: Sistemas de Eventos Discretos, Controlo por Supervisão, Autómatos de Estados Finito, Redes de Petri, Lógica Temporal Linear, Métodos Formais, Especificação de Tarefas, Robótica, Coordenação Multi-Robot, Sistemas Multi-Agente

Acknowledgements

The sun is setting in Lisbon – as usual, I'm finishing writing later than expected – and I'm in 6.15 writing the last few words of my thesis. Finally, I've got this done and, though the feelings are mixed, one of them is of fulfilment: After more than 4 years I finally have my work ready to be handed in for evaluation! This could not have been done without the help of many people and I'd like to thank some of them.

First and foremost, I'd like to thank my advisor, Prof. Pedro Lima. His impact on my academic life has been immeasurable, starting in 2006 when he accepted to supervise my Master's thesis (during which I realized that research life might not be that bad), being available to taking me back for a short-time grant when, after 6 months of consulting work, I decided that I'd like to return to academia and see if continuing for a PhD was something that I wanted and, finally, for his guidance and patience throughout my PhD studentship. I would also like to thank Profs. Alessandro Saffiotti, Miguel Salichs and Paulo Tabuada for receiving me, for 3 months each, in their respective research groups. All of these stays were very learning and enjoyable experiences.

I also want to thank my colleagues, mainly all the guys in 6.15 for putting up with me all this time, and the guys in 6.12 for putting up with my e-puck scenario in the middle of their room (and also me, once in a while). I'm leaving ISR for now, but you will be missed and I hope we can see each other soon.

To my friends, both the Lisbon people and the Faial people (some of them waiting for me for dinner right now), your constant presence and support was fundamental for me. It would have been impossible to finish this work without having people that I like in the "outside world", always available to hang out and do stuff. I will not name you all, but if you're taking your time to read this words you're very probably one of them, so thank you!

Finally, I'd like to thank my family for all their support throughout all these years, and all the sacrifices that I know were made so I could come to Lisbon to study. Unfortunately, one of the results of those sacrifices was me following a path where I cannot be near you, but you are missed everyday.

And with this, the sun is down. Time to print this thing and get out of here.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Overview

The emergence of man-made technological systems, which start to appear ubiquitously in our everyday life, has originated the need to develop formal approaches to systems modelling, where the state evolution is event-driven, instead of the classical time-driven evolution. This type of systems, coined discrete event systems (DES) [Cassandras and Lafortune, 2006], exhibits a discrete state space which evolves according to the occurrence of a discrete set of asynchronous and instantaneous events. As these systems become more complex, there is a need to establish approaches that provide the designer with tools to easily compose complex models from simpler modular ones. By applying formal methods, one is presented with a systematic approach to modelling, analysis and design, scaling up to realistic applications, and enabling analysis of formal properties, as well as design from specifications. Furthermore, one can provide guarantees about certain system properties, either by automatic analysis or by design.

The acronym DES is an umbrella term that covers a large bulk of systems that satisfy the aforementioned properties. Therefore, one needs to decide a priori what point of view to take of the system and which modelling formalism to use. Regarding the former, there are two points of view usually taken:

- A *qualtivative* point of view, where properties such as safety, liveness or invariance can be analysed. Two classes of models used when taking this point of view are finite state automata (FSA) [Hopcroft et al., 2006] and Petri nets (PN) [Murata, 1989].

- A *quantitative* point of view, where, for example, properties concerning stochastic or deterministic performance, robustness or reliability of the system can be studied. Models like stochastic

timed automata [Glynn, 1989] or generalized stochastic Petri nets [Viswanadham and Narahari, 1992] are common when taking this point of view.

A DES model can be slightly modified either to take one view or the other, by adequate labelling of states and transitions of the DES representation. Also, one can build models for complex systems by composing smaller and simpler models of subsystems. All these operations of composition and analysis can be implemented by fast and sound algorithms.

In this thesis, we will concentrate on the qualitative point of view. In particular, we will focus on the supervisory control (SC) theory for DES, introduced in [Ramadge and Wonham, 1987]. The purpose of SC is, given a logical DES model – in our case a FSA or PN – of the open-loop uncontrolled behaviour of a system, to restrict its behaviour to a given specification, by dynamically disabling a subset of the events available for the system to execute at each state. We will be interested in safety specifications given in linear temporal logic (LTL) [Emerson, 1990]. Intuitively, a safety specification can be described as specifying that something "bad" will never happen, i.e., the system will be always kept inside a set of states that are considered "good". The use of LTL formulas provides a close-to-natural-language formalism which can be used to specify the intended behaviour, thus giving the designer the ability to tackle intricate goal behaviours more naturally.

Compared to traditional planning methods [Russell et al., 1995], SC theory assumes that the connections among primitive actions that eventually lead the system to display some given behaviour are pre-wired. However, this pre-design includes a set of alternative points (or decision points in the DES modelling the uncontrolled behaviour), over which a supervisor can act, enabling or disabling controllable events. The supervised system can be seen as a reactive plan which appropriately deals with different sensor readings, according to the specifications provided by the designer. Also, SC theory does not provide quantitatively optimal solutions – it is up to the designer to specify a restriction to all behaviours that can be carried out by the system, such that the goal is intuitively achieved in the best possible way.

In Figure 1.1, we present a diagram illustrating the methodology for the construction of a PN supervisor, given a PN model and an LTL specification. The dashed arrows represent the steps of the method, and we will be explaining all the elements of the diagram throughout this thesis. To summarize, we will start with a DES model of the system and a set of safety LTL formulas written over the set of events of the system and a set of symbols describing the system's states. For each of these formulas, its translation to a Büchi automaton is built, following the method presented in [Gastin and Oddoux, 2001]. The Büchi automaton is then appropriately composed with the model of the system, according to the semantics previously defined. This composition provides a structure that represents the smallest restriction – in the sense that any restriction which is more permissive in term of events that can be fired by the system will not comply with the specification – of the behaviour of the system that satisfies the LTL formula. Then, it is necessary to guarantee that this structure is admissible and deadlock-free, so that it can be used as a realization of the supervisor that, when run in closed-loop with the uncontrolled system, restricts its behaviour so that it satisfies the LTL

formula. We will describe the application of different known approaches to deal with this problem to the structures resulting from our compositions, and also present an a priori restriction of the LTL formulas, which can be used when the designer has a total knowledge of the system model, that guarantees admissibility by construction.

While the procedure we present can be computationally expensive, all computations are performed off-line.Thus, the execution of the supervisor resulting from our methodology requires very little in terms of computational resources at run time, given that all the calculations, as complex as they might be, are done before deploying the system. This is an important point because the supervisor will used in a feedback-loop with the system, hence it is important that it is able to react to changes in the system in a timely manner.

Although the work presented in this thesis is applicable to a wide array of systems – e.g., software systems, manufacturing systems, air traffic control systems – we will focus on its application to robot systems, mainly the coordination of multi-robot teams. Using LTL to specify and enforce coordination rules for teams of robots is quite natural. Furthermore, robotics is a field where innovation is often associated with non-formal approaches. Novel ideas and concepts are typically introduced for particular applications, through well-engineered systems, but lack guarantees concerning aspects such as safety, predictability, performance, robustness and reliability, essential to support the acceptability of robotic devices in factories, offices, hospitals or field (e.g., surveillance, medical care, planetary exploration) scenarios. As previously discussed for DES in general, with the fast dissemination of robot systems in the household and daily scenarios, where they must interact more frequently and in a natural way with humans, these guarantees are essential.

We will focus most of our attention on the part of the methodology related with PNs. This is justified by the fact that PNs are a more interesting and well-suited model for distributed systems, allowing one to more easily scale to larger systems. Furthermore, the study of SC is much more developed for FSA, so our contribution for SC of PNs is more substantial. In spite of that, we decided also to present the methodology for FSA. The reason for this option is two-fold. Firstly, being a more studied and simpler model, presenting the ideas for FSA first can help the reader to better grasp the notions, thus facilitating the understanding of the PN version. Secondly, to have a fair way to compare the FSA and PN approaches: we will be comparing the two approaches for one example, giving evidence that for distributed systems PNs provide a better option for modelling, due to their distributed state representation, instead of the exhaustive enumeration of the states required by FSA models.

1.2 Related Work

Petri Net Modelling and Analysis

The PN formalism and its use for the modelling of DE) has been widely studied in the literature. A good and complete survey – however somewhat dated – can be found in [Murata, 1989]. There is also

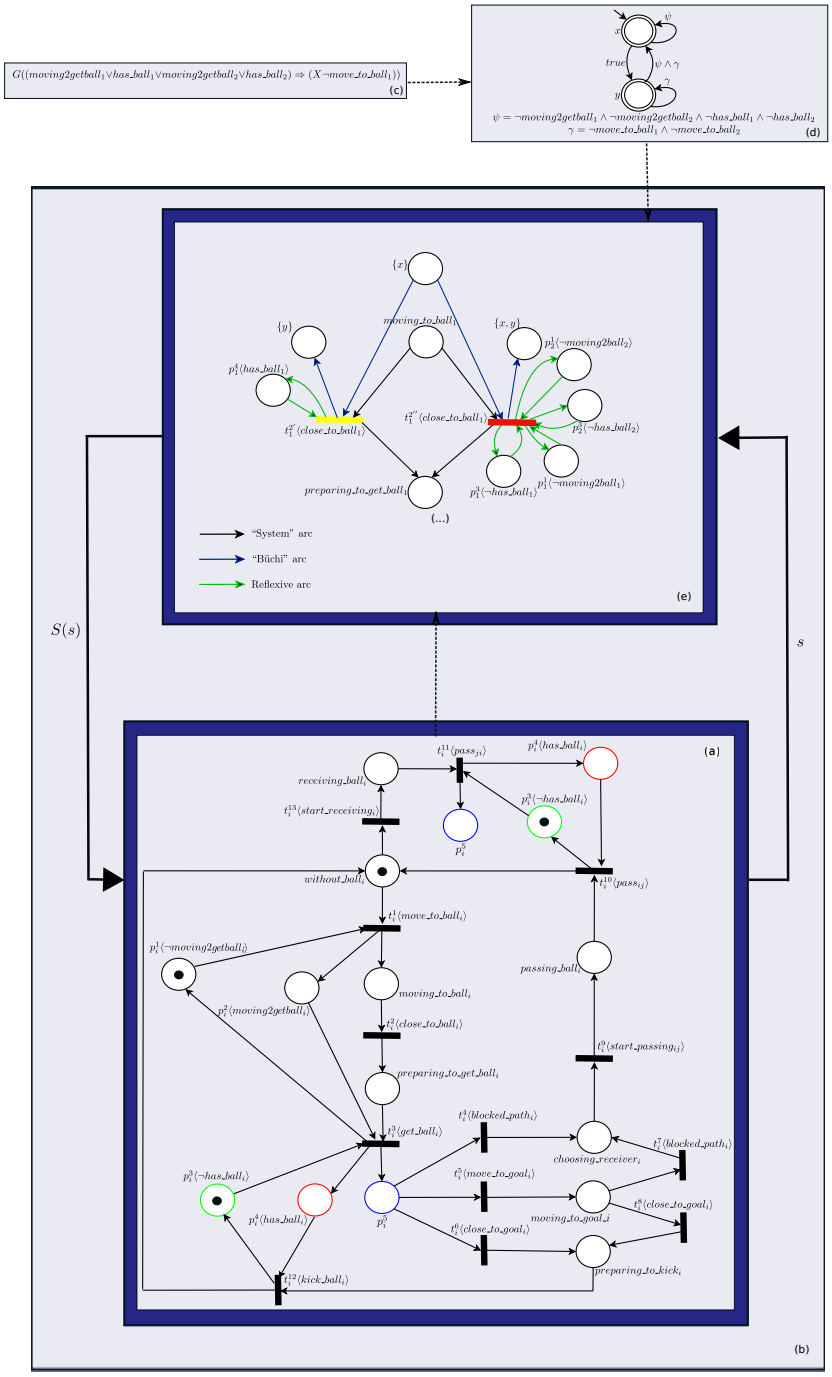


Figure 1.1: The proposed methodology (for PN models and supervisors), with the several steps of the algorithm depicted with dashed arrows and the feedback loop of supervisory control depicted with solid arrows. (a) A Petri net system model G. (b) The feedback loop of supervisory control (with a PN realizing the supervisor). (c) An LTL formula φ. (d) Büchi automaton B_φ accepting exactly the infinite sequences that satisfy φ. (e) A fragment of the supervisor S_φ that restricts the behaviour of G to the largest sub-behaviour which is consistent with φ.

a considerable number of books and collections introducing and surveying the state-of-the-art of this topic [Girault and Valk, 2002, David and Alla, 2010, Diaz, 2010, Reisig and Rozenberg, 1998a, Reisig and Rozenberg, 1998b]. In these works, one can see PNs applied to a variety of domains, such as manufacturing systems or telecommunication systems. A particular class of PNs, denominated workflow nets, is also being increasingly used to model business processes and develop process mining techniques, e.g., [van der Aalst, 1998, van der Aalst and Stahl, 2011, Weidlich and van der Werf, 2012]. This wide array of application domains illustrates the flexibility enabled by PN models and their extensions.

Regarding PN modelling in the robotics field, PNs are used for the modelling of (multi-)robot systems, due to their suitability to model distributed systems. There is a large amount of literature about modelling methodologies both for robot systems [Costelha and Lima, 2007, Milutinovic and Lima, 2002, Wang et al., 1991, Ziparo and Iocchi, 2006] and multi-robot systems [Ziparo et al., 2008, Ziparo et al., 2011, Costelha and Lima, 2012, Costelha and Lima, 2008, Sheng and Yang, 2005]. Our work in not focused on modelling, so we keep the majority of the models presented in this thesis simple, in order to facilitate the comprehension of the presented notions. In spite of that, in general, these methodologies for robot system modelling can be followed while keeping our supervisor synthesis method applicable.

In terms of qualitative analysis methodologies, the big bulk of the approaches can be classified in one of the following classes:

1. *Reachability approaches.* These approaches entail building a finite representation of the state space of the PN – the reachability graph when the PN is bounded or the coverability graph when the PN is unbounded, i.e., when its state space is infinite. While with the reachability graph, which completely represents the PN state space, one can solve all the problems that can be solved for FSA, the same is not true for the coverability graph, which provides a finite representation of an infinite state space, where some information is lost. These analysis techniques are the simplest approach to solve PN analysis problems, because they reduce them to solving a problem for FSA. Their main drawback is that they rely on explicitly building the state space of the PN, hence they are affected by the state explosion problem. Reachability approaches are described in all the introductory surveys on PNs [Girault and Valk, 2002, David and Alla, 2010, Diaz, 2010, Reisig and Rozenberg, 1998a, Reisig and Rozenberg, 1998b].

2. *Structural approaches.* These approaches have the least computational complexity, since they rely only on analysing the PN structure, thus avoiding the state explosion problem. Their main drawback is that only analysing the structure provides much less information, hence less properties can be decided. They are based on representing the PN in its matrix form and analysing it to find substructures such as traps, siphons, place invariants or transition invariants. These structures can be used to decide such properties as boundedness and liveness of the PN, independently of the initial marking. One can also find descriptions of this type of approaches in introductory surveys on PNs [Girault and Valk, 2002, David and Alla, 2010, Diaz, 2010, Reisig

and Rozenberg, 1998a, Reisig and Rozenberg, 1998b].

3. *Linear algebraic approaches.* This type of approach relies on the fact that the state evolution of a PN can be seen as linear operations on integer matrices. The methods based on this approach usually rely on translating the property to be analysed into an optimization problem, most often an integer linear program (ILP) [Schrijver, 1998]. It can be seen as a combination of the previous approaches, where one uses the PN structure and firing rule to define an ILP. Solving this ILP can be interpreted as a way to perform state exploration without explicitly building the state space. In spite of that, this approach also has computational complexity issues, since it is known that ILPs are NP-complete. A survey on these techniques is given on [Silva et al., 1998].

4. *Petri net unfolding approaches.* PN unfoldings were first introduced in [McMillan, 1992]. The goal of using PN unfoldings it to avoid the state explosion problem while maintaining the ability to analyse the PN. To achieve this, a so called occurrence net is built from the original PN. An occurrence net is a finite acyclic net that represents all the reachable markings of the PN. This structure has been used in particular for model checking of PNs [Schröter and Khomenko, 2004, Esparza, 1994, Esparza and Heljanko, 2001]. Of particular interest is [Esparza and Heljanko, 2001], where the composition procedure used has many similarities with the one used in this work. The differences are that there, the Büchi automaton "reads" the marking of the PN and then evolves to a new state accordingly, while we evolve the Büchi automaton as the PN changes state, disabling the firing of some events that would not satisfy the specification. Furthermore, we allow for bounded places with an arbitrary number of tokens while the composition presented in [Esparza and Heljanko, 2001] only allows safe PNs. Unfoldings have also been used for analysis of specific properties, such as existence of deadlocks [Melzer and Römer, 1997] or the coverability problem [Abdulla et al., 2004].

Supervisory Control

Supervisory control (SC) theory of DES was first introduced in the seminal paper [Ramadge and Wonham, 1987]. A posterior overview paper by the same authors can be found in [Ramadge and Wonham, 1989]. This is known as the Ramadge-Wonham framework and most of the works on the control of DES ever since have been based on it. It provides an analogy to DES of the classical control theory for continuous systems. More recent books dealing with SC theory are [Cassandras and Lafortune, 2006, Kumar and Garg, 1995, Seatzu et al., 2012]. The reference [Seatzu et al., 2012] is a very recent collection that deals with SC of both FSA and PN and provides a thorough look on the state-of-the-art of this topic.

Regarding temporal logic based SC of FSA models, [Jiang and Kumar, 2006b, Jiang and Kumar, 2006a] describe a methodology for building a state-based supervisor from temporal logic specifications. In this approach both the system and the goal specifications are encoded as a temporal logic formula, which is in turn translated into an FSA that satisfies it. The work in [Gromyko et al., 2006]

presents a tool to perform controller synthesis using the NuSMV [Cimatti et al., 2002] model checking tool. Contrary to our method, the temporal logic formulas are written only over the state space of the system, thus direct reasoning about sequences of events is not allowed. In [Lacerda and Lima, 2008] a similar method, that allows writing specifications only about events is described. While the applicability of these works is limited to FSA models, ours offers the possibility of building PN supervisors, which are, in general, more compact and.

There has also been some more work linking temporal logics and FSA outside of the SC scope. Instead, planning algorithms over a domain given as an FSA, where the states correspond to sets of propositional symbols and the goal is given as a temporal logic formula over those symbols, are defined [Pistore and Traverso, 2001, De Giacomo and Vardi, 2000, Cimatti et al., 2003]. This, of course, requires a state exploration to obtain the plan, which is something we will not be dealing with in our work. Our goal is to keep the system "inside" a given type of behaviour, hence we can avoid the complexity issues that arise with state space exploration.

The control of PNs has also been the subject of study. A comprehensive survey on available methods for control of PNs, including controlled PNs, a discussion on SC with language specifications and supervision based on place invariants is provided in [Holloway et al., 1997].

The use of controlled PNs, was first introduced in [Krogh, 1987, Ichikawa and Hiraishi, 1988]. Controlled PNs extend classical PNs by introducing external enabling conditions, denominated control places. A (state-based) control policy for this kind of PNs is a function that maps each marking to a token distribution in the control places. Thus, the tokens in these places do not change due to the firing rule of the PN, as with the tokens in standard places. Their token distribution for each marking is given by the control policy. In our work, we aim to have the supervisor realized as a PN so that the closed-loop behaviour of the system can be analysed using standard PN techniques. For this reason we will not get into more details on controlled PNs.

The amount of work on SC of PNs based on the Ramadge-Wonham framework, i.e., with language specifications, is not very extensive. This is probably due to some undesirable properties of PN languages. For example, in general PN languages are not closed under the supremal controllable sublanguage operator [Giua and DiCesare, 1994a], which makes it impossible to define general methods to find least restrictive admissible supervisors realized as PNs. In spite of that, in some cases it is possible to design PN supervisors based on the Ramadge-Wonham framework and some work has been done on defining a SC theory of PNs based on that framework [Giua and DiCesare, 1994b, Giua and DiCesare, 1991, Giua and DiCesare, 1995]. For a thorough state-of-the-art compilation of these results, we refer the reader to [Giua, 2013]. Furthermore, it is shown there that it is possible to check if the supervisor is admissible and non-blocking. These results will be used in our work, hence we will describe them in greater detail in Chapter 5.

In [Kumar and Holloway, 1996], it is shown that the problem of synthesizing the minimally restrictive supervisor so that the controlled system generates the supremal controllable sublanguage is reducible to a forbidden marking problem. Furthermore, a state-based control policy for minimally

restrictive supervision is defined. This control policy has the drawback of requiring the online calculation of coverability of a set of markings at each step, which is an exponential calculation in the number of places of the PN.

A widely used approach for supervisory control of PNs is the so called *supervision based on place invariants* (SBPI) or *generalized mutual exclusion constraints* (GMEC) [Giua et al., 1992, Iordache and Antsaklis, 2006b, Wu et al., 2002, Iordache and Antsaklis, 2002, Moody and Antsaklis, 1998, Iordache and Antsaklis, 2006a]. In this approach, the specifications are written as linear constraints on the reachable markings of the system and the number of firings of each transition. This line of work has a good theoretical foundation and deals with several important notions of the theory of SC, such as controllability [Basile et al., 2006], observability [**?**, Moody and Antsaklis, 1999], deadlock avoidance [Iordache et al., 2002]. This approach has some properties that we also want to include in our methodology:

- The supervisor is realized as a PN, enabling the use of PN analysis methods to study the closed-loop system. Note that, as stated before, it is proven in [Giua and DiCesare, 1994a] that having PN realizations of supervisors negates the possibility of obtaining least restrictive supervisors. In spite of that, we argue that being able to use all the PN analysis tools to study the closed-loop behaviour of the system and having a PN implementation of the control-law, which does not require extensive calculations to be executed online, makes up for this drawback.

- It is a structural approach, i.e., the construction of the supervisor is done by only takes places and transitions of the PN model of the plant, avoiding the state space explosion problem.

A major difference between GMEC and the work presented here is related with the way the rules to be fulfilled are specified. We argue that the use of LTL as the specification language is more suitable than writing linear constrains, especially for specification of coordination rules for multi-robot systems. We will further discuss the relation between our approach and GMEC in Chapter 5.

There is also work on enforcing liveness in PNs, i.e., using SC to force all the transitions of the PN to remain eventually fireable. The work of [Sreenivas, 1997, Sreenivas, 2012] deals with the foundations of this problem while [Iordache and Antsaklis, 2003] uses the SBPI framework to solve it in a non least restrictive manner.

In the robotics domain DES modelling and supervision have been used for a number of applications. In [Lee et al., 2005], a technique to design deadlock-free PN supervisors that avoid collisions between robots is presented. This method is based on defining regions that are considered shared resources, and then appropriately building PNs that disallow the usage of these resources by more than one robot. The method is restricted to this application, while ours allows the specification of more general behaviours. In [Kosecka and Bogoni, 1994], the authors present a DES-based framework for modelling behaviours and tasks for heterogeneous robotic agents, presenting examples for common robot tasks such as navigation and obstacle avoidance. The work of [Ricker et al., 1996] deals with performance of dexterous manipulation. A more recent work [Chen et al., 2012a] uses GRAFCET, a PN based graphical programming language for hardware implementation.

Temporal Logic Based Control of Robot Tasks

In the last years, there has been a great deal of work based formal approaches to robotics. Most of this work is inspired in symbolic control, with [Belta et al., 2007] providing a survey of the state-of-the-art at the time and also posing the challenges to be tackled in the field, providing inspiration for most of the succeeding work. The main goal of this type of approach is to automatically build robot controllers from high level specifications, most commonly given in LTL.

Regarding motion planning methods, where the goals are defined as LTL formulas, there has been work both on robot [Kloetzer and Belta, 2008b, Fainekos et al., 2005] and multi-robot systems [Kloetzer and Belta, 2006, Kloetzer and Belta, 2007, Kloetzer and Belta, 2008a, Loizou and Kyriakopoulos, 2004]. These works have been extended to handle the uncertainty inherent to the mobile robotics domain, by using stochastic models such as Markov decision processes [Ulusoy et al., 2012c, Cizelj and Belta, 2012, Chen et al., 2012b, Lahijanian et al., 2009] and also to obtain robust optimal paths when there is uncertainty on the robot travelling times, both in single-robot [Smith et al., 2011, Wolff et al., 2012] and multi-robot [Ulusoy et al., 2012a, Ulusoy et al., 2012b] scenarios. Using LTL to define the goals allows the specification of not only a goal region but also more intricate movements such as visiting a set of regions sequentially or travel between regions infinitely often. In these works, the LTL formulas only state when certain regions of a workspace should be visited or avoided. Furthermore, these approaches do not allow sensor readings, building controllers that are not able to react to changes in the environment.

LTL has also been used in a more classical control-theoretic approach, for hybrid control. In [Tabuada and Pappas, 2006, Tabuada and Pappas, 2003], the design of controllers enforcing LTL formulas for linear systems, based on the existence of finite bisimulations for linear systems, is described. Other works dealing with this topic are [Aydin Gol et al., 2012, Fainekos et al., 2006].

The work in [Kress-Gazit et al., 2009, Kress-Gazit et al., 2008, Kress-Gazit et al., 2007b] also deals with motion planning with temporal logic goals but allowing the robot to also react to sensor readings and perform actions other than navigation. This work has been extended to handle partially unknown environments [Sarid et al., 2012], uncertainty in sensor readings [Johnson et al., 2012] and to define a framework as user-friendly as possible, by providing the designer with the ability to define the specifications in structured english [Kress-Gazit et al., 2007a] and giving feedback when the specification cannot be satisfied [Raman and Kress-Gazit, 2012a, Raman and Kress-Gazit, 2012b]. In these approaches, both the system and the goal specifications are encoded as a temporal logic formula, while we encode the system as an FSA or a PN. For the case of PN modelling specifically, this allows for the modelling of larger and more distributed systems.

1.3 Thesis Goals and Contributions

The goal of our methodology to design supervisors is to provide the designer with a formal framework under which:

1. The unsupervised system is modelled by the designer using a DES representation (FSA or PN), and represents all possible system evolutions in its environment;

2. An FSA or PN supervisor realization is automatically constructed for each specification, written by the designer as an LTL formula that expresses a desired behaviour for a given task to be carried out.

The main contributions of this work can be summarized in the following points:

- A semantics to evaluate LTL formulas over PNs, where one can reason both about linear combinations of the number of tokens in each place plus the labels of transitions, is presented.

- A composition between the Büchi automaton obtained from an LTL formula and a PN model, that yields a PN that restricts the behaviour of the original PN to the one that satisfies the LTL formula is defined.

- A discussion on approaches to obtain admissible and deadlock-free supervisors is provided. The standard notion of uncontrollable marking usually used in the literature is shown to be incorrect. A new, sound version is proposed and some results for checking admissibility for deterministic PN supervisors are proved for the new definition.

- An application of PN-based SC to a real world multi-robot scenario is described.

1.4 Document Organization

We start the remainder of the work by introducing the modelling formalisms, in Chapter 2. We will define three different system models: FSA where the states have a symbolic propositional description, PNs where the states have a symbolic propositional description, and PNs where the states have an algebraic propositional description.

In Chapter 3, we focus on describing the formalism used to write the specifications that the system models must fulfil. We introduce LTL, define its semantics for each of the system models and discuss safe LTL formulas, the fragment of LTL that will be used in the methodology.

Chapters 4 and 5 contain the main theoretical contributions of the thesis. In Chapter 4 we provide algorithms for composing the system models with the Büchi automata obtained from the LTL formulas, creating a coarse structure for the supervisor. In Chapter 5 we focus on ways of refining that structure so that the supervisor realizations are admissible and deadlock-free. For the case of PNs we also show that the notion of uncontrollable marking usually used in the procedures to check for supervisor admissibility is not soundly defined, by means of a counter example. Then we redefine it in a proper way, and present algorithms to check for admissibility and deadlock-freeness based on this new notion of uncontrollable marking. We also point out the limitations of PNs in the scope of SC theory and discuss ways of circumventing these limitations.

In Chapter 6, we show the more practical contributions of the work, which are focused on the application of the methodology to single and multi-robot systems:

- First, a simulated soccer scenario, where a team of robots must coordinate passes and movements towards the ball. In this example, we show how to build (centralized) FSA supervisors and both centralized and decentralized PN supervisors, comparing the sizes of the obtained structures and the scalability of the different approaches.
- Second, an implementation on a real social robot where interaction rules with a human are given to the robot in the form of LTL formulas.
- Finally, a larger-scale scenario, where a team of real robots must coordinate in order to perform surveillance tasks, in a mock-up scenario, is tackled.

Finally, in Chapter 7, a discussion about the methodology is provided, along with a discussion of what was accomplished and an extensive list of possible future work routes.

CHAPTER 2

System Modelling

In this chapter, we introduce the modelling formalisms. We start by showing how to model a system using FSA equipped with a state description function. Then we show two approaches for PN modelling of the system. First, a version that mirrors the FSA model, where we use a subset of the places of the PN to represent the truth value of atomic propositions, followed by a more general version where the atomic propositions are considered to be linear inequalities over markings. In the following, let Σ be a finite alphabet, Σ^* the set of all finite strings built from Σ (including the empty string ε) and Σ^ω the set of all infinite strings that can be build from Σ. Also, given a finite set A, let $|A| \in \mathbb{N}$ be the number of elements of A.

2.1 Finite State Automata

In this section, we describe finite state automata (FSA), one of the tools we will use to model the systems. We also equip the FSA with a set of state description symbols and a function that maps each state of the FSA to the state description symbols that are true in that state. This state description symbols will, in conjunction with the set of events, be used to describe the behaviours of the system.

Definition 2.1.1 (Finite State Automaton)**.** A (deterministic) finite state automaton (FSA) is a tuple $A = \langle Q, \Sigma, \delta, Q_0 \rangle$ where:

- Q is a finite set of states;
- Σ is a finite alphabet;
- $\delta : Q \times \Sigma \to Q$ (deterministic case) or $\delta : Q \times \Sigma \to 2^Q$ (non-deterministic case) is the possibly partial transition function;

- $Q_0 = q_0 \in Q$ is the initial state (deterministic case) or $Q_0 \subseteq Q$ is the set of possible initial states (non-deterministic case).

To simplify the notation, we define the active symbol function, a function that maps each state of an FSA to the symbols that are active in that state.

Definition 2.1.2 (Active Symbol Function)**.** Let $A = \langle Q, \Sigma, \delta, q_0 \rangle$ be an FSA. The active symbol function $\Gamma_A : Q \to 2^\Sigma$ is defined as:

$$\Gamma_A(q) = \{a \in \Sigma \mid \delta(q,a) \text{ is defined}\} \tag{2.1}$$

We will use deterministic FSA to model DES. In DES models, the alphabet is a set of events[1] E and, in order to have a propositional description of the states, we equip the FSA with a set of state description symbols.

Definition 2.1.3 (FSA System Model)**.** An FSA system model is a tuple $G = \langle A, D, \mu \rangle$ where:

- $A = \langle Q, E, \delta, q_0 \rangle$ is a deterministic FSA;
- D is a finite set of state description symbols;
- $\mu : Q \to 2^D$ is a function that maps each state into the set of state description symbols that are true in that state.

We will be interested in the ω-language generated by FSA system models, which represents the set of all possible infinite behaviours of the system. The language will contain both the sequence of events and states visited.

Definition 2.1.4 (ω-Language Generated by an FSA System Model)**.** Let $G = \langle Q, E, \delta, q_0, \rangle$ be an FSA. The language generated by G is defined as:

$$\mathscr{L}(G) = \{\sigma = (e_1,q_1)(e_2,q_2)... \in (E \times Q)^\omega \mid q_{i+1} = \delta(q_i, e_{i+1}) \text{ for all } i \in \mathbb{N}\} \tag{2.2}$$

The members of $\mathscr{L}(G)$ are the infinite sequences of events and states that can occur while starting in q_0 and following the transition function δ, i.e., for $t \in \mathbb{N}$, $\sigma_t = (e,q)$ means that the system is in state q at time[2] t, and we reached q by firing event e. Note that q_1 must be reachable from the initial state q_0 through the firing of e_1. We will deal with the fact that q_0 does not appear in the language by adding a dummy state, as will be explained later.

[1] We defined both deterministic and non-deterministic FSA because we will use deterministic FSA to model the system, but we will need the notion of non-deterministic automata later, when we discuss Büchi automata. In this section, we assume that all the FSA are deterministic. We also left the alphabet set arbitrary because for Büchi automata the alphabet is not a set of events. We will refer to the active symbol function as active event function when dealing with FSA models of DES.

[2] In this work, we consider time as discrete steps, and that the system is in the initial state at time 0.

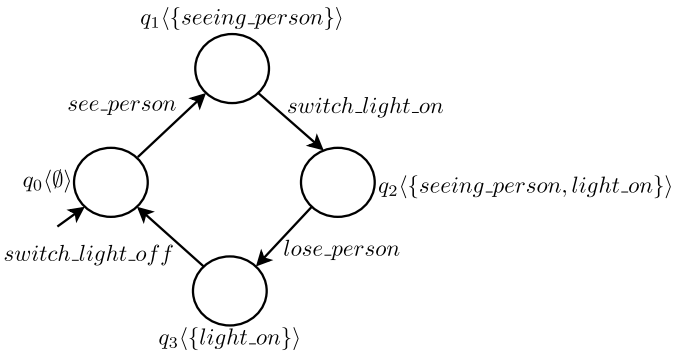


Figure 2.1: FSA System Model for Robot with a Light Actuator

Example 2.1.1 (Robot with a Light Actuator)**.** Figure 2.1 depicts an FSA system model for a robot that can detect a person and turn on its light while it is seeing that person. The state description symbols are $D = \{seeing\_person,\ light\_on\}$. The initial state is depicted with a small arrow targeting it and the state labels for each state q are of the form $q\langle\mu(q)\rangle$.

In general, we will model different modules of the system as separate FSA and compose them to obtain the FSA system model for the whole system. The composition operator used is the usual parallel composition, which models the concurrent execution of the modules, where common events between them must be synchronized.

Definition 2.1.5 (Parallel Composition)**.** Let $G_1 = \langle Q_1, E_1, \delta_1, q_{0,1}, Q_{F,1}, D_1, \mu_1\rangle$ and $G_2 = \langle Q_2, E_2, \delta_2, q_{0,2}, Q_{F,2}, D_2, \mu_2\rangle$ be FSA system models. The parallel composition of G_1 and G_2 is the FSA system model $G_1 \parallel G_2 = \langle Q_1 \times Q_2, E_1 \cup E_2, \delta, (q_{0,1}, q_{0,2}), Q_{F,1} \times Q_{F,2}, D_1 \cup D_2, \mu\rangle$, where:

$$\delta((q_1,q_2),e) = \begin{cases} (\delta_1(q_1),\delta_2(q_2)) & \text{if } e \in \Gamma_{G_1}(q_1) \cap \Gamma_{G_2}(q_2) \\ (\delta_1(q_1),q_2) & \text{if } e \in \Gamma_{G_1}(q_1) \setminus E_2 \\ (q_1,\delta_2(q_2)) & \text{if } e \in \Gamma_{G_2}(q_2) \setminus E_1 \\ \text{undefined} & \text{otherwise} \end{cases} \tag{2.3}$$

$$\mu((q_1,q_2)) = \mu_1(q_1) \cup \mu_2(q_2) \tag{2.4}$$

As usual, we only take into account the accessible states, i.e., the elements of $Q_1 \times Q_2$ that can be reached from $(q_{0,1}, q_{0,2})$ while following δ.

Example 2.1.2 (Robot with a Light and a Sound Actuator)**.** Building upon the previous example, we now consider that the robot also has a sound actuator, which is turned on when the robot is not seeing a person and turned off otherwise. The model for the sound actuator is given in Figure 2.2, and Figure 2.3 depicts the model for the two actuators, given by the parallel composition of the models for each of them.

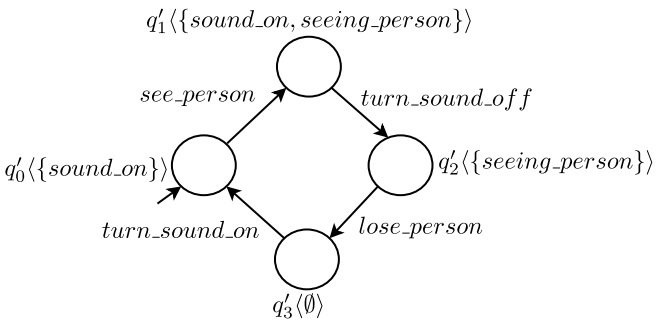


Figure 2.2: FSA System Model for a Robot with a Sound Actuator

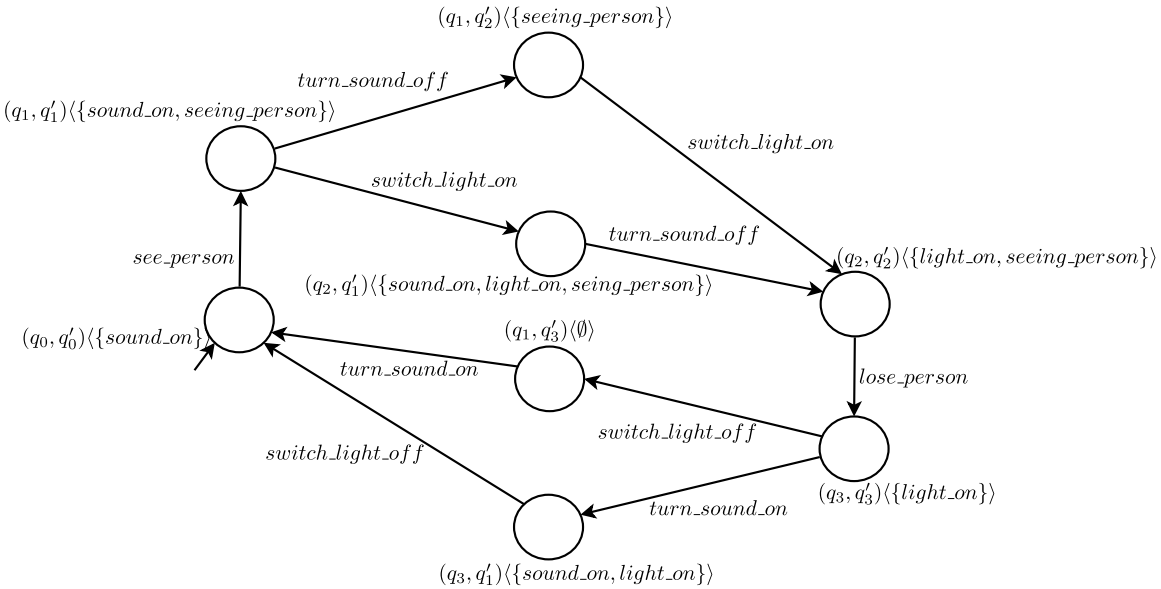


Figure 2.3: FSA System Model for a Robot with a Light and Sound Actuator

2.2 Petri Nets

The use of FSA allows us to model a good amount of systems, but, especially when dealing with distributed systems, the need to enumerate all possible states of the system in the model becomes both cumbersome and computationally inefficient. Changing the modelling formalism to Petri nets (PN) allows us to model richer systems, since PNs are known to be more expressive than FSA, in terms of generated languages, and also provides a more compact way of describing our systems because, using PNs, there is no need of enumerating all the states the system can be in. We argue that, being more compact, the use of PNs to model the uncontrolled system allows the control of more complex systems than both modelling the system as an FSA or the encoding of a system as a temporal logic formula, being the best suited formalism (of the ones referred here) to model multi-robot systems.

We start by providing a brief overview on PN structures, starting by the definition of the different elements of a PN.

Definition 2.2.1 (Petri Net Structure)**.** A Petri net structure is a tuple $N = \langle P, T, W^-, W^+, M_0 \rangle$ where:

- P is a finite, not empty, set of places;
- T is a finite, not empty, set of transitions;
- $W^- \in \mathbb{N}^{|P| \times |T|}$ is the input matrix;
- $W^+ \in \mathbb{N}^{|P| \times |T|}$ is the output matrix;
- $M_0 \in \mathbb{N}^{|P|}$ is the initial marking.

The input matrix represents arc weights between places and transitions while the output matrix represents arc weights between transitions and places. This means that a PN structure is a weighted bipartite graph, where each node is either a place or a transition. The initial marking M_0 is a vector of size $|P|$ that represents the initial state of the system, with $M_0(p) = q$ meaning that there are q tokens in place p in the initial state.

We will use the places and transitions themselves as the indices of the matrices and vectors, e.g., given $p \in P$ and $t \in T$, we use $W^-(p,t)$ to represent the entry W^-_{ij} that corresponds to the arc weight from p to t. We will also use the *incidence matrix* $W = W^+ - W^-$. Note that, in general an incidence matrix W does not uniquely define a pair W^- and W^+ of input and output matrices. However, when the PN is self-loop free – i.e., if $W^+(p,t) > 0$, then $W^-(p,t) = 0$ and if $W^-(p,t) > 0$, then $W^+(p,t) = 0$ – the incidence matrix is enough to uniquely define W^- and W^+:

$$W^-(p,t) = \begin{cases} -W(p,t) & \text{if } W(p,t) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

$$W^+(p,t) = \begin{cases} W(p,t) & \text{if } W(p,t) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

We will also use the notions of pre and post set of a node in a PN.

Definition 2.2.2 (Presets and Postsets)**.** Let $N = \langle P, T, W^-, W^+, M_0 \rangle$ be a PN structure, $p \in P$ and $t \in T$. We define the following vectors:

- The preset of p, ${}^\bullet p \in \mathbb{N}^{|T|}$, such that ${}^\bullet p(t) = W^+(p,t)$. If ${}^\bullet p(t) > 0$, we say that t is in the preset of p.
- The postset of p, $p^\bullet \in \mathbb{N}^{|T|}$, such that $p^\bullet(t) = W^-(p,t)$. If $p^\bullet(t) > 0$, we say that t is in the postset of p.
- The preset of t, ${}^\bullet t \in \mathbb{N}^{|P|}$, such that ${}^\bullet t(p) = W^-(p,t)$. If ${}^\bullet t(p) > 0$, we say that p is in the preset of t.
- The postset of t, $t^\bullet \in \mathbb{N}^{|P|}$, such that $t^\bullet(p) = W^+(p,t)$. If $t^\bullet(p) > 0$, we say that p is in the postset of t.

The dynamics of a PN are defined by the firing rule, which determines the flow of tokens between places, thus specifying how the initial marking can evolve.

Definition 2.2.3 (Firing Rule)**.** Let $N = \langle P, T, W^-, W^+, M_0 \rangle$, $t \in T$ and $M \in \mathbb{N}^{|P|}$. Transition t is said to be active in M if for all $p \in P$, ${}^\bullet t(p) \leq M(p)$. A transition t active in a marking M can fire, resulting in the marking $M' = M - W^-(.,t) + W^+(.,t) = M - {}^\bullet t + t^\bullet$. This is denoted $M \xrightarrow{t} M'$.

Using the firing rule, one can define firing sequences and the set of reachable markings of a given PN structure.

Definition 2.2.4 (Firing Sequence)**.** Let $N = \langle P, T, W^-, W^+, M_0 \rangle$ be a PN structure. A finite firing sequence from a given marking M is a sequence of transitions $\tau = t_1 t_2 ... t_n \in T^*$ such that there exists markings $M_1, ..., M_n$ such that:

$$M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_2} ... \xrightarrow{t_n} M_n \tag{2.7}$$

An infinite firing sequence from a given marking M is a sequence of transitions $\tau = t_1 t_2 ... \in T^\omega$ such that there exists markings $M_1, M_2, ...$ such that:

$$M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_2} ... \tag{2.8}$$

Finite firing sequences will be used to define reachable markings in the PN, while infinite firing sequences will be used to define the language generated by the PN.

Definition 2.2.5 (Markings Notation)**.** Let $N = \langle P, T, W^-, W^+, M_0 \rangle$, $M \in \mathbb{N}^{|P|}$ and $\tau \in T^*$. We write:

- $M \xrightarrow{\tau} M'$ to denote that the firing sequence τ drives N from marking M to marking M';
- $(M \xrightarrow{\tau})$ to denote that N can fire the sequence τ from M;

- (If $(M_0 \xrightarrow{\tau})$) M_τ to denote the marking reached after firing τ from M_0.

Definition 2.2.6 (Reachable Markings)**.** The set of all reachable markings by $N = \langle P, T, W, M_0 \rangle$ is denoted as:

$$R(N) = \{M_\tau \mid \tau \in T^* \text{ and } (M_0 \xrightarrow{\tau})\} \tag{2.9}$$

Note that all reachable markings are of the form $M_\tau = M_0 + W w_\tau$, where w_τ is the vector of natural numbers of size $|T|$ for which the i-th entry is the number of occurrences in τ of the i-th transition. This vector is denominated *firing count vector*. However, the converse is not true: in general, there exists $w \in \mathbb{N}^{|T|}$ such that $M_0 + Ww$ is not a reachable marking. This is due to the fact that a transition t is only active when ${}^\bullet t \leq M$.

We will be interested in the languages generated by PNs, hence we add labels to the transitions. Furthermore, since we will only use PNs to model DES, we can already assume that the alphabet set is a set E of events.

Definition 2.2.7 (Labelled Petri Net)**.** A labelled Petri net is a tuple $L = \langle N, E, \ell \rangle$ where:

- $N = \langle P, T, W^-, W^+, M_0 \rangle$ is a PN structure;
- E is the finite set of events;
- $\ell : T \to E$ is the labelling function, that assigns to each transition an event from E.

We extend the labelling function to a function $\ell : T^* \to E^*$ and $\ell : T^\omega \to E^\omega$ as usual:

- $\ell(t_1 ... t_n) = \ell(t_1) ... \ell(t_n)$;
- $\ell(t_1 t_2 ...) = \ell(t_1)\ell(t_2) ....$

We require the PNs is this work to be deterministic, in the sense that a sequence of labels uniquely defines a firing sequence, hence also uniquely defining the sequence of visited markings. This is done because we will use the PN model to build the PN realization of the supervisor. As will be seen later, a supervisor is formally a function, and, in order to represent a function, a PN must be deterministic, because otherwise the result for a given input will not be unequivocally defined.

Definition 2.2.8 (Deterministic Petri net)**.** A labelled PN L is deterministic if for all $t, t' \in T$ and $M \in R(L)$:

$$\text{If } \begin{cases} M \xrightarrow{t} M' \\ M \xrightarrow{t'} M'' \\ M' \neq M'' \end{cases} \text{ then } \ell(t) \neq \ell(t') \tag{2.10}$$

Definition 2.2.9 (Label-Transition Mapping)**.** Let L be a deterministic labelled PN. We define the function $\ell^{-1} : E^* \to T^*$ that maps a sequence of labels into a sequence of transitions as:

$$\ell^{-1}(s) = \begin{cases} \tau & \text{if exists } \tau \in T^* \text{ such that } \ell(\tau) = s \text{ and } (M \xrightarrow{\tau}) \\ \text{undefined} & \text{otherwise} \end{cases} \tag{2.11}$$

Note that $\ell^{-1}(s)$ is well-defined, in the sense that given $s \in E^*$ there is at most one $\tau \in T^*$ such that $\ell(\tau) = s$. This is a direct consequence of assuming that the PNs are deterministic. Hence, if $\ell^{-1}(s)$ is defined, $M_{\ell^{-1}(s)}$ is the marking reached after executing the firing sequence uniquely defined by s, starting from M_0. As with FSA, we also define the active event function in order to simplify the notation.

Definition 2.2.10 (Active Event Function)**.** Let $L = \langle P, T, W^-, W^+, M_0, E, \ell \rangle$. We define the function $\Gamma_L : R(L) \to 2^E$ as:

$$\Gamma_L(M) = \{e \in E \mid \text{exists } t \in T \text{ such that } \ell(t) = e \text{ and } t \text{ is active in } M\} \tag{2.12}$$

As with FSA system models, we will be interested in the ω-language generated by labelled PNs, which represents the set of all possible infinite behaviours of the system. The language will also contain both the sequence of events and states visited, taking into account that in this case the states visited by the system are represented by markings.

Definition 2.2.11 (Language Generated by a Labelled Petri Net)**.** Let $L = \langle N, E, \ell \rangle$ be a labelled PN. The language generated by L is defined as:

$$\mathscr{L}(L) = \left\{ (\ell(t_1), M_1)(\ell(t_2), M_2)... \in (E \times R(L))^\omega \mid \text{such that } M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} ... \right\} \tag{2.13}$$

The members of $\mathscr{L}(L)$ are the infinite sequences of event and marking pairs that can occur while starting in M_0 and following the PN firing rule, i.e., for $t \in \mathbb{N}$, $\sigma_t = (e, M)$ means that the system is in marking M at time t, and we reached M by firing a transition labelled by event e. As with the language generated by FSA system models, note that M_1 must be reachable from the initial marking M_0 through the firing of t_1. We will deal with the fact that M_0 does not appear in the language by adding a dummy place and a dummy transition, and changing the initial marking, as will be explained later.

We will use two different approaches for modelling DES as PNs:

- A *symbolic approach*, similar to the FSA approach. In this approach, the markings are described by a set of state description symbols. This is achieved by defining a subset of places that correspond to the truth value of state description symbols.

- An *algebraic approach*, which takes advantage of the fact that markings have an algebraic description. We will use this fact to define the atomic propositions for our specifications as linear constraints over the markings.

It is important to note that the symbolic approach is subsumed by the algebraic approach, as we will show later. In spite of that, we choose to present it as a stepping-stone to the most general algebraic approach, given that it is a direct translation of the approach for FSA[3]. Furthermore, (i) the work we will present later about the decentralization of the SC approach is based on PN system models with symbolic representation and (ii) we will compare the two approaches in an example, providing a definite argument for using linear constraints over the markings as the atomic propositions when dealing with logical approaches to PNs.

2.2.1 Symbolic State Description

Before we define the PN system model with symbolic state representation, we need to define the notion of a literal. Literals will be used to map the truth value of state description symbols into the marking in a given place.

Definition 2.2.12 (Literal)**.** Let Π be a set of atomic propositions. We define the set $lit(\Pi) = \{\pi, \neg\pi \mid \pi \in \Pi\}$. A positive literal is of the form $\pi \in lit(\Pi)$ and a negative literal is of the form $\neg\pi \in lit(\Pi)$.

We can now define the PN system model with symbolic state description.

Definition 2.2.13 (Petri Net System Model – Symbolic State Description)**.** A Petri net system model is a tuple $G = \langle L, D, \mu \rangle$ where:

- $L = \langle N, E, \ell \rangle$ is a labelled PN where $P = P_D \cup P_g$, with $P_D \cap P_g = \emptyset$;
- D is a set of state description symbols;
- $\mu : lit(D) \to P_D$ is a bijection[4], such that:

$$\text{For all } d \in D,\ M(\mu(\neg d)) + M(\mu(d)) = 1 \tag{2.14}$$

The set P_D is the set of places corresponding to truth values of state description symbols and P_g is the set of general places, which do not influence the state description. The bijection μ has the following meaning: if $M(\mu(d)) = 1$ then d is satisfied in marking M and if $M(\mu(\neg d)) = 1$ then d is not satisfied in marking M. Note that, for each $d \in D$, the PN system models always have one token in one of the places representing a truth value for d and zero tokens in the other.

We define the set of true state description symbols in a given marking, which will be used to symbolically represent the markings of the system

[3]Note that this follows the chronology of our work, where we first defined a methodology for FSA models, followed by an approach for PN models with symbolic state representation and, finally, an approach for PN models with algebraic state representation. For this reason, we feel that this is the most proper way to facilitate the reader's understanding of the methodology.

[4]Hence, each $p \in P_D$ corresponds to exactly one literal $l \in lit(D)$, thus $|P_D| = 2|D|$.

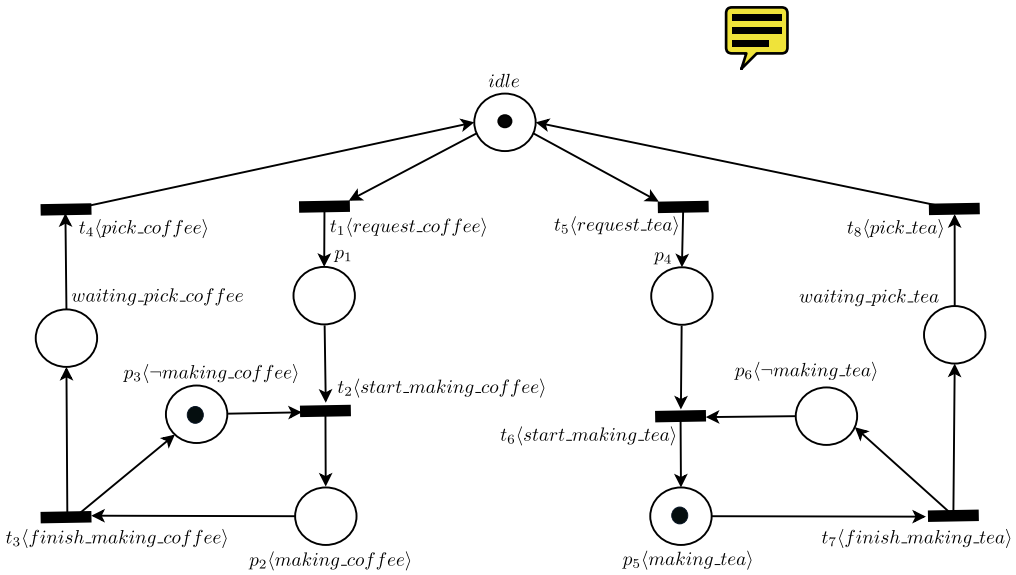


Figure 2.4: PN system model with symbolic state description for a tea/coffee machine

Definition 2.2.14 (Set of True State Description Symbols in a Marking)**.** Let $G = \langle L, D, \mu \rangle$ and $M \in R(G)$. The set of true propositional symbols in M is given by:

$$D_M = \{d \in D \mid M(\mu(d)) = 1\} \tag{2.15}$$

Example 2.2.1 (Tea/Coffee Machine)**.** Consider the model in Figure 2.4 for a machine that serves coffee and tea upon user request. The machine is in an idle state until someone requests tea or coffee. After the request, the machine prepares the appropriate drink, After the drink is picked up, the machine returns to the idle state. Note that in the marking depicted, the machine is not in the idle state, but rather preparing a cup of tea.

In this example, we assume that $P_D = \{p_2, p_3, p_5, p_6\}$, thus these are the only places with an associated state description literal. Note that it is up to the designer to decide which are the description symbols needed to describe the markings, and to create places in the model that represent the truth value of these symbols. The event and state propositional description sets are the following:

$$\begin{aligned} E = \{&request\_coffee, start\_making\_coffee, finish\_making\_coffee, pick\_coffee, \\ &request\_tea, start\_making\_tea, finish\_making\_tea, pick\_tea\} \end{aligned} \tag{2.16}$$

$$D = \{making\_coffee, making\_tea\} \tag{2.17}$$

The labels for places are of the form $p\langle\mu^{-1}(p)\rangle$, where $\mu^{-1}(p)$ is the literal l such that $\mu(l) = p$. In cases where there is no such literal, i.e., for places in P_g, we simply omit the $\langle . \rangle$ part of the label. Note that the names given to the places are irrelevant, and are only used to improve readability. For example, place *idle* could be called simply p_0. The labels for transitions are of the form $t\langle\ell(t)\rangle$. For the marking M depicted in the figure, $D_M = \{making\_tea\}$.

As with FSA, we finish the presentation of the symbolic state description for PNs, we describe the parallel composition of PN models. In the following, let $G_1 = \langle P_1 = P_{g,1} \cup P_{D,1}, T_1, W_1^-, W_1^+, M_{0,1}, E_1, \ell_1, D_1, \mu_1 \rangle$ and $G_2 = \langle P_2 = P_{g,2} \cup P_{D,2}, T_2, W_2^-, W_2^+, M_{0,2}, E_2, \ell_2, D_2, \mu_2 \rangle$ be PN models.

We start by defining the set of all pairs of transitions sharing the same event label and the set of all pairs of places representing the same truth value for the same propositional symbol.

Definition 2.2.15 (Shared Transitions)**.** The set of shared transitions of G_1 and G_2 is given by:

$$T_{E_1 \cap E_2} = \{(t_1, t_2) \in T_1 \times T_2 \mid \ell_1(t_1) = \ell(t_2)\} \tag{2.18}$$

Definition 2.2.16 (Shared Places)**.** The set of shared places of G_1 and G_2 is given by:

$$P_{D_1 \cap D_2} = \bigcup_{d \in D_1 \cap D_2} \{(\mu_1(d), \mu_2(d)),\ (\mu_1(\neg d), \mu_2(\neg d))\} \tag{2.19}$$

Definition 2.2.17 (Non-Conflicting Petri Net Models)**.** G_1 and G_2 are said to be non-conflicting if they satisfy the following statements:

- $P_1 \cap P_2 = \emptyset$;
- $T_1 \cap T_2 = \emptyset$;
- The initial markings assign the same truth value to the propositional symbols in $D_1 \cap D_2$:

$$\text{For all } (p_1, p_2) \in P_{D_1 \cap D_2},\ M_{0,1}(p_1) = M_{0,2}(p_2) \tag{2.20}$$

- Transition pairs in $T_{E_1 \cap E_2}$ assign the same truth value to the propositional symbols in $D_1 \cap D_2$:

$$\text{For all } (t_1, t_2) \in T_{E_1 \cap E_2} \text{ and } (p_1, p_2) \in P_{D_1 \cap D_2},\ W_1^+(p_1, t_1) = W_2^+(p_2, t_2) \tag{2.21}$$

To guarantee that we are able to soundly define the literals corresponding to shared places, we need to assume that G_1 and G_2 are non-conflicting, i.e., the firing of shared events in G_1 and G_2 yields the same truth value for the shared state description symbols of both nets. The parallel composition $G_1 \parallel G_2$ of G_1 and G_2 is obtained by simply connecting both models through the merging of pairs of transitions $(t_1, t_2) \in T_{E_1 \cap E_2}$ and the pairs of places $(p_1, p_2) \in P_{D_1 \cap D_2}$, while keeping the arc-weight functions of G_1 and G_2.

Definition 2.2.18 (Parallel Composition)**.** The parallel composition of G_1 and G_2 is the PN model $G_1 \parallel G_2 = \langle P_{1\|2}, T_{1\|2}, W_{1\|2}^-, W_{1\|2}^+, M_{0,1\|2}, E_1 \cup E_2, \ell_{1\|2}, D_1 \cup D_2, \mu_{1\|2} \rangle$, where:

- $P_{1\|2} = P_{g,1\|2} \cup P_{D,1\|2}$ is the union of non-shared places of G_1 and G_2 with the pairs representing the shared places, i.e., $P_{g,1\|2} = P_{g,1} \cup P_{g,2}$ and $P_{D,1\|2} = P_{D_1 \setminus D_2} \cup P_{D_2 \setminus D_1} \cup P_{D_1 \cap D_2}$, where:

$$P_{D_1 \setminus D_2} = \{p \in P_{D_1} \mid p = \mu_1(d) \text{ or } p = \mu_1(\neg d), \text{ with } d \in D_1 \setminus D_2\} \tag{2.22}$$

$$P_{D_2 \setminus D_1} = \{p \in P_{D_2} \mid p = \mu_2(d) \text{ or } p = \mu_2(\neg d), \text{ with } d \in D_2 \setminus D_1\} \tag{2.23}$$

- $T_{1\|2}$ is the union of the non-shared transitions of G_1 and G_2 with the pairs representing the shared transitions, i.e., $T_{1\|2} = T_{E_1 \setminus E_2} \cup T_{E_2 \setminus E_1} \cup T_{E_1 \cap E_2}$, where:

$$T_{E_1 \setminus E_2} = \{t \in T_1 \mid \ell_1(t) \in E_1 \setminus E_2\} \tag{2.24}$$

$$T_{E_2 \setminus E_1} = \{t \in T_2 \mid \ell_2(t) \in E_2 \setminus E_1\} \tag{2.25}$$

- $W^-_{1\|2} \in \mathbb{N}^{|P_{1\|2}| \times |T_{1\|2}|}$ maintains the same arc weights as W_1^- and W_2^-, taking into account that the shared places and shared transitions of G_1 and G_2 are now merged into a single place and a single transition, respectively:

$$W^-_{1\|2}(p,t) = \begin{cases} W_1^-(p,t) & \text{if } t \in T_{E_1 \setminus E_2} \text{ and } p \in P_{D_1 \setminus D_2} \\ W_2^-(p,t) & \text{if } t \in T_{E_2 \setminus E_1} \text{ and } p \in P_{D_2 \setminus D_1} \\ W_1^-(p,t_1) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p \in P_{D_1 \setminus D_2} \\ W_2^-(p,t_2) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p \in P_{D_2 \setminus D_1} \\ W_1^-(p_1,t) & \text{if } t \in T_{E_1 \setminus E_2} \text{ and } p = (p_1,p_2) \in P_{D_1 \cap D_2} \\ W_2^-(p_2,t) & \text{if } t \in T_{E_2 \setminus E_1} \text{ and } p = (p_1,p_2) \in P_{D_1 \cap D_2} \\ W_1^-(p_1,t_1) = W_2^-(p_2,t_2) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p = (p_1,p_2) \in P_{D_1 \cap D_2} \\ 0 & \text{otherwise} \end{cases} \tag{2.26}$$

  Note that, for $t = (t_1,t_2) \in T_{E_1 \cap E_2}$ and $p = (p_1,p_2) \in P_{D_1 \cap D_2}$, we have the guarantee that $W_1^-(p_1,t_1) = W_2^-(p_2,t_2)$ because the PN models are assumed to be non-conflicting. Throughout the rest of the definition we will continue to explicitly state the equalities that stem from this assumption in order to clarify the reader where the non-conflicting assumption is required for a sound definition of parallel composition.

- $W^+_{1\|2} \in \mathbb{N}^{|P_{1\|2}| \times |T_{1\|2}|}$ is defined analogously to $W^-_{1\|2}$:

$$W^+_{1\|2}(p,t) = \begin{cases} W_1^+(p,t) & \text{if } t \in T_{E_1 \setminus E_2} \text{ and } p \in P_{D_1 \setminus D_2} \\ W_2^+(p,t) & \text{if } t \in T_{E_2 \setminus E_1} \text{ and } p \in P_{D_2 \setminus D_1} \\ W_1^+(p,t_1) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p \in P_{D_1 \setminus D_2} \\ W_2^+(p,t_2) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p \in P_{D_2 \setminus D_1} \\ W_1^+(p_1,t) & \text{if } t \in T_{E_1 \setminus E_2} \text{ and } p = (p_1,p_2) \in P_{D_1 \cap D_2} \\ W_2^+(p_2,t) & \text{if } t \in T_{E_2 \setminus E_1} \text{ and } p = (p_1,p_2) \in P_{D_1 \cap D_2} \\ W_1^+(p_1,t_1) = W_2^+(p_2,t_2) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p = (p_1,p_2) \in P_{D_1 \cap D_2} \\ 0 & \text{otherwise} \end{cases} \tag{2.27}$$

- $M_{0,1\|2} : P_{1\|2} \to \mathbb{N}$ maintains the respective initial markings of G_1 and G_2, taking into account

the merging of shared places:

$$M_{0,1\|2}(p) = \begin{cases} M_{0,1}(p) & \text{if } p \in P_{D_1 \setminus D_2} \\ M_{0,2}(p) & \text{if } p \in P_{D_2 \setminus D_1} \\ M_{0,1}(p_1) = M_{0,2}(p_2) & \text{if } p = (p_1, p_2) \in P_{D_1 \cap D_2} \end{cases} \tag{2.28}$$

- $\ell_{1\|2} : T_{1\|2} \to E_1 \cup E_2$ maintains the respective transition labels of G_1 and G_2, taking into account the merging of shared transitions:

$$\ell_{1\|2}(t) = \begin{cases} \ell_1(t) & \text{if } t \in T_{E_1 \setminus E_2} \\ \ell_2(t) & \text{if } t \in T_{E_2 \setminus E_1} \\ \ell_1(t_1) = \ell_2(t_2) & \text{if } t = (t_1, t_2) \in T_{T_1 \cap T_2} \end{cases} \tag{2.29}$$

- $\mu_{1\|2} : lit(D_1 \cup D_2) \to P_{D_{1\|2}}$ maintains the relations between places and truth values of state description symbols of G_1 and G_2, taking into account the merging of shared places:

$$\mu_{1\|2}(d) = \begin{cases} \mu_1(d) & \text{if } d \in D_1 \setminus D_2 \\ \mu_2(d) & \text{if } d \in D_2 \setminus D_1 \\ (\mu_1(d), \mu_2(d)) & \text{if } d \in D_1 \cap D_2 \end{cases} \tag{2.30}$$

$$\mu_{1\|2}(\neg d) = \begin{cases} \mu_1(\neg d) & \text{if } d \in D_1 \setminus D_2 \\ \mu_2(\neg d) & \text{if } d \in D_2 \setminus D_1 \\ (\mu_1(\neg d), \mu_2(\neg d)) & \text{if } d \in D_1 \cap D_2 \end{cases} \tag{2.31}$$

The parallel composition simply models the simultaneous execution of G_1 and G_2, where common events are synchronized, i.e., a common event can only be executed if both G_1 and G_2 can execute it simultaneously. We also merge the places that correspond to the same truth value of the same propositional symbol, since they represent the same information. We will use the parallel composition on the PN models of different modules of the same system, thus obtaining the PN model for the simultaneous execution of all modules[5].

Example 2.2.2 (Two Robots Moving a Bar)**.** In Figures 2.5 and 2.6, we show two PN system models G_1 and G_2 for robots that are able to grab a bar and move it. Due to the length of the bar, it can only be moved if each robot grabs one of the bar ends. Each of the models contain the actions available to the corresponding robots, and the places describing the state of the robot (if it is holding the bar) and the state of the bar (which sides of the bar are being held, and if the bar is moving). The *move_bar* action is a shared action which abstracts the coordination needed to perform the actual movement. Hence, these models have both shared events and shared state description symbols. We list the sets used in

[5]Note that in the case of multi-robot systems which we will focus on for the application of our methodology, we can use parallel composition on models of different possible behaviours for a given single robot and also on the individual models of a team of robots. This will be exemplified on Section 6.3.

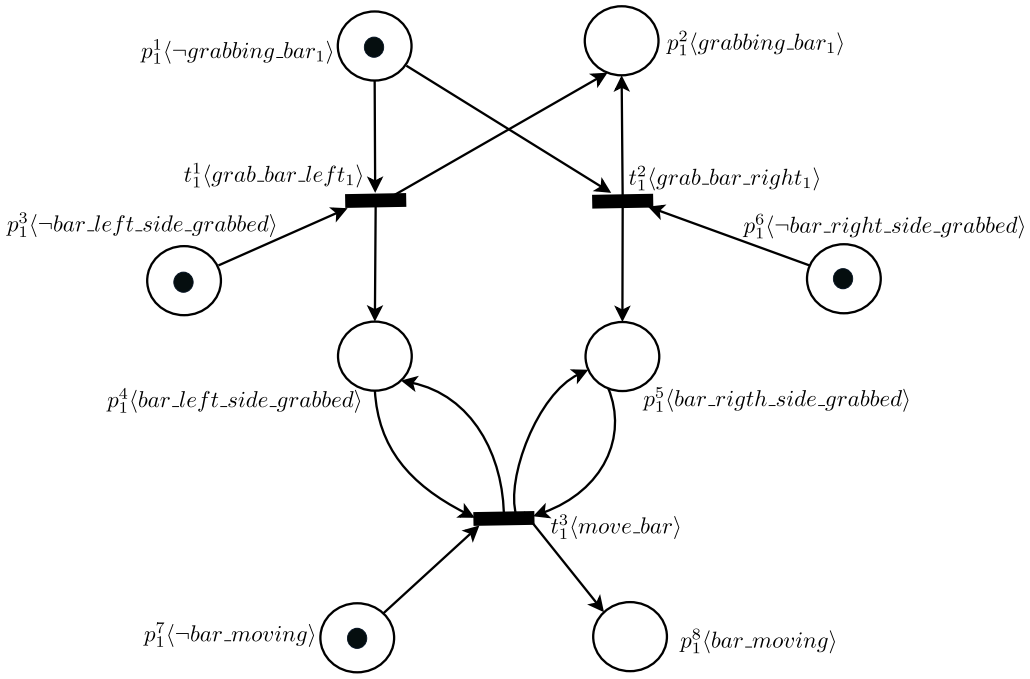


Figure 2.5: PN model for robot 1

the parallel composition definition for this specific case:

- $P_{D_1 \setminus D_2} = \{p_1^1, p_1^2\}$;
- $P_{D_2 \setminus D_1} = \{p_2^1, p_2^2\}$;
- $P_{D_1 \cap D_2} = \{(p_1^3, p_2^3), (p_1^4, p_2^4), (p_1^5, p_2^5), (p_1^6, p_2^6), (p_1^7, p_2^7), (p_1^8, p_2^8)\}$;
- $T_{E_1 \setminus E_2} = \{t_1^1, t_1^2\}$;
- $T_{E_2 \setminus E_1} = \{t_2^1, t_2^2\}$;
- $T_{E_1 \cap E_2} = \{(t_1^3, t_2^3)\}$.

In Figure 2.7, we depict the PN system model for the whole system, which is obtained by the parallel composition of the robot models. Note that the shared places and shared transitions were merged, but both the initial markings and the arc weights of G_1 and G_2 are maintained. Thus, the substructures of G_1 and G_2 are clearly represented in the output of the composition.

2.2.2 Algebraic State Description

When using the symbolic state representation, we do not take into direct account the fact that the PN state can be represented algebraically. This fact will enable us to enrich and shorten our specifications

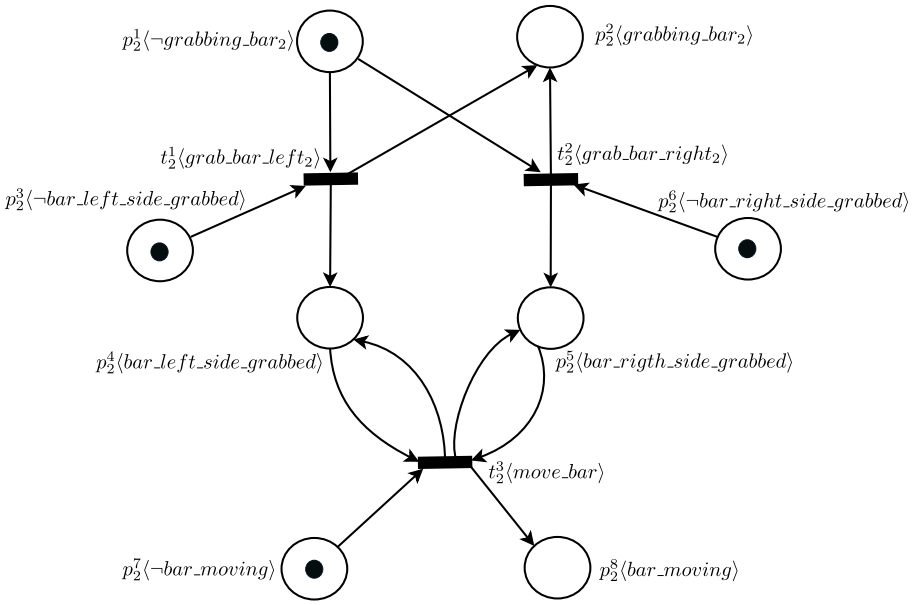


Figure 2.6: PN model for robot 2

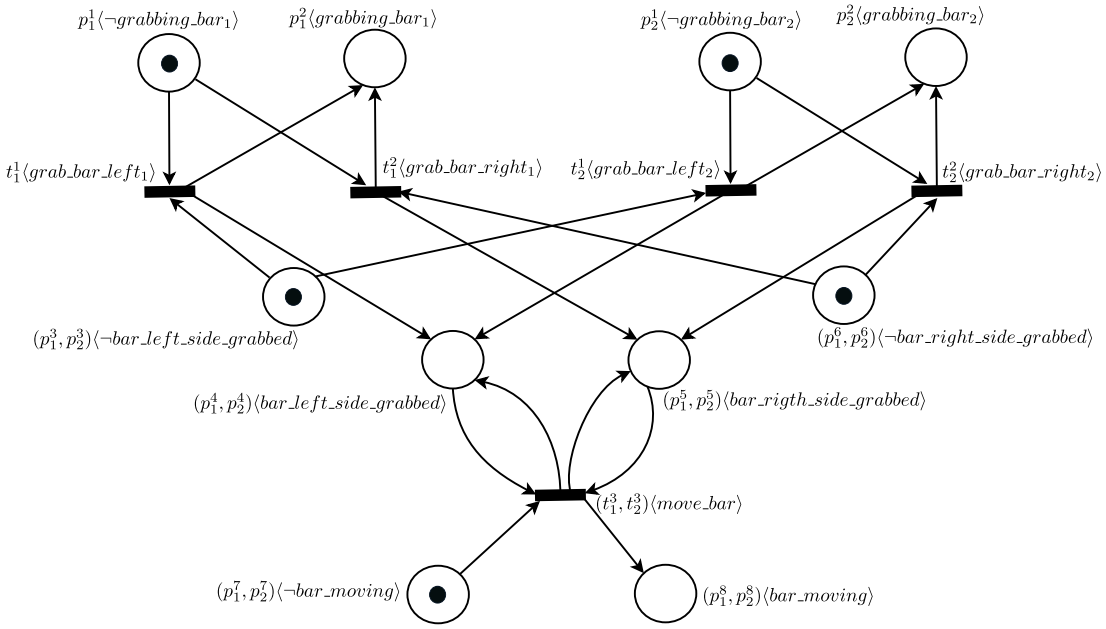


Figure 2.7: PN representing the composition of 2.5 and 2.6

as we will see when discussing the LTL semantics for this approach. To take advantage of the algebraic state description, we change the way we model the system, and use the notion of complement place of a bounded place.

Definition 2.2.19 (Place Bound)**.** Let $N = \langle P, T, W^-, W^+, M_0 \rangle$ be a PN structure and $p \in P$. A bound for p is a number $k \in \mathbb{N}$ such that $M(p) \leq k$ for all $M \in R(N)$. If there exists a bound for p we say that p is bounded, otherwise we say that p is unbounded. If there exists $M \in R(N)$ such that $M(p) = k$ we say that k is the strict bound.

Definition 2.2.20 (Complement Place)**.** Let $N = \langle P, T, W^-, W^+, M_0 \rangle$ be a PN structure and $p \in P$ a bounded place. A complement place for p is a place p' such that $M(p) + M(p') = k$ for all $M \in R(N)$, where k is a bound for p.

Definition 2.2.21 (Petri Net System Model – Algebraic State Description)**.** A Petri net system model is a tuple $G = \langle L, \kappa, ^- \rangle$ where:

- $L = \langle P, T, W^-, W^+, M_0, E, \ell \rangle$ is a labelled PN where $P = P_b \cup P_\omega$, with $P_b \cap P_\omega = \emptyset$;
- $\kappa : P_b \to \mathbb{N}$ is the maximum token function where, for all $p \in P_b$ and $M \in R(G)$:
  - $M(p) \leq \kappa(p)$.
- $^- : P_b \to P_b$ is the complement place function, where , for all $p \in P_b$ and $M \in R(G)$:
  - $\overline{\overline{p}} = p$;
  - $M(p) + M(\overline{p}) = \kappa(p) = \kappa(\overline{p})$.

The places $p \in P_b$ are places for which the designer knows a bound $\kappa(p)$. Note that this bound can be obtained by using analysis techniques and that it is not necessarily strict: $\kappa(p)$ can be strictly greater than $M(p)$ for all $M \in R(G)$. We also introduce a function that defines a complement relation between pairs of places in P_b. Places $p \in P_\omega$ are either (i) unbounded places or (ii) bounded places for which the designer does not know the bound.

Remark 2.2.1. Note that PN system models with symbolic state representation are subsumed in PN system models with algebraic state representation. To see this, consider the PN system model with symbolic state representation $G_{symb} = \langle L, D, \mu \rangle$. It can be modelled as the PN system model with algebraic state representation $G_{alg} = \langle L, \kappa, ^- \rangle$, where:

- $P_b = P_D$;
- $P_\omega = P_g$;
- $\kappa(p) = 1$ for all $p \in P_b$;
- For all $p, p' \in P_b$, $\overline{p} = p'$ if and only if exists $d \in D$ such that $\mu(d) = p$ and $\mu(\neg d) = p'$ or $\mu(d) = p'$ and $\mu(\neg d) = p$.

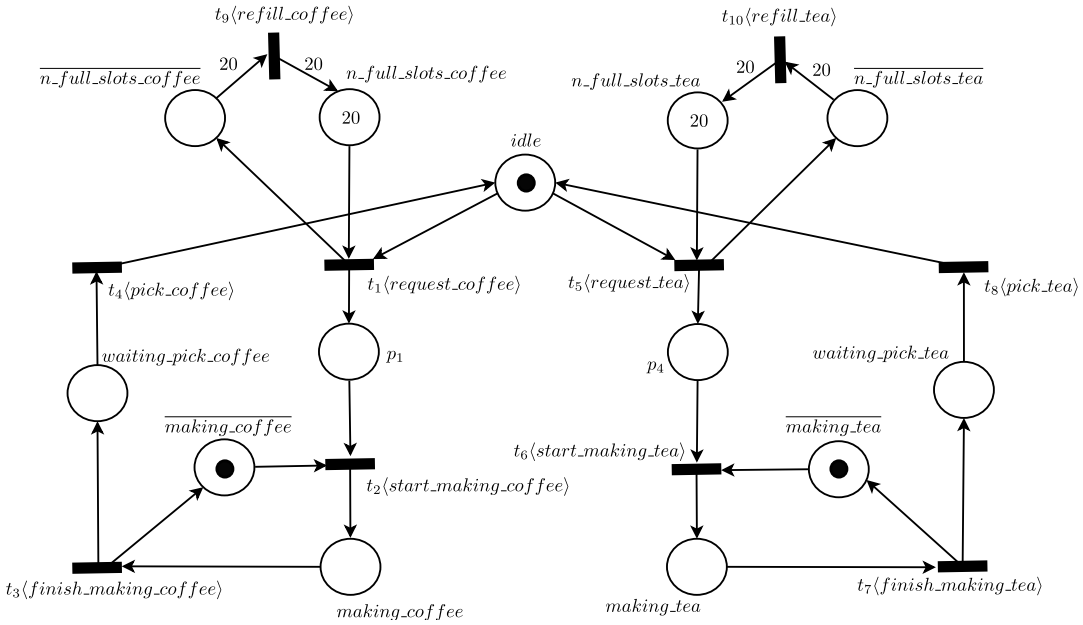


Figure 2.8: PN system model with algebraic state description for a tea/coffee machine

Example 2.2.3 (Tea/Coffee Machine)**.** In Figure 2.8 we depict the algebraic state description version of the PN with symbolic state description for a tea and coffee machine depicted in Figure 2.4. We also add a limit to the number of teas and coffees that can be served before the machine needs to be refilled. We assume that the machine can only hold 20 doses of coffee and 20 doses of tea.

The labels for the transitions are depicted as $t\langle\ell(t)\rangle$, as before. For the places, we directly depict the complement function when it is defined. Note that, for example, $\overline{\overline{making\_tea}} = making\_tea$. For the place bounds, we have the following:

- $\kappa(making\_tea) = \kappa(\overline{making\_tea}) = 1$;
- $\kappa(making\_coffee) = \kappa(\overline{making\_coffee}) = 1$;
- $\kappa(n\_full\_slots\_tea) = \kappa(\overline{n\_full\_slots\_tea}) = 20$;
- $\kappa(n\_full\_slots\_coffee) = \kappa(\overline{n\_full\_slots\_coffee}) = 20$;

Note that, for example $\kappa(waiting\_pick\_coffee) = 1$, but its complement place is not defined. However, we can add the complement place for all places for which we know a bound without changing the behaviour of the PN, as we will show next.

Definition 2.2.22 (Complement Place Addition)**.** Let $G = \langle P, T, M_0, W^-, W^+, E, \ell, \kappa, {}^- \rangle$ be a PN system model and $p \in P_b$ such that $\overline{p}$ is not defined. We add[6] the complement place of p, denoted p_c, to G, yielding the PN $G^c = \langle P \cup \{p_c\}, T, M_0^c, W^{c-}, W^{c+}, E, \ell, \kappa^c, {}^{-c} \rangle$, where, for $t \in T$ and for p_c:

[6]By "adding" a place, we mean that the structure of the PN remains the same for all $p \in P$, i.e., $M_0^c(p) = M_0(p)$, $W^{c-}(p,t) = W^-(p,t)$, $W^{c+}(p,t) = W^+(p,t)$, $\kappa^c(p) = \kappa(p)$ and $\overline{p}^c = \overline{p}$.

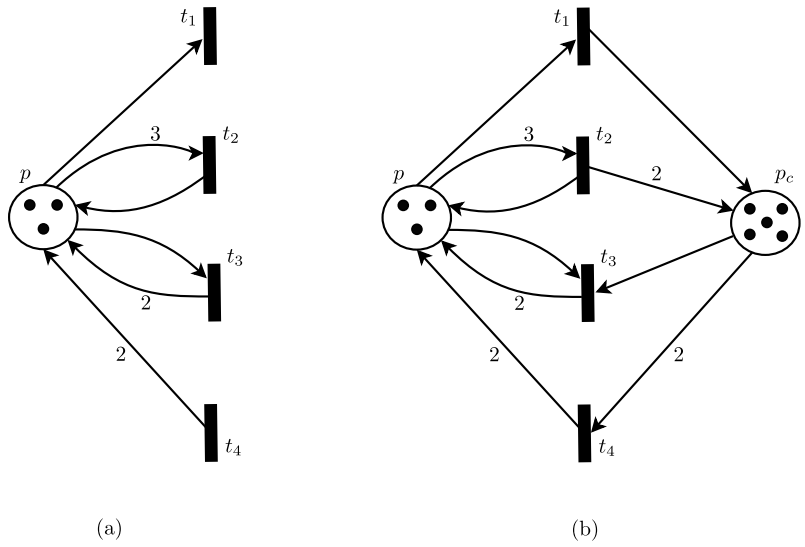


Figure 2.9: Illustration of the complement place construction.

- $M_0^c(p_c) = \kappa(p) - M_0(p)$;
- $W^{c-}(p_c,t) = \begin{cases} W^+(p,t) - W^-(p,t) & \text{if } W^+(p,t) - W^-(p,t) > 0 \\ 0 & \text{otherwise} \end{cases}$
- $W^{c+}(p_c,t) = \begin{cases} W^-(p,t) - W^+(p,t) & \text{if } W^-(p,t) - W^+(p,t) > 0 \\ 0 & \text{otherwise} \end{cases}$
- $\kappa^c(p_c) = \kappa(p)$;
- $\overline{p}^c = p_c$ and $\overline{p_c}^c = p$.

To build the complement place for a given $p \in P_b$, we just add a new place that starts with $\kappa - M_0(p)$ tokens and, for each firing, receives or loses the same amount of tokens that p loses or receives, respectively. Figure 2.9 (a) depicts a fragment of a PN where we assume that $\kappa(p) = 8$. Figure 2.9 (b) depicts the addition of the place p_c obtained from p by applying the complement place construction described above.

The PN G^c obtained by adding a complement place is well-defined, in the sense that the added place fulfils the requirements for complement place and the behaviour of G^c is unaltered, as stated in the following proposition.

Proposition 2.2.1. *Let G be a PN system model, $p \in P_b$ such that $\overline{p}$ is undefined, G^c the PN obtained by adding the complement place p_c for p and $\tau \in T^*$. Then, $M_0 \xrightarrow{\tau} M$ if and only if $M_0^c \xrightarrow{\tau} M^c$, where:*

$$M^c(p') = \begin{cases} M(p') & \text{if } p' \in P \\ \kappa(p) - M(p) & \text{if } p' = p_c \end{cases} \tag{2.32}$$

Proof. Note that it is clear that if $M_0^c \xrightarrow{\tau} M^c$, then $M_0 \xrightarrow{\tau} M$. This is true because, by adding a place, we can only restrict the behaviour of the original PN. Furthermore, the initial markings in G and G^c coincide for $p \neq p_c$, as do the pre and postsets. Thus, we just need to show that, given $\tau \in T^*$, if $M_0 \xrightarrow{\tau} M$, then $M_0^c \xrightarrow{\tau} M^c$. We will show that $M_0^c \xrightarrow{\tau} M^c$ by induction on the length of τ.

1. $|\tau| = 0$, i.e., $\tau = \varepsilon$:
   In this case we need to analyse the initial marking M_0^c, which satisfies the requirement by definition.

2. $|\tau| = n+1$, i.e., $\tau = \tau' t$, $\tau' \in T^*$, $t \in T$: By hypothesis, we have that:

$$M_{\tau'}^c(p') = \begin{cases} M_{\tau'}(p') & \text{if } p' \in P \\ \kappa(p) - M_{\tau'}(p) & \text{if } p' = p_c \end{cases} \tag{2.33}$$

   Hence, we need to prove that:

   (i) Given that t is active in $M_{\tau'}$ for G, then it is also active for $M_{\tau'}^c$ for G^c, i.e., given that $(M_{\tau'} \xrightarrow{t})$, then $(M_{\tau'}^c \xrightarrow{t})$:
   We assume that it is true that $(M_{\tau'} \xrightarrow{t})$, but not $(M_{\tau'}^c \xrightarrow{t})$. Hence, we have $W^{c-}(p,t) = W^-(p,t) \leq M_{\tau'}(p)$ for all $p \in P$. Thus, t not being active in $M_{\tau'}^c$ must be due to place p_c, i.e., $W^{c-}(p_c,t) > M_{\tau'}^c(p_c) = \kappa(p) - M_{\tau'}(p)$. According to the definition, we have two possible cases for the value of $W^{c-}(p_c,t)$. First if $W^{c-}(p_c,t) = 0$, then $M_{\tau'}(p) > \kappa(p)$, which contradicts the fact that $\kappa(p)$ represents the maximum amount of tokens that can be in places $p \in P$. Second, if $W^{c-}(p_c,t) = W^+(p,t) - W^-(p,t)$, then $M_{\tau'}(p) + W^+(p,t) - W^-(p,t) > \kappa(p)$, i.e., $M_{\tau' t}(p) > \kappa(p)$,which also contradicts the fact that $\kappa(p)$ represents the maximum amount of tokens that can be in places $p \in P$.

   (ii) The marking obtained by firing t in $M_{\tau'}^c$ satisfies the equality stated in the proposition, i.e.:

$$M_{\tau' t}^c(p') = \begin{cases} M_{\tau'}(p') - W^-(p',t) + W^+(p',t) & \text{if } p' \in P \\ \kappa(p) - (M_{\tau'}(p) - W^-(p,t) + W^+(p,t)) & \text{if } p = p_c \end{cases} \tag{2.34}$$

   For $p' \in P$, the equality is obvious because W^{c-} coincides with W^- and W^{c+} coincides with W^+. For $p' = p_c$, we have:

$$\begin{aligned} M_{\tau' t}^c(p_c) &= M_{\tau'}^c(p_c) - W^{c-}(p_c,t) + W^{c+}(p_c,t) \\ &= \kappa(p) - M_{\tau'}(p) - W^{c-}(p_c,t) + W^{c+}(t,p_c) \end{aligned}$$

   From the definition of W^{c-} and W^{c+}, we need to analyse two possible cases. First,

$W^{c-}(p_c,t) = 0$ and $W^{c+}(p_c,t) \geq 0$. In this case:

$$\begin{aligned} M^c_{\tau' t}(p_c) &= \kappa(p) - M_{\tau'}(p) + W^{c+}(p_c,t) \\ &= \kappa(p) - M_{\tau'}(p) + W^{-}(p,t) - W^{+}(p,t) \\ &= \kappa(p) - (M_{\tau'}(p) - W^{-}(p,t) + W^{+}(p,t)) \end{aligned}$$

Second $W^{c-}(p_c,t) \geq 0$ and $W^{c+}(p_c,t) = 0$. In this case:

$$\begin{aligned} M^c_{\tau' t}(p_c) &= \kappa(p) - M_{\tau'}(p) - W^{c-}(p_c,t) \\ &= \kappa(p) - M_{\tau'}(p) - (W^{+}(p,t) - W^{-}(p,t)) \\ &= \kappa(p) - (M_{\tau'}(p) - W^{-}(p,t) + W^{+}(p,t)) \end{aligned}$$

Hence the proof is completed.

□

Note that, given that PN system models with symbolical state descriptions are subsumed by PN system models with algebraic state descriptions, this entails that, if we know that the bound of a given place p is 1, we can always assign a new atomic proposition d to it, i.e., $\mu(d) = p$ and add a new place p' obtained by the complement place construction to the PN such that $\mu(\neg d) = p'$.

A direct consequence of this proposition is that G and G^c represent the same behaviour.

Corollary 2.2.1. *Let G be a PN system model. Then* $\mathscr{L}(G) = \mathscr{L}(G^c)$.[7]

From the proposition above, we can always assume that $^-$ is defined for all $p \in P_b$. In the cases where it is not, we add the complement place for p using the construction we just described. Hence, from now on we will assume that $^-$ is defined for all $p \in P_b$, and we will omit from the figures the complement places that are not required to specify a given behaviour, which are added using the complement place construction.

We finish by describing the parallel composition for PN models with algebraic state descriptions. In the following, let $G_i = \left\langle P_i = P_{b,i} \cup P_{\omega,i}, T_i, W_i^-, W_i^+, M_{0,i}, E_i, \ell_i, \kappa_i, {}^{-i} \right\rangle$, $i = 1,2$ be PN system models.

The set of shared transitions is defined as before (see definition 2.2.15), but in this case we do not need to define the set of shared places from the state description symbols.

Definition 2.2.23 (Non-Conflicting Petri Net Models)**.** G_1 and G_2 are said to be non-conflicting if they satisfy the following statements:

- $T_1 \cap T_2 = \emptyset$;

[7] There is a slight abuse of notation here, since G^c has more places that G. However, if we only consider the projection of the markings in $R(G^c)$ to the places that are in both PNs, the result holds.

- The initial markings assign the same truth value to places in $P_1 \cap P_2$:

$$\text{For all } p \in P_1 \cap P_2,\ \ M_{0,1}(p) = M_{0,2}(p) \tag{2.35}$$

- Transition pairs in $T_{E_1 \cap E_2}$ have coincident pre and post sets for places $p \in P_1 \cap P_2$:

$$\text{For all } (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p \in P_1 \cap P_2,\ W_1^-(p,t_1) = W_2^-(p,t_2) \text{ and } W_1^+(p,t_1) = W_2^+(p,t_2) \tag{2.36}$$

- Places in $P_1 \cap P_2$ have the same bound and the same complement place:

$$\text{For all } p \in P_1 \cap P_2,\ \kappa_1(p) = \kappa_2(p) \text{ and } \overline{p}^1 = \overline{p}^2 \tag{2.37}$$

Note that, in this case, we do not assume that $P_1 \cap P_2 = \emptyset$. This is because, in this case, we use P_1 and P_2 to identify shared places, instead of D_1 and D_2, which were used in the symbolic state representation case. As for the PN models with symbolical state descriptions, we need to assume that G_1 and G_2 are non-conflicting so that the parallel composition is well-defined. We note that the idea for this parallel composition is exactly the same as the one defined for PN system models with symbolic state description, the main reason to include it here is to keep both subsections self-contained and explicitly state the small adaptations needed.

Definition 2.2.24 (Parallel Composition)**.** The parallel composition of G_1 and G_2 is the PN model $G_1 \parallel G_2 = \langle P_1 \cup P_2, T_{1\|2}, W^-_{1\|2}, W^+_{1\|2}, M_{0,1\|2}, E_1 \cup E_2, \ell_{1\|2}, \kappa_{1\|2}, {}^{-1\|2} \rangle$, where:

- $T_{1\|2}$ is the union of the non-shared transitions of G_1 and G_2 with the pairs representing the shared transitions, i.e., $T_{1\|2} = T_{E_1 \setminus E_2} \cup T_{E_2 \setminus E_1} \cup T_{E_1 \cap E_2}$, where:

$$T_{E_1 \setminus E_2} = \{t \in T_1 \mid \ell_1(t) \in E_1 \setminus E_2\} \tag{2.38}$$

$$T_{E_2 \setminus E_1} = \{t \in T_2 \mid \ell_2(t) \in E_2 \setminus E_1\} \tag{2.39}$$

- $W^-_{1\|2} \in \mathbb{N}^{|P_{1\|2}| \times |T_{1\|2}|}$ maintains the same arc weights as W_1^- and W_2^-, taking into account that the shared places and shared transitions of G_1 and G_2 are now merged into a single place and a

single transition, respectively:

$$W^-_{1\|2}(p,t) = \begin{cases} W^-_1(p,t) & \text{if } t \in T_{E_1 \setminus E_2} \text{ and } p \in P_1 \setminus P_2 \\ W^-_2(p,t) & \text{if } t \in T_{E_2 \setminus E_1} \text{ and } p \in P_2 \setminus P_1 \\ W^-_1(p,t_1) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p \in P_1 \setminus P_2 \\ W^-_2(p,t_2) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p \in P_2 \setminus P_1 \\ W^-_1(p,t) & \text{if } t \in T_{E_1 \setminus E_2} \text{ and } p \in P_1 \cap P_2 \\ W^-_2(p,t) & \text{if } t \in T_{E_2 \setminus E_1} \text{ and } p \in P_1 \cap P_2 \\ W^-_1(p,t_1) = W^-_2(p,t_2) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p \in P_1 \cap P_2 \\ 0 & \text{otherwise} \end{cases} \tag{2.40}$$

Note that, for $t = (t_1,t_2) \in T_{E_1 \cap E_2}$ and $p \in P_1 \cap P_2$, we have the guarantee that $W^-_1(p_1,t_1) = W^-_2(p_2,t_2)$ because the PN models are assumed to be non-conflicting. Throughout the rest of the definition we will continue to explicitly state the equalities that stem from this assumption in order to clarify the reader where the non-conflicting assumption is required for a sound definition of parallel composition.

- $W^+_{1\|2} \in \mathbb{N}^{|P_{1\|2}| \times |T_{1\|2}|}$ is defined analogously to $W^-_{1\|2}$:

$$W^+_{1\|2}(p,t) = \begin{cases} W^+_1(p,t) & \text{if } t \in T_{E_1 \setminus E_2} \text{ and } p \in P_1 \setminus P_2 \\ W^+_2(p,t) & \text{if } t \in T_{E_2 \setminus E_1} \text{ and } p \in P_2 \setminus P_1 \\ W^+_1(p,t_1) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p \in P_1 \setminus P_2 \\ W^+_2(p,t_2) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p \in P_2 \setminus P_1 \\ W^+_1(p,t) & \text{if } t \in T_{E_1 \setminus E_2} \text{ and } p \in P_1 \cap P_2 \\ W^+_2(p,t) & \text{if } t \in T_{E_2 \setminus E_1} \text{ and } p \in P_1 \cap P_2 \\ W^+_1(p,t_1) = W^+_2(p,t_2) & \text{if } t = (t_1,t_2) \in T_{E_1 \cap E_2} \text{ and } p \in P_1 \cap P_2 \\ 0 & \text{otherwise} \end{cases} \tag{2.41}$$

- $M_{0,1\|2} : P_{1\|2} \to \mathbb{N}$ maintains the respective initial markings of G_1 and G_2, taking into account the merging of shared places:

$$M_{0,1\|2}(p) = \begin{cases} M_{0,1}(p) & \text{if } p \in P_1 \\ M_{0,2}(p) & \text{if } p \in P_2 \\ M_{0,1}(p) = M_{0,2}(p) & \text{if } p \in P_1 \cap P_2 \end{cases} \tag{2.42}$$

- $\ell_{1\|2} : T_{1\|2} \to E_1 \cup E_2$ maintains the respective transition labels of G_1 and G_2, taking into account the merging of shared transitions:

$$\ell_{1\|2}(t) = \begin{cases} \ell_1(t) & \text{if } t \in T_{E_1 \setminus E_2} \\ \ell_2(t) & \text{if } t \in T_{E_2 \setminus E_1} \\ \ell_1(t_1) = \ell_2(t_2) & \text{if } t = (t_1,t_2) \in T_{T_1 \cap T_2} \end{cases} \tag{2.43}$$

- $\kappa_{1\|2} : P_1 \cup P_2 \to \mathbb{N}$ maintains the bounds for all places:

$$\kappa_{1\|2}(p) = \begin{cases} \kappa_1(p) & \text{if } p \in P_1 \setminus P_2 \\ \kappa_2(p) & \text{if } p \in P_2 \setminus P_1 \\ \kappa_1(p) = \kappa_2(p) & \text{if } p \in P_1 \cap P_2 \end{cases} \tag{2.44}$$

- $^{-1\|2} : P_{b,1} \cup P_{b,2} \to P_{b,1} \cup P_{b,2}$ maintains the complement places for all places:

$$\overline{p}^{1\|2} = \begin{cases} \overline{p}^1 & \text{if } p \in P_{b,1} \setminus P_{b,2} \\ \overline{p}^2 & \text{if } p \in P_{b,2} \setminus P_{b,1} \\ \overline{p}^1 = \overline{p}^2 & \text{if } p \in P_{b,1} \cap P_{b,2} \end{cases} \tag{2.45}$$

The parallel composition simply models the simultaneous execution of G_1 and G_2, where common events are synchronized, i.e., a common event can only be executed if both G_1 and G_2 can execute it simultaneously. We also merge the places that are equal.

Example 2.2.4. Consider Figure 2.10, which depicts the model G_i of the connection and team choice of player i, $i \in \{1,2,3,4\}$ to an online 2 versus 2 card playing game. The player starts by connecting to the game server, and, once it is connected it chooses whether he wants to be part of team 1 or team 2. Transitions t_i^5 correspond to starting the game and are all labelled by the same event in all models, hence they are shared and need to be synchronized. Furthermore, places n_team1, $\overline{n\_team1}$, n_team2, $\overline{n\_team2}$ and $game\_started$ are also shared by all the player models, so they are merged. These places represent global variables of the system. Thus, the sets used in the parallel definition composition of, for example, G_1 and G_2, are the following:

- $P_1 \setminus P_2 = \{idle_1, connecting\_to\_game_1, in\_game_1\}$;
- $P_2 \setminus P_1 = \{idle_2, connecting\_to\_game_2, in\_game_2\}$;
- $P1 \cap P_2 = \{n\_team1, \overline{n\_team1}, n\_team2, \overline{n\_team2}, game\_started\}$;
- $T_{E_1 \setminus E_2} = \{t_1^1, t_1^2, t_1^3, t_1^4\}$;
- $T_{E_2 \setminus E_1} = \{t_2^1, t_2^2, t_2^3, t_2^4\}$;
- $T_{E_1 \cap E_2} = \{(t_1^5, t_2^5)\}$.

Figure 2.11 depicts the model $G = G_1 \parallel G_2 \parallel G_3 \parallel G_4$ of the whole player connection and team choices, obtained by the parallel composition of the 4 player models. Note that in the team model, only 2 players can choose to be in a given team, given that we merged the places corresponding to the shared resources n_team1 and n_team2. This example also illustrates the scalability of the parallel composition for PNs when compared to the parallel composition for FSA: The substructure of each of the four subsystems is clearly identifiable in the result of their parallel composition, given that only

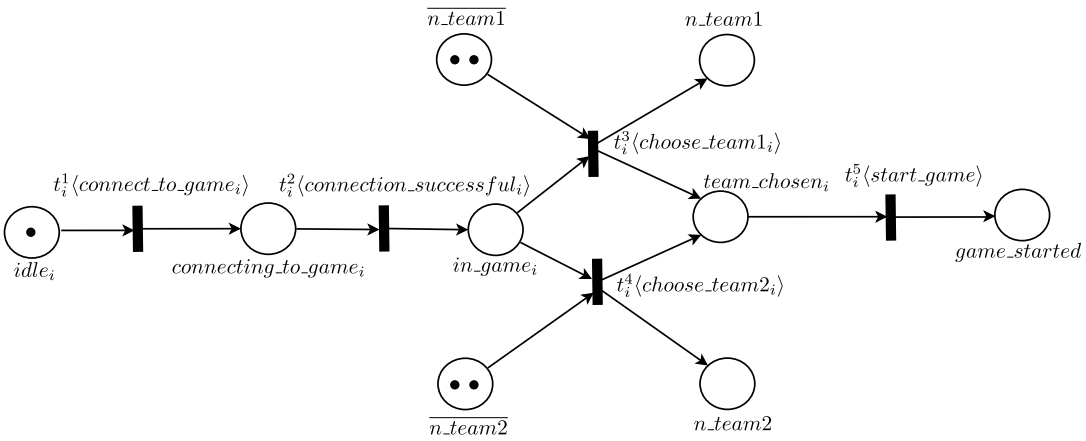


Figure 2.10: Petri net model of a card player

some places and transitions were merged, while in the case of FSA parallel composition, the states of the composition would represent the Cartesian product of states in each of the subsystems, thus growing much more rapidly.

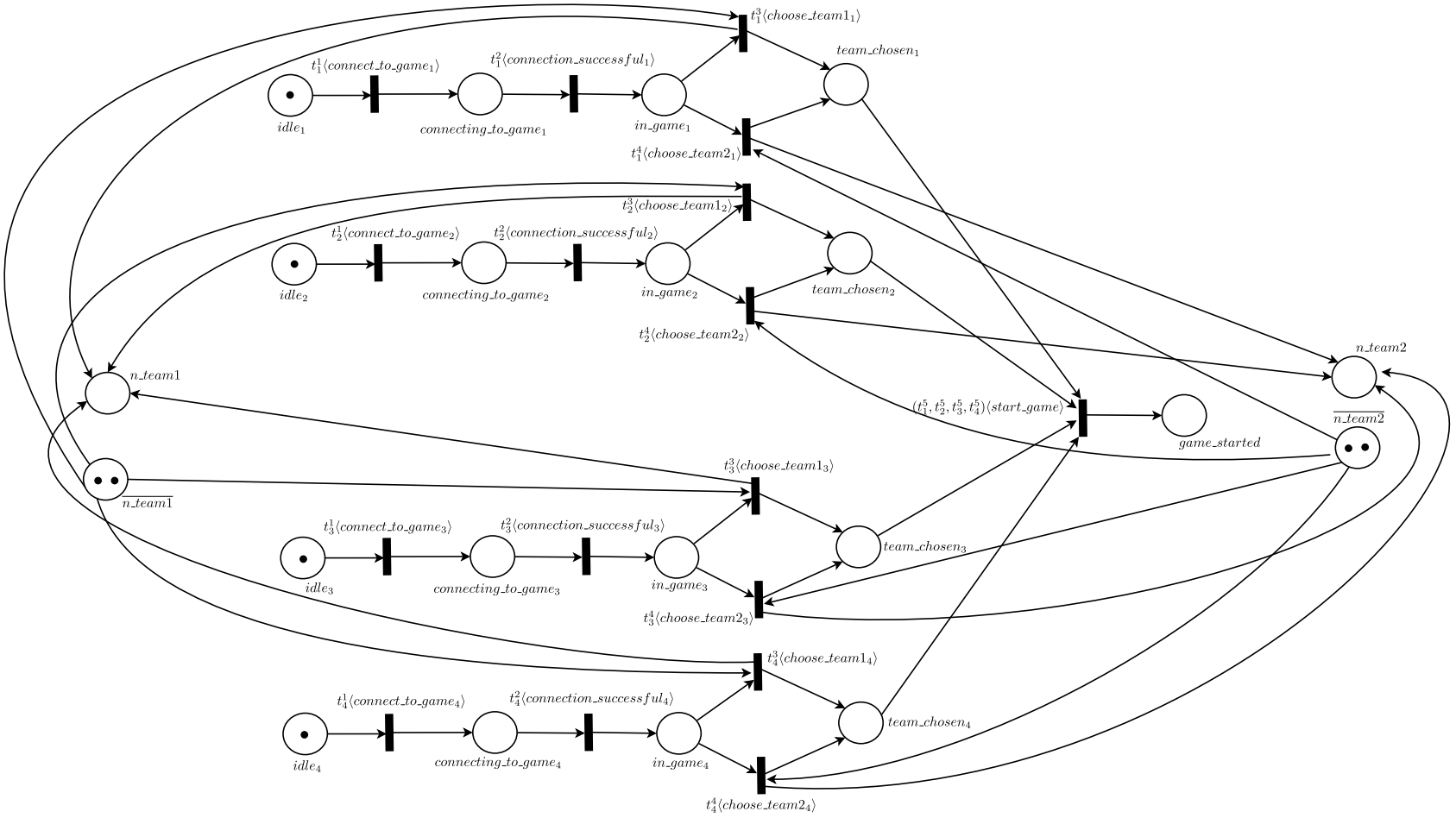


Figure 2.11: Petri net model of the complete card playing game

CHAPTER 3

Specification Language

In this chapter, we present the concepts of formal logic and formal languages that will be used throughout the work. We start by providing the basics of linear temporal logic (LTL), its semantics over the basic models defined in the previous chapter, and its translation to Büchi automata (BA). Afterwards, we will discuss the notion of safety languages and restrict the specification language to safe LTL formulas, providing a discussion on the expressiveness of this type of specification. In the following, let Π be a countable set of atomic propositions and $(2^{\Pi})^{\omega}$ the set of all infinite sequences of subsets of Π.

3.1 Linear Temporal Logic

In this section we will briefly define linear temporal logic (LTL) and describe its semantics over the system models we presented. Furthermore, we will describe the translation of LTL to Büchi automata, which is a crucial step in our methodology, given that the fact that we can have an automata-theoretic view of LTL formulas is what allows us to then proceed with the restriction of the system models so that they satisfy a formula. LTL is an extension of propositional logic which allows reasoning over an infinite sequence of states. It was developed as a formal verification tool for concurrent systems, as it provides a close-to-natural-language way to specify and verify qualitative properties of a system.

3.1.1 Syntax and Semantics over the System Models

Definition 3.1.1 (Syntax)**.** An LTL formula over a set Π of atomic propositions has the following syntax:

- $true$, $false$ and $\pi \in \Pi$ are LTL formulas;

- If φ and ψ are LTL formulas then $(\neg\varphi)$, $(\varphi \vee \psi)$, $(X\varphi)$, $(\varphi U \psi)$ and $(\varphi R \psi)$ are also LTL formulas.

LTL formulas are evaluated over infinite sequences of sets of propositional symbols $\sigma = \sigma_0\sigma_1\sigma_2... \in (2^{\Pi})^{\omega}$, i.e., $\sigma_i \subseteq 2^{\Pi}$ for $i \in \mathbb{N}$. The logical operators will have the same semantics for all the models, but we will have different ways of evaluating atomic propositions, depending on the model we are using. Thus, we start by giving the usual definition of satisfaction for LTL and a generic set of atomic propositions Π, and then discuss how the satisfaction of atomic propositions is specialized to the system models.

Definition 3.1.2 (Satisfaction)**.** Let $\sigma \in (2^{\Pi})^{\omega}$, $t \in \mathbb{N}$. The notion of local satisfaction, $\Vdash$, is defined as follows:

- $\sigma_t \Vdash true$ and $\sigma_t \nVdash false$;
- $\sigma_t \Vdash \pi$ if and only if $\pi \in \sigma_t$;
- $\sigma_t \Vdash (\neg\varphi)$ if and only if $\sigma_t \nVdash \varphi$;
- $\sigma_t \Vdash (\varphi \vee \psi)$ if and only if $\sigma_t \Vdash \varphi$ or $\sigma_t \Vdash \psi$;
- $\sigma_t \Vdash (\varphi \wedge \psi)$ if and only if $\sigma_t \Vdash \varphi$ and $\sigma_t \Vdash \psi$;
- $\sigma_t \Vdash (X\varphi)$ if and only if $\sigma_{t+1} \Vdash \varphi$;
- $\sigma_t \Vdash (\varphi U \psi)$ if and only if there exists $t' \geq t$ such that $\sigma_{t'} \Vdash \psi$ and for all $t'' \in [t, t'[$ $\sigma_{t''} \Vdash \varphi$;
- $\sigma_t \Vdash (\varphi R \psi)$ if and only if for all $t' \geq t$, if $\sigma_{t'} \Vdash \psi$ then there exists $t'' \in [t, t'[$ such that $\sigma_{t''} \Vdash \varphi$;

If $\sigma_0 \Vdash \varphi$ we say that σ (globally) satisfies φ, which is simply written $\sigma \Vdash \varphi$.

The propositional fragment of LTL, i.e., formulas written without the temporal operators X, U and R, is evaluated only over one element $\sigma_t \in 2^{\Pi}$ of $\sigma \in (2^{\Pi})^{\omega}$. Hence, when evaluating propositional formulas φ, we will sometimes use the notation $S \Vdash \varphi$, where $S \in 2^{\Pi}$.

The temporal operators are understood as follows:

- The X operator is read "next", meaning that the formula it precedes will be true in the next state.
- The U operator is read "until", meaning that its first argument will be true until its second argument becomes true (and the second argument must become true in some state, i.e., a sequence where φ is always satisfied but ψ is never satisfied does not satisfy $\varphi U \psi$).
- The R operator is read "release", meaning that its second argument must remain true until the first time its first argument becomes true (i.e., the occurrence of the first argument releases the need for the second argument to be true). For this operator, if the first argument never becomes true, the second argument must remain true for all times.

We will now instantiate the set of propositional symbols to the system model being used.

Definition 3.1.3 (Atomic Propositions – System Models)**.** Let G be a system model. We set of atomic propositions will be defined as:

- $\Pi_{symb} = D \cup E$ if $G = \langle Q, E, \delta, q_0, D, \mu \rangle$ is an FSA system model or $G = \langle P, T, W, M_0, E, \ell, D, \mu \rangle$ is a PN system model with symbolic state description.
- $\Pi_{alg} = (P_b \times \mathbb{N}) \cup E$ if $G = \langle P_b \cup P_\omega, T, W, M_0, E, \ell, \kappa, ^- \rangle$ is a PN system model with algebraic state description.

Intuitively, an element (p,b) of Π_{alg} means that the number of tokens in place p should be greater or equal than b. Thus, to facilitate the comprehension, when presenting formulas we will write the elements (p,b) of Π_{alg} as $M(p) \geq b$, $b \in \mathbb{N}$.

Given that the languages generated by the system models are infinite sequences of event/state pairs, and our goal is to restrict these languages using LTL, we need to define what subset of atomic propositions these pairs represent, so that we can evaluate LTL formulas over the languages generated by the systems.

Definition 3.1.4 (Event/State Semantics – FSA)**.** Let $G = \langle Q, E, \delta, q_0, D, \mu \rangle$ be an FSA system model, $q \in Q$, $e \in E$. The pair (q,e) generates the set $\{e\} \cup \mu(q) \in 2^{\Pi_{symb}}$.

Definition 3.1.5 (Event/Marking Semantics – PN with Symbolic State Representation)**.** Let $G = \langle P, T, W, M_0, E, \ell, D, \mu \rangle$ be an PN system model with symbolic state representation, $M \in R(G)$, $e \in E$. The pair (M,e) generates the set $\{e\} \cup D_M \in 2^{\Pi_{symb}}$.

Definition 3.1.6 (Event/Marking Semantics – PN with Algebraic State Representation)**.** Let $G = \langle P, T, W, M_0, E, \ell, \kappa, ^- \rangle$ be an PN system model with algebraic state representation, $M \in R(G)$, $e \in E$. The pair (M,e) generates the set $\{e\} \cup \{(p,b) \in P_b \times \mathbb{N} \mid M(p) \geq b\} \in 2^{\Pi_{alg}}$.

Note that, to check if a given pair (p,b) is in the set generated by (M,e), we just need to check if $M(p) \geq b$ is a true statement. We will slightly abuse the notation and, given an element $\sigma \in \mathscr{L}(G)$, i.e., an infinite sequence of event/state pairs, we will write $\sigma \Vdash \varphi$ if the infinite sequence composed of the sets generated by the event/state pairs in σ satisfies φ, according to Definition 3.1.2.

For PN system models with algebraic state representation, we only consider linear constraints of the form $M(p) \geq b$ with $p \in P$ and $b \in \mathbb{N}$ in this semantics. In spite of that, this already gives us a good amount of expressibility, since any constraint consisting on linear combinations of the number of tokens in the bounded places of a PN can be re-written in this form, as we will show later.

The more attentive reader might have noticed that we are using the $\wedge$ and R operators, which are not fundamental operators, in the sense that any LTL formula can be written using only the other operators. This of because if the identities $(\varphi \wedge \psi) = (\neg(\neg\varphi \vee \neg\psi))$ and $(\varphi R \psi) = (\neg(\neg\varphi U \neg\psi))$. However, $\wedge$ and R are required to write formulas in the so called *positive normal form* (PNF), where only atomic propositions can be negated. As will be discussed later, we will write all our formulas in the PNF, so we can syntactically ensure that we are writing safety formulas.

Definition 3.1.7 (Positive Normal Form)**.** An LTL formula over Π in the positive normal form (PNF) has the following syntax:

- $true$, $false$, π and $\neg\pi$, with $\pi \in \Pi$ are LTL formulas in the PNF;
- If φ and ψ are LTL formulas in the PNF then $(\varphi \vee \psi)$, $(X\varphi)$, $(\varphi U \psi)$ and $\varphi R \psi)$ are also LTL formulas in the PNF.

Using the following identities, one can convert any LTL formula to the PNF:

- $(\neg true) = false$ and $(\neg false) = true$;
- $(\neg(\neg\varphi)) = \varphi$;
- $(\neg(\varphi \wedge \psi)) = (\neg\varphi \vee \neg\psi)$ and $(\neg(\varphi \vee \psi)) = (\neg\varphi \wedge \neg\psi)$;
- $(\neg X\varphi) = (X\neg\varphi)$;
- $(\neg(\varphi U \psi)) = (\neg\varphi R \neg\psi)$ and $(\neg(\varphi R \psi)) = (\neg\varphi U \neg\psi)$.

Hence, we do not lose expressiveness by using the PNF.

Example 3.1.1. We provide an example of conversion of an LTL formula to the PNF:

$$\begin{aligned}\neg(((X\pi_1)U(\neg(\pi_2 R \neg\pi_3))) \wedge (true U \neg\pi_2)) &= \neg((X\pi_1)U(\neg(\pi_2 R \neg\pi_3))) \vee \neg(true U \neg\pi_2) \\ &= (\neg X\pi_1)R(\neg\neg(\pi_2 R \neg\pi_3)) \vee false R \pi_2 \\ &= (X\neg\pi_1)R(\pi_2 R \neg\pi_3) \vee false R \pi_2\end{aligned}$$

In addition to the usual propositional logic abbreviations, there are other temporal operators usually also defined by abbreviation.

Definition 3.1.8 (Abbreviations)**.** We define the following abbreviations:

- $(\varphi \Rightarrow \psi) \equiv_{abv} ((\neg\varphi) \vee \psi)$;
- $(\varphi \Leftrightarrow \psi) \equiv_{abv} ((\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi))$;
- $(F\varphi) \equiv_{abv} (true U \varphi)$;
- $(G\varphi) \equiv_{abv} (false R \varphi) = (\neg F(\neg\varphi))$;
- $(\varphi W \psi) \equiv_{abv} ((\varphi U \psi) \vee (G\varphi)) = (\psi R(\psi \vee \varphi))$.

The temporal abbreviations are understood as follows:

- The F operator is read "eventually" and requires the existence of a future state where the formula it precedes is true.

- The G operator is read "always" and requires the formula it precedes to be true in all future states.
- The W operator is read "weak until" and relaxes the U operator by allowing its second argument to never be satisfied.

Example 3.1.2 (Writing LTL formulas)**.** The natural language specification "Whenever the robot starts moving, it should not stop until it reaches the goal region" can be written as an LTL formula over the set $\Pi_{symb} = \{start\_moving,\ stop\_moving,\ goal\_reached\}$:

$$G(start\_moving \Rightarrow (X(\neg stop\_movingU goal\_reached))) \tag{3.1}$$

The natural language specification "In all markings, if the number of tokens in place p_1 is greater or equal than 2 then, in the next marking, either the marking in place p_2 is greater or equal than 4 or event $put\_token\_in\_p_2$ just occurred:

$$G((p_1,2) \Rightarrow (X((p_2,4) \vee put\_token\_in\_p_2))) \tag{3.2}$$

As we already mentioned, when writing formulas, we will replace atomic propositions (p,b) by $M(p) \geq b$ in order to improve readability,. Hence, the specification can be re-written as:

$$G(M(p_1) \geq 2 \Rightarrow (X(M(p_2) \geq 4 \vee put\_token\_in\_p_2))) \tag{3.3}$$

3.1.2 Translation to Büchi automata

We now present Büchi automata (BA), which will be used as a representation of an LTL formula. A BA is a finite state automaton equipped with the so called Büchi acceptance condition. We start by noting that LTL formulas can be used to generate ω-languages.

Definition 3.1.9 (Language generated by an LTL formula)**.** Let φ be an LTL formula. The ω-language generated by φ is given by:

$$\mathscr{L}(\varphi) = \{\sigma \in (2^{\Pi})^{\omega} \mid \sigma \Vdash \varphi\} \tag{3.4}$$

We designate the class of languages generated by LTL formulas as LTL languages.

Example 3.1.3. We provide examples on the languages generated by LTL formulas:

- $\mathscr{L}(F\pi_1) = \{\sigma = \sigma_0\sigma_1... \in (2^{\Pi})^{\omega} \mid \text{exists } t \in \mathbb{N} \text{ such that } \pi_1 \in \sigma_t\}$. For example, any ω-string with the prefix $\{\pi_0\}\{\pi_0,\pi_2,\pi_3\}\emptyset\{\pi_0,\pi_1\}\{\pi_2\}$ is part of $\mathscr{L}(F\pi_1)$, because $\pi_1 \in \sigma_3$.
- $\mathscr{L}(G\pi_1) = \{\sigma = \sigma_0\sigma_1... \in (2^{\Pi})^{\omega} \mid \text{for all } t \in \mathbb{N},\ \ \pi_1 \in \sigma_t\}$. For example, the ω-string $\{\pi_1\}\{\pi_0,\pi_1\}\{\pi_1,\pi_2\}\{\pi_1\}\{\pi_1\}\{\pi_1\}...$ is part of $\mathscr{L}(G\pi_1)$.
- $\mathscr{L}(FG\pi_1) = \{\sigma = \sigma_0\sigma_1... \in (2^{\Pi})^{\omega} \mid \text{exists } t \in \mathbb{N} \text{ such that for all } t' \geq t,\ \pi_1 \in \sigma_{t'}\}$. For example, the ω-string $\{\pi_0\}\{\pi_3\}\{\pi_0,\pi_1\}\{\pi_0\}\emptyset\{\pi_1,\pi_2\}\{\pi_1,\pi_2\}\{\pi_1,\pi_2\}...$ is part of $\mathscr{L}(FG\pi_1)$.

Hence, we are interested in finding another way to represent the language generated by an LTL formula, which is more related to our system models. The structure we will use to represent LTL languages will be Büchi automata, for which translation algorithms from LTL formulas have been widely studied.

Definition 3.1.10 (Büchi Automaton)**.** A Büchi automaton (BA) is a tuple $B = \langle Q, \Sigma, \delta, Q_0, Q_F \rangle$ where:

- $\langle Q, \Sigma, \delta, Q_0 \rangle$ is a (non-deterministic) FSA.
- $Q_F \subseteq Q$ is the set of accepting states.

In order to define the accepted ω-language by a BA, we introduce valid state labellings.

Definition 3.1.11 (Valid State Labelling)**.** Let $B = \langle Q, \Sigma, \delta, Q_0, Q_F \rangle$ and $\sigma \in \Sigma^\omega$. A valid state labelling for B and σ is an infinite sequence $\rho \in Q^\omega$ such that:

$$\rho_0 \in Q_0 \tag{3.5}$$

$$\rho_{i+1} \in \delta(\rho_i, \sigma_i) \text{ for all } i \in \mathbb{N} \tag{3.6}$$

We denote $P(B, \sigma)$ as the set of all possible valid state labellings for B and σ.

A valid state labelling for B and σ is an infinite sequence of states that are visited in a "run" of B while following σ. $P(B, \sigma)$ can be empty if σ does not respect the transition function δ or can have one or more elements due to the non-determinism of the automaton.

Definition 3.1.12 (Language Accepted by a Büchi Automaton)**.** The ω-language accepted by a BA $B = \langle Q, \Sigma, \delta, Q_0, Q_F \rangle$ is defined as follows:

$$\mathscr{L}(B) = \{\sigma \in \Sigma^\omega \mid \text{exists } \rho \in P(B, \sigma) \text{ such that } inf(\rho) \cap Q_F \neq \emptyset\} \tag{3.7}$$

For $\rho \in Q^\omega$, $inf(\rho)$ is the set of all $q \in Q$ that appear infinite times in ρ. The class of languages accepted by Büchi automata is the class of ω-regular languages.

The accepted ω-strings by B are the ones that correspond to at least one run of B that visits at least one of the accepting states infinite times. The following proposition states the relation between BA and LTL formulas.

Proposition 3.1.1 ([Wolper, 2001])**.** *Let φ be an LTL formula written over Π. Then there exists a (non-deterministic) BA $B_\varphi = \langle Q, 2^\Pi, \delta, q_0, Q_F \rangle$ such that $\mathscr{L}(B_\varphi) = \mathscr{L}(\varphi)$.*

There are several methods for the construction of this automaton. In the implementation of the method we present here, we use one of the most efficient translation algorithms, LTL2BA, described in [Gastin and Oddoux, 2001]. In the BA obtained by applying the LTL2BA algorithm, propositional

formulas in the disjunctive normal form (DNF) are used to describe the transition labels in a more compact way. A formula is in the DNF if it is of the form:

$$\bigvee_{i=1}^{n} \bigwedge_{j=i}^{m_i} l_{ij}, \text{ where } l_{ij} \in lit(\Pi) \tag{3.8}$$

The elements $\bigwedge_{j=i}^{m_i} l_{ij}$ are called conjunctive clauses. In words, a conjunctive clause is the conjunction of a set of literals and a DNF formula is the disjunction of a set of conjunctive clauses. For example, if $\psi_1 = (\pi_1 \wedge \neg\pi_2) \vee (\pi_3)$ is a transition label, then all elements of 2^{Π} that contain π_1 and do not contain π_2, or that contain π_3 are labels of that transition, i.e., it represents the set of transition labels $\{S \in 2^{\Pi} \mid \pi_1 \in S \text{ and } \pi_2 \notin S\} \cup \{S \in 2^{\Pi} \mid \pi_3 \in S\}$. The label *true* is also used, to represent the set of all possible transition labels, i.e., to represent that any subset of Π can occur. Given an element $S \in 2^{\Pi}$ and a DNF formula ψ written over Π, the easiest way to check if S is one of the transition labels coded by ψ is to check if $S \Vdash \psi$, using the notions of satisfaction for the propositional connectives $\neg$, $\wedge$ and $\vee$ and for the atomic propositions we presented. Note that this is the propositional fragment of LTL, hence we only need one element of an infinite string to evaluate them.

Example 3.1.4 (Evaluating DNF Labels)**.** Given $S = \{\pi_1, \pi_3\}$, $\psi_1 = (\pi_1 \wedge \neg\pi_2) \vee (\pi_3)$ and $\psi_2 = (\pi_1 \wedge \pi_2) \vee (\neg\pi_2 \wedge \neg\pi_3)$, we have the following:

- $S \Vdash \psi_1$, because both $\pi_1 \in S$ and $\pi_2 \notin S$, and $\pi_3 \in S$. Hence, both conjunctive clauses are satisfied, thus so is the DNF formula. Note that satisfying one of the conjunctive clauses would suffice.
- $S \nVdash \psi_2$, because, for the first conjunctive clause, $\pi_2 \notin S$ and, for the second conjunctive clause $\pi_3 \in S$. Hence, both conjunctive clauses are not satisfied, and neither is the DNF formula.

Given that the labels are written in the DNF and a DNF formula is satisfied if and only if at least one of its conjunctive clauses are satisfied, it is easy to check if a given subset of 2^{Π} satisfies the formula, by checking each conjunctive clause until we find one that is satisfied, or we conclude that none of the conjunctive clauses in the DNF formula are satisfied. Furthermore, any propositional formula can be rewritten in the DNF and, given an arbitrary propositional formula ψ, we will write $DNF(\psi)$ to denote the DNF representation of ψ.

Example 3.1.5 (A Büchi Automaton)**.** Figure 3.1 depicts the BA obtained from formula

$$G(start\_moving \Rightarrow (X(\neg stop\_movingU goal\_reached))) \tag{3.9}$$

The label *true* means that any symbol from the alphabet 2^{Π} can drive the automaton from state 1 to state 2. Hence, any symbol (i.e., any subset of $\{start\_moving, stop\_moving, goal\_reached\}$ that does not contain *start_moving* will correspond to a non-deterministic transition, where the automaton can go to both state 1 or 2. State 1 is the initial state, and states 1 and 3 are accepting states, hence any run which visits 1 or 3 infinite times is accepted by the BA.

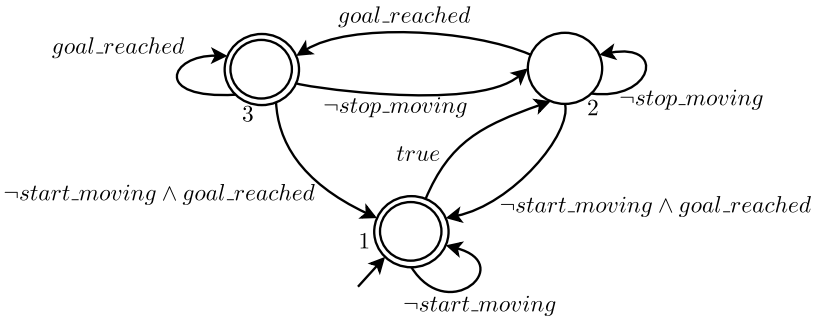


Figure 3.1: BA obtained from $G(start\_moving \Rightarrow (X(\neg stop\_movingU goal\_reached)))$

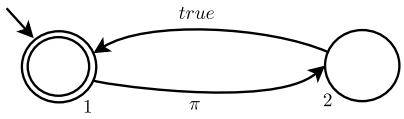


Figure 3.2: BA accepting $L = \{\sigma \in (2^{\Pi})^{\omega} \mid \pi \in \sigma_{2i}, i \in \mathbb{N}\}$

We also note that the inclusion of LTL languages in ω-regular languages is strict. For example the language $L = \{\sigma \in (2^{\Pi})^{\omega} \mid \pi \in \sigma_{2i}, i \in \mathbb{N}\}$, i.e., the language of infinite strings where π occurs in every even index, can be represented by the BA in Figure 3.2. However it is not an LTL language [Wolper, 1983].

We now introduce two operators over BA that will be useful to clarify our discussion.

Definition 3.1.13 (Accessible and Co-Accessible States)**.** Let $B = \langle Q, \Sigma, \delta, Q_0, Q_F \rangle$ be a BA. We define the sets of:

- *Accessible states* as $Q_{ac} \subseteq Q$ such that there is a path for a state $q_0 \in Q_0$ to a state $q_{ac} \in Q_{ac}$.
- *Co-accessible states* as $Q_{co-ac} \subseteq Q$ such that there is a path for a state $q_{co-ac} \in Q_{ac}$ to a state $q_f \in Q_F$.

Definition 3.1.14 (Trim Büchi Automaton)**.** Let B be a BA. We define $trim(B)$ as the automaton obtained by deleting from B all states that are not accessible or not co-accessible.

It is clear that $\mathscr{L}(B) = \mathscr{L}(trim(B))$. We will assume that all our BA are trim.

Definition 3.1.15 (Closure of a Büchi Automaton)**.** Let $B = \langle Q, \Sigma, \delta, Q_0, Q_F \rangle$ be a BA. We define $cl(B) = \langle Q, \Sigma, \delta, Q_0, Q \rangle$.

We obtain the closure of a BA by simply defining all its states as accepting states. Note that the language accepted by the closure of a BA $B = \langle Q, \Sigma, \delta, Q_0, Q_F \rangle$ is equal to the language generated

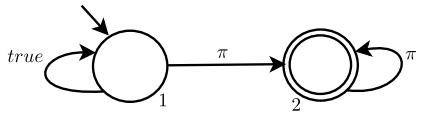


Figure 3.3: Non-deterministic BA accepting the ω-language generated by $(FG\pi)$.

by the underlying non-deterministic FSA $\langle Q, \Sigma, \delta, Q_0 \rangle$. Furthermore, in general, it is possible to follow a BA transition function and generate an ω-string that is not accepted by the BA, i.e., $\mathscr{L}(B) \subseteq \mathscr{L}(cl(B))$. For example, in Figure 3.1, the ω-string $\{start\_moving\}\emptyset\emptyset...$, can be generated by the depicted BA by following its transition function, but it will keep the BA indefinitely in the non-accepting state 2, thus the ω-string is not accepted by the BA. In fact, given a BA where $\mathscr{L}(B) \subset \mathscr{L}(cl(B))$, one needs to find its strongly connected components to be able to generate ω-strings that visit accepting states infinite times. When $\mathscr{L}(B) = \mathscr{L}(cl(B))$ we say that B is closed.

Another important property of BA is that the class of non-deterministic BA is more expressive than the class of deterministic BA, i.e., there are non-deterministic BA that accept ω-languages that cannot be represented by a deterministic BA. In particular, the power-set construction used to build the deterministic version of a non-deterministic automaton is not valid for BA. An example of an ω-language that is not accepted by a deterministic BA is the language generated by the LTL formula $(FG\pi)$. The non-deterministic BA accepting this language is depicted in Figure 3.3. Note that the BA depicted in Figures 3.2 and 3.3 also show that the classes of LTL languages and ω-languages accepted by deterministic BA are not comparable.

In our case, we want to use a deterministic BA – so that we can have a deterministic supervisor realization – and also we want to avoid the need of applying search algorithms to the BA, hence we want to deal with BA B such that $\mathscr{L}(B) = \mathscr{L}(cl(B))$. To be able to obtain BA with these properties, we will restrict the LTL formulas we can use to specify behaviours to the so called safety fragment of LTL. We will be discussing this in the next section.

3.2 Restricting LTL to Safety Properties

One important classes of ω-languages are *safety* languages. Intuitively, a safety language specifies a property that states that something "bad" never happens in the system. Safety languages are associated with invariants of the system. In this section we will discuss safety languages and the restriction of LTL to specify only safety languages. In the following, let Σ be an alphabet, Σ^* the set of all finite strings composed of elements of Σ and Σ^ω the set of all the infinite strings composed of elements of Σ.

We start by defining the notion of *bad prefix*.

Definition 3.2.1 (Bad Prefix)**.** Let $L \subseteq \Sigma^\omega$ be an ω-language and $s \in \Sigma^*$. s is a bad prefix for L if for all $\sigma \in \Sigma^\omega$, $s.\sigma \notin L$, where $s.\sigma \in \Sigma^\omega$ is the concatenation of s and σ.

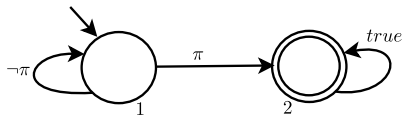


Figure 3.4: Deterministic BA accepting the language generated by $(F\pi)$.

A bad prefix is a finite string that cannot be extended to an infinite string that is in L. Safety languages are characterized using this notion.

Definition 3.2.2 (Safety Language)**.** Let $L \subseteq \Sigma^{\omega}$ be an ω-language. L is called a safety language if all $\sigma \notin L$ have a bad prefix for L.

For a safety language, one can always decide if an infinite string is not in the language by observing a finite part of it. This fact makes these types of languages especially "well-behaved" regarding the goals of this work, as we will explain next. One should notice that regular safety languages (i.e., safety languages that are accepted by BA) are strictly contained in regular ω-languages. The following proposition characterizes regular safety languages.

Proposition 3.2.1 ([Alpern and Schneider, 1987])**.** *Let B be a trim BA. Then $\mathscr{L}(B)$ is a safety language if and only if $\mathscr{L}(B) = \mathscr{L}(cl(B))$.*

Informally, this proposition states that for a trim BA that accepts a safety language, all the runs are accepting, i.e., there are no strongly connected components composed of only non-accepting states. Furthermore, since the power-set construction can be used for $cl(B)$, by defining all the states of the deterministic automaton as accepting states, we have the following corollary:

Corollary 3.2.1. *Let L be a regular safety language. Then there exists a deterministic BA B such that $\mathscr{L}(B) = L$.*

This corollary means that the class of regular safety languages is contained in the class of ω-languages accepted by deterministic BA. One can also prove that there are ω-languages accepted by deterministic BA which are not safety languages. For example, the BA in Figure 3.4, which generates the ω-language that satisfies the LTL formula $(F\pi)$ is deterministic but does not represent a safety language. Thus, the inclusion of the class of regular languages in the class of languages accepted by deterministic BA is strict.

We now focus our attention on LTL formulas that generate safety languages.

Definition 3.2.3 (Safe LTL)**.** Let φ be an LTL formula. We say that φ is an LTL safety formula if $\mathscr{L}(\varphi)$ is a safety language.

Safe LTL is strictly contained in regular safety languages. To see that, the BA in Figure 3.2 represents a safety language that is not representable in LTL. The following corollary is a direct consequence of the fact that LTL safety formulas generate safety languages.

Corollary 3.2.2. *Let φ be an LTL safety formula. Then there exists a deterministic and closed BA B_φ^D such that $\mathscr{L}(B_\varphi^D) = \mathscr{L}(\varphi)$.*

Hence, we have a procedure to go from an LTL safety formula φ to a deterministic BA for which every run is accepting, i.e., satisfies φ:

1. Translate φ into a BA B_φ using the LTL2BA tool.
2. Apply the power-set construction to $cl(trim(B_\varphi))$, obtaining B_φ^D, where all the sets are accepting states. Taking into account the discussion above, $\mathscr{L}(B_\varphi^D) = \mathscr{L}(\varphi)$.

This procedure has the drawback that the power-set construction is exponential in the size of the nondeterministic automaton. To mitigate this fact, our composition algorithms will take the non-deterministic BA as input and determinize it "on-the-fly", while performing the composition. By doing so, we only add a new state to the deterministic version of the BA if the synchronized run of the system with the BA can lead us to that state.

To complete our discussion, we just need to guarantee that our specifications are LTL safety formulas. Unfortunately, it is proven in [Sistla, 1994] that checking in an LTL formula generates a safety language is a PSPACE-complete problem. Thus, to avoid checking our specifications for safety, we will use *syntactically safe* LTL.

Definition 3.2.4 (Syntactically Safe LTL)**.** An LTL formula is syntactically safe if it is in the PNF and only the operators X, R, G and W are used.

As expected, given our nomenclature, syntactically safe LTL is a subset of safe LTL.

Proposition 3.2.2 ([Sistla, 1994])**.** *Let φ be a syntactically safe LTL formula. Then $\mathscr{L}(\varphi)$ is a safety language.*

From the designer point of view, the class of syntactically safe formulas is the class of interest when specifying safety properties in LTL. This can be justified because, while LTL formulas where F and U appear can be safe, this happens by "accident", in the sense that, in general, these operators require something to eventually happen. Thus, when a formula using F or U is a safety formula, it means that these operators are redundant in some way. A simple example of this is the formula $(G\pi) \wedge (F\pi)$, which is clearly a safety formula but contains the F operator. We will require our specifications to be given as syntactically safe LTL formulas.

To summarize this overview of safety languages and LTL, figure 3.5 a Venn diagram of the different classes of languages we presented. We also refer the interested reader to [Vardi, 1996] for an introduction to Büchi automata and its properties and to [Kupferman and Vardi, 2001] for an introduction to model checking safety properties.

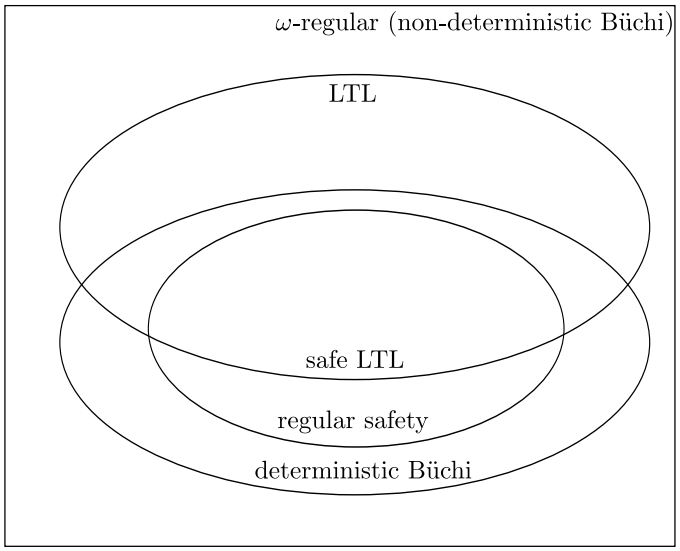


Figure 3.5: Venn diagram for the classes of ω-languages described in this chapter.

CHAPTER 4

Composition of Büchi Automaton with System Models

In this chapter, we present the main algorithms developed in this thesis. These are composition algorithms between the system models presented in Chapter 2 and the Büchi automata obtained from LTL specifications, using the LTL2BA [Gastin and Oddoux, 2001] tool. The result of these compositions is a system model whose generated language is the generated language of the unsupervised original system restricted to the ω-strings that satisfy the specification. This model will be used as a coarse structure for building the supervisor, since it does not guarantee admissibility and deadlock-freeness, which are two fundamental notions for a proper supervisor in the Ramadge-Wonham framework [Ramadge and Wonham, 1989]. We will see how one can build a supervisor from these structures, using available approaches in the literature, in the next chapter. We follow the same order as Chapter 2, starting with the composition of the Büchi automaton with an FSA system model, and then moving to PN system models with symbolic state description and algebraic state description. In the following, given a finite set A, let $|A|$ denote the number of elements in A and A^* the set of all finite strings built from A (including the empty string ε).

The problem to be solved is the same for all system models, so before we go to the specifics of how to solve it for each case, we will state it. For each system model, we want to build an algorithm that solves the following problem:

Problem 4.1. Let G be one of the three system models we presented and φ be a syntactically safe LTL formula written over Π_{symb} or Π_{alg}, according to the system model being taken into account. Build a system model G_φ, of the same type as G, such that:

$$\sigma \in \mathscr{L}(G_\varphi) \text{ if and only if } \sigma \in \mathscr{L}(G) \text{ and } \sigma \Vdash \varphi \tag{4.1}$$

4.1 Finite State Automata

We start by showing the composition algorithm for FSA. The contents of this section were first described in [Lacerda and Lima, 2009].

To ensure that the FSA supervisor takes the initial state of the system into account, we assume that G has an *init* state as the initial state and an *initialize_system* event, with $\mu(init) = \emptyset$, $\delta(init, initialize\_system) = q$, where q is the state representing the initial state of the system and no $q \in Q$ and $e \in E$ such that $\delta(q,e) = init$. This dummy state is needed because the first element of $\mathscr{L}(G)$ is (e_1, q_1) such that $\delta(q_0, e_1) = q_1$ and we want to take the real initial state of the system into account.

The problem will be solved by defining a composition function that, given the FSA system model of the system $G = \langle Q, E, \delta, q_0, D, \mu \rangle$ and the (non-deterministic) BA $B_\varphi = \left\langle Q^B, 2^{E \cup D}, \delta^B, q_0^B, Q_f^B \right\rangle$, build an FSA system model that simulates the running of these two automata and parallel, such that G only fires an event if this event, in conjunction with the state it drives G to, satisfies one of the active transition labels in the current state of B_φ. The construction of this FSA follows Algorithm 1.

The algorithm creates an FSA that simulates a run in parallel of the FSA model of the system and the determinization (built "on-the-fly") of the BA. The states of the FSA supervisor are pairs where the first element is the current state of the FSA model and the second element is set of states where the BA can be in.

We start by analysing the pair composed of the initial state of the FSA model and the singleton which contains the initial state of the BA (line 4). When this analysis creates new pairs that have not been analysed yet, they are added to a FIFO queue. While this queue is not empty, the algorithm keeps analysing new states (the maximum amount of analysed states will be $|Q| \times |2^{Q^B}|$, after which the loop will stop).

For a given pair $(q, \overline{q})$ of an FSA state q and a set of BA states $\overline{q}$, we analyse each active event e in state q (line 10). In this analysis, we need to check, for each DNF formula ψ labelling a transition from a Büchi state $q^B \in \overline{q}$, if the firing of e satisfies ψ. The firing of e satisfies ψ if the valuation obtained from the set composed of e plus the state description of state $\delta(q,e)$ satisfies ψ. If it does, we add the state that is targeted by the transition labelled with ψ to the set $\overline{q}'$ of next Büchi states (lines 14–18).

If the firing of e satisfied at least one of the ψ labels, i.e., if $\overline{q}'$ is not empty, it means that e can occur when the system is in state q and the BA determinization is in state $\overline{q}$. This firing makes the system evolve to $\delta(q,e)$ and the BA determinization to the set $\overline{q}'$. If this pair is still not analysed, it is added to the queue, as we explained before (lines 20–30).

Hence, Problem 4.1 is solved because we only allow the firing of events in G that lead to sequences of events plus state descriptions that are in conformity with the BA transitions. Furthermore, given that φ is syntactically safe, we can assume that $Q_f^B = Q^B$, thus any string generated by B_φ satisfies φ, as discussed before. Also, we analyse all the possible combinations of accessible states in $Q \times 2^{Q^B}$, hence all the possible ω-strings in $\mathscr{L}(G)$ that satisfy φ are taken into account.

Algorithm 1 Büchi/FSA System Composition

Input: FSA system model $G = \langle Q, E, \delta, q_0, D, \mu \rangle$ and syntactically safe LTL formula φ, written over $D \cup E$

Output: FSA system model $G_\varphi = \langle Q', E, \delta', q'_0, D, \mu' \rangle$

$B_\varphi = \left\langle Q^B, 2^{E \cup D}, \delta^B, q_0^B, Q_f^B \right\rangle \leftarrow LTL2BA(\varphi)$
add state $(q_0, \{q_0^B\})$ to Q'
$\mu'((q_0, \{q_0^B\})) \leftarrow \mu(q_0)$
$q'_0 \leftarrow (q_0, \{q_0^B\})$
$states\_queue.push((q_0, \{q_0^B\}))$
while $states\_queue \neq \emptyset$ **do**
  $current\_sup\_state \leftarrow states\_queue.pop()$
  $current\_system\_state \leftarrow current\_sup\_state[1]$ {[1] represents the first element of the pair}
  $current\_buchi\_states \leftarrow current\_sup\_state[2]$ {[2] represents the second element of the pair}
  for all $e \in \Gamma_G(current\_system\_state)$ **do**
    $next\_system\_state \leftarrow \delta(current\_system\_state, e)$
    $next\_buchi\_states \leftarrow \emptyset$
    for all $q^B \in current\_buchi\_states$ **do**
      for all $\psi \in \Gamma_{B_\varphi}(q^B)$ **do**
        if $\{e\} \cup \mu(next\_system\_state) \Vdash \psi$ **then**
          $next\_buchi\_states \leftarrow next\_buchi\_states \cup \delta^B(q^B, \psi)$
        end if
      end for
    end for
    if $next\_buchi\_states \neq \emptyset$ **then**
      $next\_sup\_state \leftarrow (next\_system\_state, next\_buchi\_states)$
      if $next\_sup\_state \notin Q'$ **then**
        add $next\_sup\_state$ to Q'
        $\mu'(next\_sup\_state) \leftarrow \mu(next\_system\_state)$
        $states\_queue.push(next\_sup\_state)$
      end if
      $\delta'(current\_sup\_state, e) \leftarrow next\_sup\_state$
    end if
  end for
end while

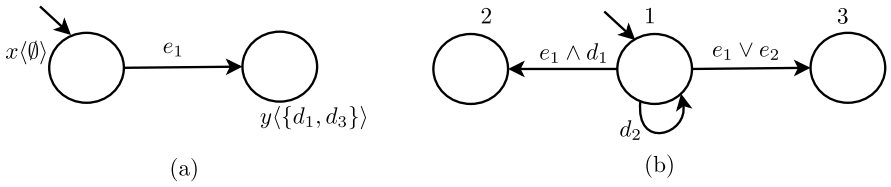


Figure 4.1: (a) A fragment of an FSA system model (b) A fragment of a BA

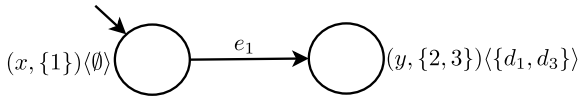


Figure 4.2: A fragment of the obtained FSA system model

Example 4.1.1 (Büchi/FSA System Model Composition)**.** Consider the fragment of an FSA system model and of a BA depicted in Figure 4.1. The event set is $E = \{e_1, e_2\}$ and the state description symbols are $\{d_1, d_2, d_3\}$.

We will exemplify on how to compose the transitions from Büchi state 1 with the transition depicted in the FSA. We note that the set S obtained from going from state x to state y through the transition labelled by e_1 is given by $S = \{e_1\} \cup \{d_1,\ d_3\}$. We need to check what are the labels of the transitions in the BA that are satisfied by S:

$$S \nVdash d_2$$

$$S \Vdash e_1 \wedge d_1$$

$$S \Vdash e_1 \vee d_2$$

Hence, when e_1 occurs in the FSA, with the BA in state 1, it can go to state 2 or to state 3. The supervisor obtained from this analysis is depicted in Figure 4.2.

Note that, even assuming that the original FSA system model does not have deadlocks, it is possible for the composition to have them. It suffices that we enter a state q where for all $e \in \Gamma_G(q)$, $\{e\} \cup \mu(\delta(q,e))$ does not satisfy any of the active transition labels from the possible states the Büchi automaton can be in when g is in q. Thus, the result of the composition is not deadlock-free. We will tackle this problem in the next chapter.

4.2 Petri Nets

In this section, we deal with PN composition. We first show how to add minimal satisfying transitions for both PN system models with symbolic and with algebraic state description, which will be the

basic building block of the composition algorithm, and the main difference between the 2 approaches. Afterwards, we present the composition algorithm, abstracting the type of state description used. We finish with showing how to trim the composition, deleting transitions that are never active.

4.2.1 Adding Minimal Satisfying Transitions

Symbolic State Description

Our goal is to avoid building the whole state space (i.e., the whole set of reachable markings) of the PN to apply our method. In order to be able to avoid it, we will directly work on the structure of the system, by building transitions that can only be fired when they lead the PN to a marking where a given conjunctive clause is satisfied.

We start by defining the set of places associated to a conjunctive clause written over D.

Definition 4.2.1. Let $G = \langle L, D, \mu, K \rangle$ be a symbolic PN system model, and $\psi = \psi_E \wedge \psi_D$ be a conjunctive clause written over $E \cup D$, where ψ_E is a conjunctive clause written over E and $\psi_D = \bigwedge_{i=1}^{n} l_i$ is a conjunctive clause written over D. We define the set of places associated to ψ as:

$$P_\psi = \bigcup_{i=1}^{n} \mu(l_i) \tag{4.2}$$

The set of places associated to ψ represents the places that require a token for ψ to be satisfied. Given that we will only analyse transitions during the composition, we will need to add new arcs to them that guarantee that they only fire when they lead the PN to a marking that satisfies ψ.

Definition 4.2.2 (Minimal Satisfying Transition)**.** Let $G = \langle L, D, \mu \rangle$ be a symbolic PN system model, $t \in T$ and $\psi = \psi_D \wedge \psi_E = \bigwedge_{i=1^n} l_i \wedge \bigwedge_{i=1}^{m} l'_i$ a conjunctive clause written over $D \cup E$. If $t^\bullet(p) - {}^\bullet t(p) \neq -1$ for all $p \in P_\psi$ and $\{\ell(t)\} \Vdash \psi_E$, the minimal satisfying transition $t_{t,\psi}$ obtained from t and ψ is defined as:

$${}^\bullet t_{t,\psi}(p) = \begin{cases} {}^\bullet t(p) & \text{if } p \notin P_\psi \text{ or } t^\bullet(p) = 1 \\ 1 & \text{if } p \in P_\psi \text{ and } t^\bullet(p) = 0 \end{cases} \tag{4.3}$$

$$t^\bullet_{t,\psi}(p) = \begin{cases} t^\bullet(p) & \text{if } p \notin P_\psi \text{ or } t^\bullet(p) = 1 \\ 1 & \text{if } p \in P_\psi \text{ and } t^\bullet(p) = 0 \end{cases} \tag{4.4}$$

In the cases where exists $p \in P_\psi$ such that $t^\bullet(p) - {}^\bullet t(p) = -1$ or $\{\ell(t)\} \nVdash \psi_E$, we say that the minimal satisfying transition is undefined.

The minimal satisfying transition for t and ψ is a transition that is active if and only if t is active and the firing of t will drive the PN to a marking that, in conjunction with $\ell(t)$, satisfies the conjunctive clause ψ. Furthermore, when the minimal satisfying transition is not defined, then either the firing of t does not satisfy ψ_E or none of the possible markings reached immediately after the firing of t

satisfy ψ_D. The minimal satisfying transition $t_{t,\psi}$ has the same input and output places as t, except for some places $p \in P_\psi$. Note that, given that we are dealing with a consistent PN model, $t^\bullet(p) - {}^\bullet t(p) \in \{-1,0,1\}$, since we assume $t^\bullet(p) \leq 1$ and ${}^\bullet t(p) \leq 1$ for all $p \in P_D$.Thus, for places $p \in P_\psi$, we have three possibilities:

1. If $t^\bullet(p) - {}^\bullet t(p) = -1$, then immediately after the firing of t, p will always have 0 tokens. In this case, the firing of t will never satisfy the literal associated to p, independently of the marking it is fired from. Thus, it will never satisfy ψ_D either, thus the minimal satisfying transition is not defined.

2. If $t^\bullet(p) = 1$, then immediately after the firing of t, p will always have 1 token. In this case, the firing of t will always satisfy the literal associated to p, independently of the marking it is fired from. Thus, we do not need to add new arcs between t and p.

3. If $t^\bullet(p) = 0$ and we are not in case 1., i.e., ${}^\bullet t(p) = 0$, then the number of tokens in p immediately after the firing of t depends on the marking from which t was fired. In fact, this number of tokens is exactly the same before and after the firing of t. Thus, we need to guarantee that t only fires when p already has one token and that its firing does not change the number of tokens in p. To do this, we simply add an arc with weight 1 from p to $t_{t,\psi}$, so that the minimal satisfying transition can only fire when there is already a token in p, and an arc with weight 1 from $t_{t,\psi}$ to p, so that the number of tokens in p does not change after the firing, and the minimal satisfying transition remains coherent with the transition it is being built from.

For the subformula of the conjunctive clause referring to events ψ_E, the analysis is much simpler, given that from the firing of a transition we know the truth value for all events: the event $\ell(t)$ occurred and all other events did not occur[1]. Before formally stating and proving the properties of the minimal satisfying transition, we give an example of its construction.

Example 4.2.1 (Building Minimal Satisfying Transitions)**.** Consider the fragment of a PN system model depicted in Figure 4.3 (a) and the formula $\psi = d_1 \wedge \neg d_2 \wedge \neg e_3$, i.e., $\psi_D = d_1 \wedge \neg d_2$ and $\psi_E = \neg e_3$. We also depict, in Figure 4.3 (b), the minimal satisfying transitions built for t_1, t_2, t_3 and t_4. Note that for t_3, $\{e_3\} \not\Vdash \neg e_3$ and, for t_4, $t_4^\bullet(\mu(d_1)) - {}^\bullet t_4(\mu(d_1)) = -1$, thus the minimal satisfying transition is not defined for these transitions. For transitions t_1, we just add arcs to and from the place corresponding to d_1, because the place corresponding to $\neg d_2$ is part of the postset of the transition, hence we are guaranteed that $\neg d_2$ will always be satisfied immediately after the firing of t_1. For t_2 we also need to add arcs from and to p_1 and p_4 because the structure of the PN does not give us any direct information about the number of tokens in both p_1 and p_4 immediately after the firing of t_2.

Proposition 4.2.1. *Let G be a PN system model and $\psi = \psi_D \wedge \psi_E$ be a conjunctive clause written over $D \cup E$. The minimal satisfying transition $t_{t,\psi}$ has the following properties:*

[1] Thus if ψ_E has more than one positive literal, the minimal satisfying transition will be undefined for all $t \in T$.

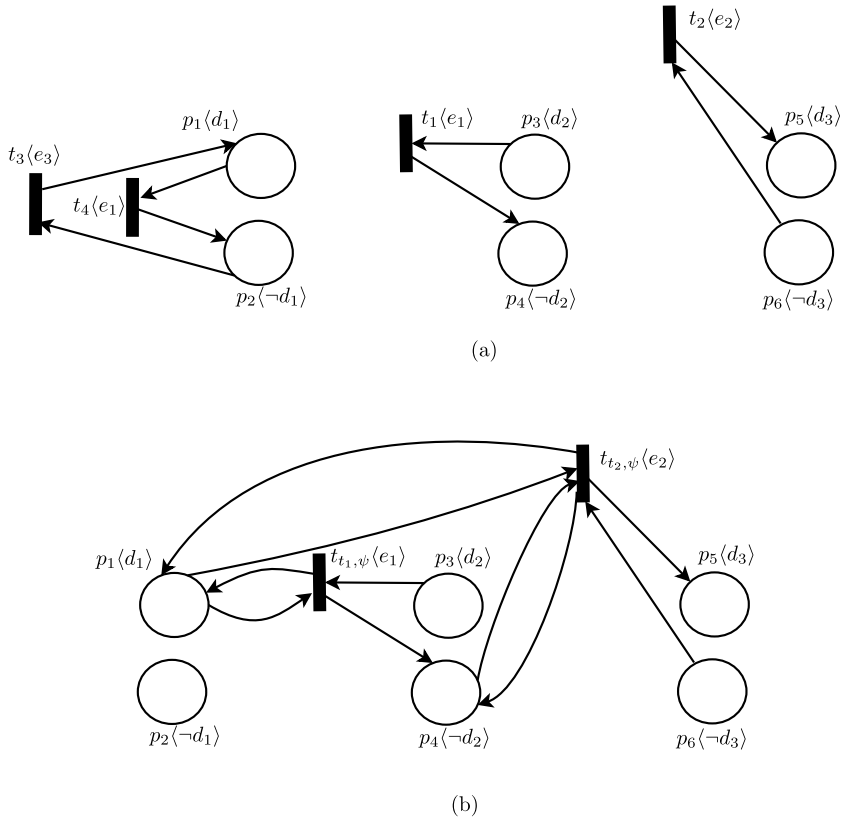


Figure 4.3: Illustration of the minimal satisfying transition construction. (a) A fragment of a PN. (b) The minimal satisfying transitions built for t_1, t_2, t_3 and t_4, for the formula $\psi = d_1 \wedge \neg d_2 \wedge \neg e_3$.

1. *For all $M \in R(G)$, $M \xrightarrow{t_{t,\psi}} M'$ if and only if $M \xrightarrow{t} M'$ and $\{\ell(t)\} \cup D_{M'} \Vdash \psi$.*

2. *If $t_{t,\psi}$ is undefined, then for all $M' \in R(G)$ for which there exists $M \in R(G)$ such that $M \xrightarrow{t} M'$, $\{\ell(t)\} \cup D_{M'} \nVdash \psi$.*

Proof.

1. We first prove that for all $M \in R(G)$, if $M \xrightarrow{t_{t,\psi}} M'$ then $M \xrightarrow{t} M'$ and $\{\ell(t)\} \cup D_{M'} \Vdash \psi$. To prove that $M \xrightarrow{t} M'$, we start by noting that pre and postsets of t and $t_{t,\psi}$ coincide for all places p such that $p \notin P_\psi$ or $t^\bullet(p) = 1$. Hence, we only need to analyse places p such that $p \in P_\psi$ and $t^\bullet(p) = 0$. We need to prove that:

   (i) $(M \xrightarrow{t})$, i.e., $M \geq {}^\bullet t$. For all $p \in \mu(\psi)$ such that $t^\bullet(p) = 0$, we have:

   $$M(p) \geq {}^\bullet t_{t,\psi}(p) = 1 \geq {}^\bullet t(p) \tag{4.5}$$

   (ii) $M - {}^\bullet t + t^\bullet = M - {}^\bullet t_{t,P',b} + t^\bullet_{t,P',b}$. For all $p \in \mu(\psi)$ such that $t^\bullet(p) = 0$, we have:

   $$M(p) - {}^\bullet t_{t,\psi}(p) + t^\bullet_{t,\psi}(p) = M(p) - 1 + 1 = M(p) - 0 + 0 = M(p) - {}^\bullet t(p) + t^\bullet(p) \tag{4.6}$$

   Note that $t^\bullet(p) = 0$ implies that ${}^\bullet t(p) = 0$, because the minimal satisfying transitions is only defined if $t^\bullet(p) - {}^\bullet t(p) \geq 0$.

   To prove that $\{\ell(t)\} \cup D_{M'} \Vdash \psi$, we will prove that $\{\ell(t)\} \cup D_{M'} \Vdash \psi_E$ and $\{\ell(t)\} \cup D_{M'} \Vdash \psi_D$. For ψ_E, the result follows directly from the definition. For ψ_D, we start by noting that, according to the PN firing rule, it suffices to prove that $t^\bullet_{t,\psi}(\mu(l_i)) = 1$ for all $i = 1, ..., n$. This fact is a direct consequence of the definition of $t_{t,\psi}$.

   We now prove that for all $M \in R(G)$, if $M \xrightarrow{t} M'$ and $\{\ell(t)\} \cup D_{M'} \Vdash \psi$, then $M \xrightarrow{t_{t,\psi}} M'$. Like in the previous case, we only need to analyse places p such that $p \in P_\psi$ and $t^\bullet(p) = 0$ because these are the places for which the transitions have different pre and postsets:

   (i) $(M \xrightarrow{t_{t,\psi}})$, i.e., $M \geq {}^\bullet t_{t,\psi}$. Given that $t^\bullet(p) = 0$ implies ${}^\bullet t(p) = 0$, we have $M(p) = M'(p)$. Furthermore, by hypothesis, $M'(p) = 1$ for all $p \in P_\psi$. Thus, $M(p) = 1$, i.e., $M(p) \geq {}^\bullet t_{t,\psi}(p)$, by definition of ${}^\bullet t_{t,\psi}$.

   (ii) $M - {}^\bullet t_{t,\psi} + t^\bullet_{t,\psi} = M - {}^\bullet t + t^\bullet$. This was already proven in point (ii) above.

2. Since $t_{t,\psi}$ is undefined, either there exists $p \in P_\psi$ such that $t^\bullet(p) - {}^\bullet t(p) = -1$ or $\{\ell(t)\} \nVdash \psi_E$. If $\{\ell(t)\} \nVdash \psi_E$, it is clear that $\{\ell(t)\} \nVdash \psi$. For the case where there exists $p \in P_\psi$ such that $t^\bullet(p) - {}^\bullet t(p) = -1$, let M, $M' \in R(G)$ such that $M \xrightarrow{t} M'$. We have that:

   $$M'(p) = M(p) + {}^\bullet t(p) - t^\bullet(p) = M(p) - 1$$

Given that we are dealing with a consistent PN model and t is active, $M(p) = 1$, thus $M'(p) = 0$. Hence, M' does not satisfy the literal associated to p. Given that this literal occurs in ψ (because $p \in P_\psi$), it is clear that $D_{M'} \not\Vdash \psi$.

□

Adding a Knowledge Base

When building the minimal satisfying transition for a given $t \in T$ and ψ, the only information we have about the marking obtained from the firing of t is given by its pre and postsets. This lack of information will have an impact on the size of the compositions, as we will see later. To mitigate this, we allow the designer to add more information that he knows is valid in all markings of the system. This additional information is not required for building minimal satisfying transitions, it is simply used to reduce the number of arcs added when building them, or to immediately conclude that the firing of a transition will never satisfy a given conjunctive clause, thus avoiding the building of minimal satisfying transitions that will never be active.. With this, one may reduce the number of transitions in the PN obtained from the composition. Furthermore, the information in this knowledge base is assumed to be valid in the model, thus the designer should only add information that he/she is certain to hold. One way to find such information for larger-scale models is, for example, to use invariant analysis, however, in our case, we will simply use information that can be obtained an intuitive analysis of the model.

Definition 4.2.3 (Knowledge Base)**.** Let $G = \langle L, D, \mu \rangle$. A knowledge base K for G is a relation between conjunctive clauses written over $E \cup D$ and conjunctive clauses written over D such that, for all $M,\ M' \in R(G)$ and $t \in T$ such that $M \xrightarrow{t} M'$:

$$\text{For all } (\varphi, \psi) \in K, \text{ if } \{\ell(t)\} \cup D_{M'} \Vdash \varphi \text{ then } D_{M'} \Vdash \psi \tag{4.7}$$

Intuitively, $(\varphi, \psi) \in K$ means that whenever φ is satisfied in our model, then ψ is also satisfied.

Example 4.2.2 (Tea/Coffee Machine)**.** In Figure 4.4, we depict again the PN system model for the tea and coffee machine, to facilitate the reading.

There are several facts about this model, which are not directly encoded in the output places of the transitions, that can be added to K:

1. After either transition t_1 or t_5 fire, there will always be a token in places p_3 and p_6, i.e., neither tea nor coffee can be being prepared immediately after one of them is requested:

$$(request\_tea, \neg making\_tea \wedge \neg making\_coffee) \in K \tag{4.8}$$

$$(request\_coffee, \neg making\_tea \wedge \neg making\_coffee) \in K \tag{4.9}$$

2. After transitions t_2 or t_3 fire, there will always be a token in place p_6, i.e., a tea cannot be being prepared immediately before or after the preparation of a coffee (note that we could also

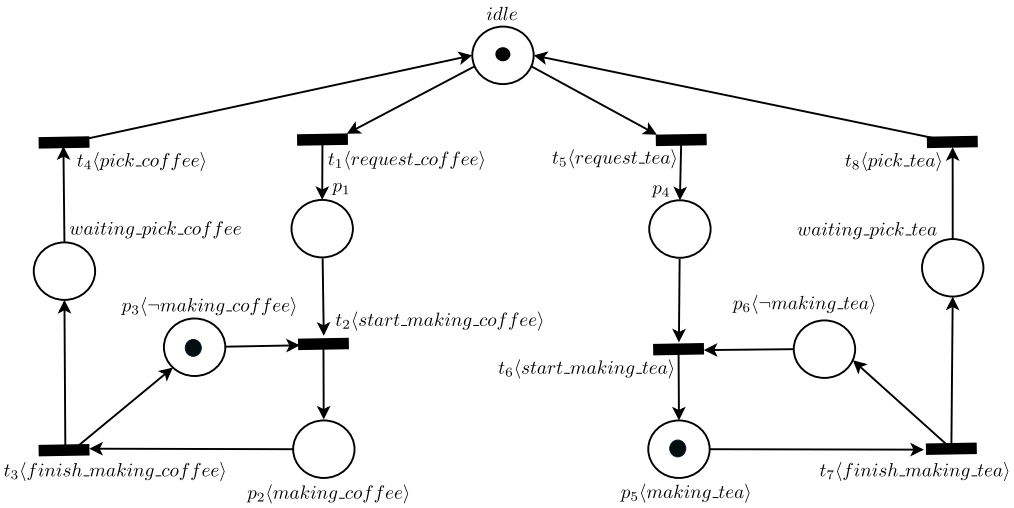


Figure 4.4: PN system model with symbolic state description for a tea/coffee machine

add that there will be a token in place p_2 after t_2 fires and a token in p_3 after t_3 fires, but this information is directly encoded in the postsets of t_2 and t_3):

$$(start\_making\_coffee, \neg making\_tea) \in K \tag{4.10}$$

$$(finish\_making\_coffee, \neg making\_tea) \in K \tag{4.11}$$

3. The dual of statement 2 is also valid:

$$(start\_making\_tea, \neg making\_coffee) \in K \tag{4.12}$$

$$(finish\_making\_tea, \neg making\_coffee) \in K \tag{4.13}$$

4. If there is a token in place p_2, then there must be a token in place p_6, i.e., if a coffee is being prepared, then a tea cannot be also being prepared:

$$(making\_coffee, \neg making\_tea) \in K \tag{4.14}$$

5. The dual of statement 4 is also valid:

$$(making\_tea, \neg making\_coffee) \in K \tag{4.15}$$

In order to define the minimal satisfying transition for a PN system model with a knowledge base K, we will need to introduce some definitions related to conjunctive clauses, given that we use them to specify the knowledge base.

Definition 4.2.4 (Literals of a Conjunctive Clause)**.** Let $\varphi = \wedge_{i=1}^{n} l_i$ be a conjunctive clause written over Π. We define the sets of positive and negative literals of φ as, respectively:

$$lit_{\varphi}^{+} = \{\pi \in \Pi \mid \text{exists } i \in \{1,...,n\} \text{ such that } \pi = l_i\} \tag{4.16}$$

$$lit_{\varphi}^{-} = \{\pi \in \Pi \mid \text{exists } i \in \{1,...,n\} \text{ such that } \neg\pi = l_i\} \tag{4.17}$$

Note that, for a conjunctive clause φ, $S \Vdash \varphi$ if and only if $lit_{\varphi}^{+} \subseteq S$ (all positive literals are satisfied in S) and $lit_{\varphi}^{-} \cap S = \emptyset$ (none of the negative literals is satisfied in S). We will assume that the conjunctive clauses φ are satisfiable, i.e., $lit_{\varphi}^{+} \cap lit_{\varphi}^{-} = \emptyset$.[2]

Definition 4.2.5 (Entailment)**.** Let φ, ψ be 2 propositional formulas written over a set of atomic propositions Π. We say that φ entails ψ, written $\varphi \vDash \psi$, if for all $S \in 2^{\Pi}$, if $S \Vdash \varphi$, then $S \Vdash \psi$.

A propositional formula φ entails another formula ψ when all the subsets of Π that satisfy φ also satisfy ψ. This definition of entailment can be specialized for conjunctive clauses, by using simple set relations between the literals of φ and the literals of ψ. We state this in the next proposition, which provides an efficient way to check entailment of conjunctive clauses.

Proposition 4.2.2. *Let φ, ψ be 2 conjunctive clauses. Then:*

1. *$\varphi \vDash \psi$ if and only if $lit_{\psi}^{+} \subseteq lit_{\varphi}^{+}$ and $lit_{\psi}^{-} \subseteq lit_{\varphi}^{-}$.*
2. *$\varphi \vDash \neg\psi$ if and only if $lit_{\psi}^{+} \cap lit_{\varphi}^{-} \neq \emptyset$ or $lit_{\psi}^{-} \cap lit_{\varphi}^{+} \neq \emptyset$.*

Proof.

1. We start by proving that if $\varphi \vDash \psi$, then $lit_{\psi}^{+} \subseteq lit_{\varphi}^{+}$ and $lit_{\psi}^{-} \subseteq lit_{\varphi}^{-}$, using the contrapositive. Assume that exists $\pi \in lit_{\psi}^{+}$ such that $\pi \notin lit_{\varphi}^{+}$. This means that exists $S \in 2^{\Pi}$ such that $S \Vdash \varphi$ and $\pi \notin S$. Thus, $S \nVdash \psi$, i.e., $\varphi \nvDash \psi$. The proof for the sets of negative literals in analogous.

   We now prove that if $lit_{\psi}^{+} \subseteq lit_{\varphi}^{+}$ and $lit_{\psi}^{-} \subseteq lit_{\varphi}^{-}$, then $\varphi \vDash \psi$, again using the contrapositive. Assume that exists S such that $S \Vdash \varphi$ and $S \nVdash \psi$. Hence, there exists $\pi \in \Pi$ such that (i) $\pi \notin S$ and $\pi \in lit_{\psi}^{+}$ or (ii) $\pi \in S$ and $\pi \in lit_{\psi}^{-}$. Assume π satisfies (i). Given that $S \Vdash \varphi$, $\pi \notin lit_{\varphi}^{+}$. Thus, $lit_{\psi}^{+} \not\subseteq lit_{\varphi}^{+}$. Assume π satisfies (ii). Given that $S \Vdash \varphi$, $\pi \notin lit_{\varphi}^{-}$. Thus, $lit_{\psi}^{-} \not\subseteq lit_{\varphi}^{-}$.

2. We start by proving that if $\varphi \vDash \neg\psi$, then $lit_{\psi}^{+} \cap lit_{\varphi}^{-} \neq \emptyset$ or $lit_{\psi}^{-} \cap lit_{\varphi}^{+} \neq \emptyset$, using the contrapositive. Assume that $lit_{\psi}^{+} \cap lit_{\varphi}^{-} = \emptyset$ and $lit_{\psi}^{-} \cap lit_{\varphi}^{+} = \emptyset$, and let $S = lit_{\varphi}^{+} \cup lit_{\psi}^{+}$. It is clear that $lit_{\varphi}^{+} \subseteq S$, and $lit_{\varphi}^{-} \cap S = \emptyset$ because $lit_{\varphi}^{-} \cap lit_{\psi}^{+} = \emptyset$, by hypothesis. Thus, $S \Vdash \varphi$. Using the same reasoning, we can prove that $S \Vdash \psi$, i.e., $S \nVdash \neg\psi$. Thus, we can conclude that $\varphi \nvDash \neg\psi$.

   We now prove that if $lit_{\psi}^{+} \cap lit_{\varphi}^{-} \neq \emptyset$ or $lit_{\psi}^{-} \cap lit_{\varphi}^{+} \neq \emptyset$ then $\varphi \vDash \neg\psi$, again using the contrapositive. Assume that $\varphi \nvDash \neg\psi$, i.e., there exists $S \in 2^{\Pi}$ such that $S \Vdash \varphi$ and $S \nVdash \neg\psi$ (thus,

[2] It is easy to see that a conjunctive clause is satisfiable if and only if no atomic proposition and its negation appear in the formula.

$S \Vdash \psi$). Hence, we can conclude that (i) $lit_{\varphi}^{+} \subseteq S$ and (ii) $lit_{\varphi}^{-} \cap S = \emptyset$ and (iii) $lit_{\psi}^{+} \subseteq S$ and (iv) $lit_{\psi}^{-} \cap S = \emptyset$. From (i) and (iv), we conclude that $lit_{\psi}^{-} \cap lit_{\varphi}^{+} = \emptyset$ and from (ii) and (iii) that $lit_{\psi}^{+} \cap lit_{\varphi}^{-} = \emptyset$.

□

To finish the definitions about conjunctive clauses needed to define the minimal satisfying transition for a PN system model equipped with a knowledge base K, we need to define the conjunctive clause associated with the firing of a transition t. This will be a conjunctive clause that encodes the fact that event $\ell(t)$ has fired and also that in all markings obtained immediately after the firing of t the literals l for which $\mu(l)$ is in the postset of t are guaranteed to be satisfied.

Definition 4.2.6 (Conjunctive Clause Associated with the Firing of a Transition)**.** Let G be a PN system model and $t \in T$. The conjunctive clause written over $D \cup E$ associated with the firing of t is given by:

$$\varphi_t = \ell(t) \wedge \bigwedge_{\substack{e \in E \\ e \neq \ell(t)}} \neg e \wedge \bigwedge_{l \in lit(D) \mid t^{\bullet}(\mu(l))=1} l \tag{4.18}$$

Example 4.2.3 (Tea/Coffee Machine)**.** Regarding the tea/coffee machine example depicted in Figure 2.4, the conjunctive clause associated with the firing of t_3 is given by:

$$\varphi_{t_3} = finish\_making\_coffee \wedge \bigwedge_{\substack{e \in E \\ e \neq finish\_making\_coffee}} \neg e \wedge \neg making\_coffee \tag{4.19}$$

The next lemma will be used when proving the properties of the minimal satisfying transition for PN system models with a knowledge base K and is a direct consequence of the definition.

Lemma 4.2.1. *Let G be a PN system model, $t \in T$ and φ_t the conjunctive clause associated to the firing of t. Then, for all M, $M' \in R(G)$ such that $M \xrightarrow{t} M'$, $\{\ell(t)\} \cup D_{M'} \Vdash \varphi_t$.*

To give the reader further insight about the conjunctive clause associated to the firing of t and its relation with the minimal satisfying transition, we prove the following property.

Proposition 4.2.3. *Let G be a PN system model, $t \in T$ and ψ a conjunctive clause written over $D \cup E$. If $\varphi_t \vDash \psi$, then ${}^{\bullet}t_{t,\psi} = {}^{\bullet}t$ and $t_{t,\psi}^{\bullet} = t^{\bullet}$.*

Proof. The proof of this proposition is direct from the definitions. We have that $\varphi_t \vDash \psi$, hence, by definition of entailment:

$$lit_{\psi}^{+} \subseteq lit_{\varphi_t}^{+} \tag{4.20}$$

$$lit_{\psi}^{-} \subseteq lit_{\varphi_t}^{-} \tag{4.21}$$

Thus, we can conclude that the places associated with literals in ψ are contained in the places associated with literals in φ_t, i.e., $P_{\psi} \subseteq P_{\varphi_t}$. Furthermore, by definition of φ_t, $t^{\bullet}(p) = 1$ for all $p \in P_{\varphi_t}$.

Thus, $t^\bullet(p) = 1$ for all $p \in P_\psi$. Hence, all places will satisfy the condition of equation 4.3 for which ${}^\bullet t_{t,\psi}(p) = {}^\bullet t(p)$ and $t^\bullet_{t,\psi}(p) = t^\bullet(p)$.

□

We are now in conditions to define the minimal satisfying transition with knowledge base K.

Definition 4.2.7 (Minimal Satisfying Transition with Knowledge Base)**.** Let $G = \langle L, D, \mu \rangle$ be a symbolic PN system model, K a knowledge base over $D \cup E$, $t \in T$ and $\psi = \psi_D \wedge \psi_E = \bigwedge_{i=1^n} l_i \wedge \bigwedge_{i=1}^{m} l'_i$ a conjunctive clause written over $D \cup E$. If:

1. $t^\bullet(\mu(l_i)) - {}^\bullet t(\mu(l_i)) \neq -1$ for all $i = 1, ..., n$,
2. $\{\ell(t)\} \Vdash \psi_E$,
3. for all $(\gamma, \theta) \in K$, if $\varphi_t \vDash \gamma$, then $\theta \nvDash \neg\psi$.

Then, the minimal satisfying transition $t_{t,\psi,K}$ obtained from t, ψ and K is defined as:

$$ {}^\bullet t_{t,\psi,K}(p) = \begin{cases} {}^\bullet t(p) & \text{if } p \notin P_\psi \text{ or } t^\bullet(p) = 1 \text{ or exists } (\gamma,\theta) \in K \text{ such that } \varphi_t \vDash \gamma \text{ and } p \in P_\theta \\ 1 & \text{if } p \in P_\psi \text{ and } t^\bullet(p) = 0 \text{ and for all } (\gamma,\theta) \in K \text{ such that } \varphi_t \vDash \gamma,\ p \notin P_\theta \end{cases} \tag{4.22} $$

$$ t^\bullet_{t,\psi,K}(p) = \begin{cases} t^\bullet(p) & \text{if } p \notin P_\psi \text{ or } t^\bullet(p) = 1 \text{ or exists } (\gamma,\theta) \in K \text{ such that } \varphi_t \vDash \gamma \text{ and } p \in P_\theta \\ 1 & \text{if } p \in P_\psi \text{ and } t^\bullet(p) = 0 \text{ and for all } (\gamma,\theta) \in K \text{ such that } \varphi_t \vDash \gamma,\ p \notin P_\theta \end{cases} \tag{4.23} $$

In the cases where at least one of the conditions 1., 2. or 3. is not satisfied, we say that the minimal satisfying transition is undefined.

The idea of building the minimal satisfying transition $t_{t,\psi,K}$ from t, ψ and K builds upon the notion of building the minimal satisfying transition. It uses the extra information given by K to reduce the number of arcs added to the minimal satisfying transition, or to conclude that it is impossible for the firing of a transition to satisfy the conjunctive clause, thus the minimal satisfying transition is not defined. To include this new information, we add an extra condition for the minimal satisfying transition to be undefined: if there exists an element (γ, θ) of K such that the firing of t satisfies γ, and θ entails the negation of ψ, it means that, given the information provided by K, all markings obtained immediately after the firing of t do not satisfy ψ. Thus, in this case, the minimal satisfying transition is undefined. We also add an extra condition to keep the arcs between the minimal satisfying transition and a given place p equal to the arcs between t and p: if there exists an element (γ, θ) of K such that the firing of t satisfies γ, and p is a member of the set of places associated to θ, then K provides the information that for all markings obtained immediately after the firing of t, place p will have one token. Thus we do not need to add arcs from and to this place.

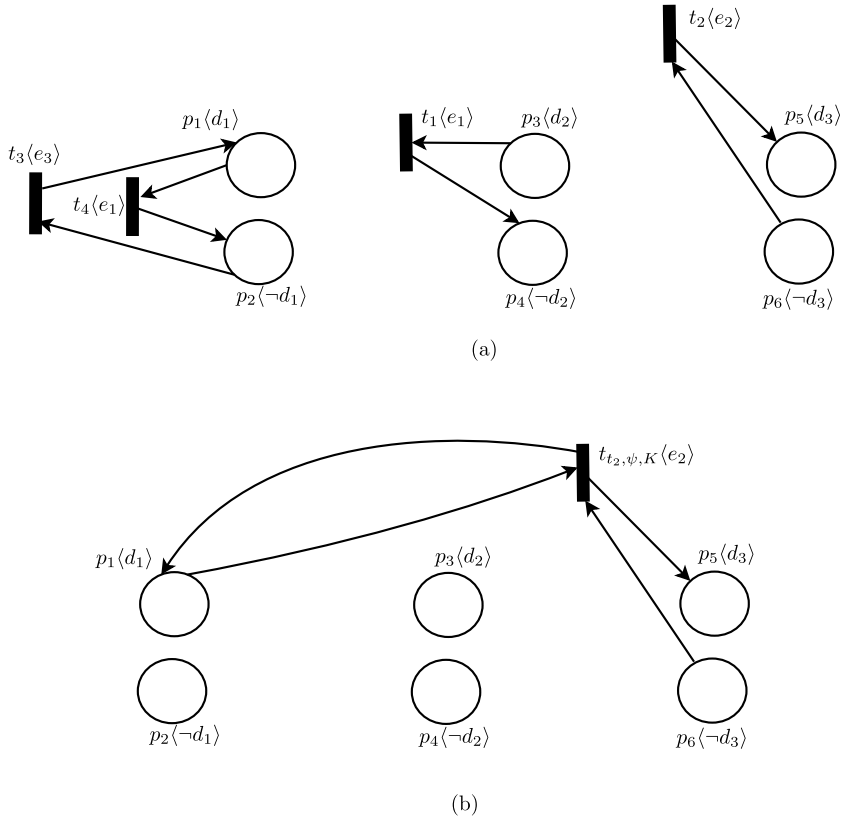


Figure 4.5: Illustration of the minimal satisfying transition construction using a knowledge base. (a) A fragment of a PN. (b) The minimal satisfying transitions built for t_1, t_2, t_3 and t_4 and the information given by K.

Example 4.2.4 (Building Minimal Satisfying Transitions with Knowledge Base)**.** Consider again the fragment of a PN system model depicted in Figure 4.5 (a) and the formula $\psi = d_1 \wedge \neg d_2 \wedge \neg e_3$. We also add a knowledge base $K = \{(\neg e_1 \wedge d_3, \neg d_2), (e_1, \neg d_1)\}$. This has an impact on the construction of the minimal satisfying transitions. For t_1, note that $\varphi_{t_1} = e_1 \wedge \neg e_2 \wedge \neg e_3 \wedge \neg d_2 \vDash e_1$, $(e_1, \neg d_1) \in K$ and $\neg d_1 \vDash \neg d_1 \vee d_2 \vee e_3 = \neg\psi$, hence the minimal satisfying transition is not defined. This reasoning is basically using the additional information provided by K to conclude that ψ can never be satisfied after the firing of t_1, thus a minimal satisfying transition built from t_1 can not be built. For t_2, $\varphi_{t_2} = e_2 \wedge \neg e_1 \wedge \neg e_3 \wedge d_3 \vDash \neg e_1 \wedge d_3$, $(\neg e_1 \wedge d_3, \neg d_2) \in K$ and $p_4 \in P_{\neg d_2}$, thus the arcs from and to place p_4 do not need to be added in this case, because K gives us the information that immediately after t_2 fires, place p_4 is guaranteed to have one token. The result of building the minimal satisfying transitions for t_1, t_2, t_3 and t_4 is depicted in Figure 4.5 (b), taking into account that the minimal satisfying transition is only defined for t_2. One can also see the impact of adding a knowledge base, by comparing Figure 4.3 (b) and Figure 4.5 (b).

We now prove that the minimal satisfying transition for a PN system model with a knowledge

base has the same properties as the minimal satisfying transition built without a knowledge base.

Proposition 4.2.4. *Let G be a PN system model, $\psi = \psi_D \wedge \psi_E$ be a conjunctive clause written over $D \cup E$ and K be a knowledge base over $D \cup E$. The minimal satisfying transition $t_{t,\psi,K}$ has the following properties:*

1. *For all $M \in R(G)$, $M \overset{t_{t,\psi,K}}{\rightarrow} M'$ if and only if $M \overset{t}{\rightarrow} M'$ and $\{\ell(t)\} \cup D_{M'} \Vdash \psi$.*
2. *If $t_{t,\psi,K}$ is undefined, then for all $M' \in R(G)$ for which there exists $M \in R(G)$ such that $M \overset{t}{\rightarrow} M'$, $\{\ell(t)\} \cup D_{M'} \nVdash \psi$.*

Proof. We start by noting that the definition of minimal satisfying transition with a knowledge base K just adds another case – related to K– where the pre and postsets coincide with the original transition. This means that the set of places for which the original transition and the minimal satisfying transition built without using K coincide is contained in the set of places for which the original transition and the minimal satisfying transition built using K coincide. Thus, the part of the proof related to $t_{t,\psi,K}$ being active only when t is active, and the firing of these transitions from the same marking yielding the same next marking is analogous to the proof of Proposition 4.2.1. Thus, we just need to prove that:

1. If $M \overset{t_{t,\psi,K}}{\rightarrow} M'$ then $\{\ell(t)\} \cup D_{M'} \Vdash \psi$, for which it is enough to prove that $D_{M'} \Vdash \psi_D$. Furthermore, we have proved this result for the cases where $p \notin P_{\psi_D}$ and $t^\bullet(p) = 1$ when proving proposition 4.2.1. Hence, let $p \in P_{\psi_D}$ be such that exists (γ, θ) such that $\varphi_t \vDash \gamma$ and $p \in P_\theta$. We want to prove that $M'(p) = 1$, thus proving that the literals in the conjunctive clause ψ_D are satisfied in M'. By lemma 4.2.1, we have $\{\ell(t)\} \cup D_{M'} \Vdash \varphi_t$, thus $\{\ell(t)\} \cup D_{M'} \Vdash \gamma$, because $\varphi_t \vDash \gamma$. Given the definition of knowledge base, we can conclude from $\{\ell(t)\} \cup D_{M'} \Vdash \gamma$ that $D_{M'} \Vdash \theta$. Thus, $M'(p) = 1$ for all paces in P_θ. We can then conclude that $\{\ell(t)\} \cup D_{M'} \Vdash \psi$.
2. If exists $(\gamma, \theta) \in K$ such that $\varphi_t \vDash \gamma$ and $\theta \vDash \neg\psi$ then $\{\ell(t)\} \cup D_{M'} \nVdash \psi$. Then $lit^+_\psi \cap lit^-_\theta \neq \emptyset$ or $lit^-_\psi \cap lit^+_\theta$. Assuming that $lit^+_\psi \cap lit^-_\theta \neq \emptyset$, let $d \in lit^+_\psi \cap lit^-_\theta$. By the same reasoning as above, from $d \in lit^-_\theta$ we can conclude that $M'(\mu(\neg d)) = 1$. Thus, $M'(\mu(d)) = 0$, i.e., d is a literal in ψ that is not satisfied in M'. Thus $D_{M'} \nVdash \psi$. The reasoning for the case where $lit^-_\psi \cap lit^+_\theta \neq \emptyset$ is analogous.

□

Algebraic State Description

We now proceed to show the construction of minimal satisfying transitions for restrictions of the form $\psi = \psi_E \wedge \psi_P$, where ψ_E is a conjunctive clause over E and ψ_P is a conjunctive clause written over $P_b \times \mathbb{N}$ where all the literals are positive, i.e., ψ_P is of the form:

$$\bigwedge_{i=1}^{n} M(p_i) \geq b_i,\ p_i \in P_b,\ b_i \in \mathbb{N} \tag{4.24}$$

We assume that all the p_i's are different because if there are two constraints over the same place p, we just need to take into account the constraint with greater b. Before showing how to build minimal satisfying transitions for this type of constraint, we will present a procedure – based in the relation between places and their complement places, and in the addition of counter places – to reduce to the form $M(p) \geq b$ (or a conjunction or disjunction of 2 constraints of the form $M(p) \geq b$, for the case of equal or not equal constraints, respectively) any constraint of the form:

$$\begin{gathered} \sum_{p \in P_b} v(p)M(p) \lessgtr b,\ v(p) \in \mathbb{Z},\ b \in \mathbb{N},\ \lessgtr \in \{<, \leq, =, \neq, \geq, >\} \\ \Leftrightarrow \\ v^T M \lessgtr b,\ v \in \mathbb{Z}^{|P|},\ b \in \mathbb{N},\ \lessgtr \in \{<, \leq, =, \neq, \geq, >\},\ v(p) = 0 \text{ for all } p \in P_\omega \end{gathered} \tag{4.25}$$

Thus, we can assume that the atomic propositions are of the form (4.25). These constraints provide the designer with a more expressive specification language than the symbolic approach presented before, and we will also show how one can translate symbolic specifications to algebraic specifications which can even be more compact.

Firstly, we can assume that the conjunctive clauses do not contain negative literals, because the negation of a linear constraint can always be transformed in an equivalent "positive" linear constraint. This is stated in the following proposition, which is a direct consequence of v and M being vectors of integers and b being an integer.

Proposition 4.2.5. *Let $v, \in \mathbb{Z}^{|P|}$, and $b \in \mathbb{N}$. The following statements hold:*

- $\neg(v^T M < b)$ *if and only if* $v^T M \geq b$, *and* $\neg(v^T M \geq b)$ *if and only if* $v^T M < b$;
- $\neg(v^T M \leq b)$ *if and only if* $v^T M > b$, *and* $\neg(v^T M > b)$ *if and only if* $v^T M \leq b$;
- $\neg(v^T M = b)$ *if and only if* $v^T M \neq b$, *and* $\neg(v^T M \neq b)$ *if and only if* $v^T M = b$.

We now show that, because b is a natural number, any linear constraint over the markings can be easily re-written as a greater-or-equal (GEQ) constraint, a less-or-equal (LEQ) constraint or the conjunction or disjunction of a LEQ and a GEQ constraint.

Proposition 4.2.6. *Let $l \in \mathbb{Z}^{|P|}$, $M \in \mathbb{N}^{|P|}$ and $b \in \mathbb{N}$. Then:*

- $v^T M > b$ *if and only if* $v^T M \geq b + 1$;
- $v^T M < b$ *if and only if* $v^T M \leq b - 1$;
- $v^T M = b$ *if and only if* $v^T M \leq b \wedge v^T M \geq b$;
- $v^T M \neq b$ *if and only if* $v^T M \leq b - 1 \vee v^T M \geq b + 1$

Thus, we can assume that our linear constraints are of the form:

$$\begin{gathered}\sum_{p\in P_b} v(p)M(p) \lesseqgtr b,\ v(p)\in\mathbb{Z},\ b\in\mathbb{N},\ \lesseqgtr \in \{\leq,\geq\} \\ \Leftrightarrow \\ v^T M \lesseqgtr b,\ v\in\mathbb{Z}^{|P|},\ b\in\mathbb{N},\ \lesseqgtr\in\{\leq,\geq\},\ v(p)=0 \text{ for all } p\in P_\omega\end{gathered} \tag{4.26}$$

This form still does not allow us to build minimal satisfying transitions. For that, we will need, for each conjunctive clause $v^T M \lesseqgtr b$, to add a place p' to the PN such that, for all reachable markings M, $M(p') = v^T M$. Furthermore, we want to re-write the constraint as a GEQ constraint so we can use the minimal satisfying transition construction. Hence, our goal is to re-write constraints of the type (4.26) into constraints of the form:

$$\begin{gathered}\sum_{p\in P_b} v(p)M(p) \geq b,\ v(p)\in\mathbb{N},\ b\in\mathbb{N} \\ \Leftrightarrow \\ v^T M \geq b,\ v\in\mathbb{N}^{|P|},\ b\in\mathbb{N},\ v(p)=0 \text{ for all } p\in P_\omega\end{gathered} \tag{4.27}$$

To do this, we apply Algorithm 2 which, in words, uses the relation between the marking in a place and the marking in its complement place to apply the following sequence of modifications on the constraint:

1. If the input constraint is a LEQ constraint, turn it into a GEQ constraint;
2. If the constraint is referring to both a place p and its complement $\overline{p}$, write it only over p, making the coefficient associated to $\overline{p}$ equal to zero;
3. If the coefficient associated to a given place in negative, turn it into a positive coefficient associated to its complement place.

We proceed to prove the algorithm correctness.

Proposition 4.2.7. *Let G be a PN system model , $v\in\mathbb{Z}^{|P|}$ such that $v(p)=0$ for all $p\in P_\omega$, $b\in\mathbb{N}$ and $\lesseqgtr\in\{\leq,\geq\}$. Let $v'\in\mathbb{N}^{|P|}$ and $b'\in\mathbb{Z}$ be the outputs of applying Algorithm 2 to inputs G, v, b and $\lesseqgtr$. Then, for all $M\in R(G)$, $v^T M \lesseqgtr b$ if and only if $v'^T M \geq b'$, and:*

- *If the algorithm does not output a warning, $b\in\mathbb{N}$.*
- *If the algorithm outputs WARNING 1, then all reachable markings of the PN satisfy the constraint.*
- *If the algorithm outputs WARNING 2, then no reachable marking of the PN satisfies the constraint.*

Proof. From the last loop, in lines 14–20, it is clear that the output $v'\in\mathbb{N}^{|P|}$. Hence, we just need to show that each assignment that the algorithm can do to v' and b' yields equivalent constraints. For

Algorithm 2 Convert constraints into positive GEQ constraints

Input: PN system model G, $v \in \mathbb{Z}^{|P|}$ such that $v(p) = 0$ for all $p \in P_\omega$, $b \in \mathbb{N}$ and $\lesseqgtr \in \{\leq, \geq\}$.

Output: $v' \in \mathbb{N}^{|P|}$ and $b' \in \mathbb{N}$ such that, for all $M \in R(G)$, $v'^T M \geq b'$ if and only if $v^T M \lesseqgtr b$.

$v' \leftarrow v$
$b' \leftarrow b$
if $\lesseqgtr = \leq$ **then**
  $b' \leftarrow -b' + \sum_{p \in P_b} v'(p)\kappa(p)$
  $v' \leftarrow v''$, where, for $p \in P_b$, $v''(p) = v'(\overline{p})$
end if
for all $p \in P_b$ **do**
  if $v'(p) \neq 0$ and $v'(\overline{p}) \neq 0$ **then**
    $b' \leftarrow b - v'(\overline{p})\kappa(p)$
    $v'(p) \leftarrow v'(p) - v'(\overline{p})$
    $v'(\overline{p}) \leftarrow 0$
  end if
end for
for all $p \in P_b$ **do**
  if $v'(p) < 0$ **then**
    $b' \leftarrow b' - v'(p)\kappa(p)$
    $v'(\overline{p}) \leftarrow -v'(p)$
    $v'(p) \leftarrow 0$
  end if
end for
if $b' \leq 0$ **then**
  WARNING 1: The constraint is satisfied in all markings. {This warning means that the constraint is equivalent to the propositional formula $true$}
end if
if $b' > \sum_{p \in P_b} l'(p)\kappa(p)$ **then**
  WARNING 2: The constraint is unsatisfiable. {This warning means that the constraint is equivalent to the propositional formula $false$}
end if

this, let v_{old} and b_{old} be the values of v' and b' before an assignment and v_{new} and b_{new} the values of v' and b' after that assignment.

1. For the assignment in lines 4 and 5, we have the following:

$$\begin{aligned}
\sum_{p\in P_b} v_{old}(p)M(p) \leq b_{old} &\Leftrightarrow \sum_{p\in P_b} -v_{old}(p)M(p) \geq -b_{old} \\
&\Leftrightarrow \sum_{p\in P_b} -v_{old}(p)(\kappa(p) - M(\overline{p})) \geq -b_{old} \\
&\Leftrightarrow \sum_{p\in P_b} v_{old}(p)M(\overline{p}) \geq -b_{old} + \sum_{p\in P_b} v_{old}(p)\kappa(p) \\
&\Leftrightarrow \sum_{p\in P_b} v_{old}(\overline{p})M(p) \geq b_{new} \\
&\Leftrightarrow \sum_{p\in P_b} v_{new}(p)M(p) \geq b_{new}
\end{aligned}$$

2. For the assignment in lines 9–11, let $p' \in P_b$ such that $v'(p') \neq 0$ and $v'(\overline{p'}) \neq 0$. We have the following:

$$\begin{aligned}
\sum_{p\in P_b} v_{old}(p)M(p) \geq b_{old} &\Leftrightarrow \left(\sum_{\substack{p\in P_b \\ p\neq\overline{p'}}} v_{old}(p)M(p)\right) + v_{old}(\overline{p'})(\kappa(p') - M(p')) \geq b_{old} \\
&\Leftrightarrow \left(\sum_{\substack{p\in P_b \\ p\neq\overline{p'}}} v_{old}(p)M(p)\right) - v_{old}(\overline{p'})M(p') \geq b_{old} - v_{old}(\overline{p'})\kappa(p') \\
&\Leftrightarrow \sum_{p\in P_b} v_{new}(p)M(p) \geq b_{new}
\end{aligned}$$

   It is clear that, after this loop that contains this assignment is completed, all $p \in P_b$ is such that $v'(p) \neq 0$ implies $v'(\overline{p}) = 0$.

3. For the assignment in lines 16–18, and assuming that $v'(p) \neq 0$ implies $v'(\overline{p}) = 0$ for all $p \in P_b$ (this assumption is valid because the previous loop corresponding to the assignment in lines

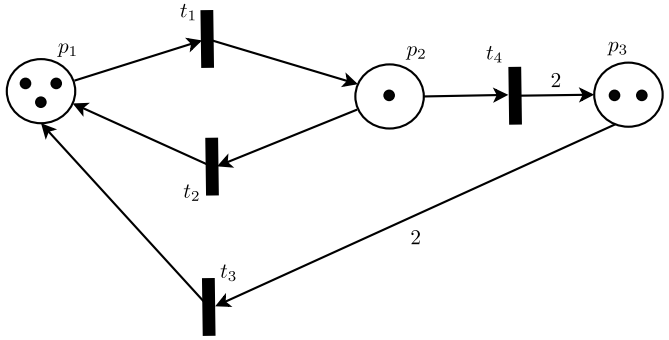


Figure 4.6: Petri net for illustrating the re-writing of linear constraints.

9–11 is completed), let $p' \in P_b$ such that $v'(p') < 0$. We have the following:

$$\begin{aligned}
\sum_{p \in P_b} v_{old}(p)M(p) \geq b_{old} &\Leftrightarrow \left(\sum_{\substack{p \in P_b \\ p \notin \{p', \overline{p'}\}}} v_{old}(p)M(p) \right) + v_{old}(p')(\kappa(p') - M(\overline{p'})) \geq b_{old} \\
&\Leftrightarrow \left(\sum_{\substack{p \in P_b \\ p \notin \{p', \overline{p'}\}}} v_{old}(p)M(p) \right) - v_{old}(p')M(\overline{p'}) \geq b_{old} - v_{old}(p')\kappa(p') \\
&\Leftrightarrow \sum_{p \in P_b} v_{new}(p)M(p) \geq b_{new}
\end{aligned}$$

Note that we can assume that $p \notin \{p', \overline{p'}\}$ in the sum, because we isolate p' outside the sum and $v'(\overline{p'})$ is guaranteed to be 0.

Hence, since all the assignments made to v' and b' yield an equivalent constraint, we proved that the output constraint $v'^T M \geq b'$ is equivalent to the input constraint $v^T M \leqq b$. Furthermore, given that (i) $v' \in \mathbb{N}^{|P|}$, (ii) by definition, $M \in \mathbb{N}^{|P|}$, and (iii) the if-statements in lines 21 and 24, it is obvious that the statements regarding the the warnings are also true because $v'^T M \geq 0$ for all $M \in R(G)$ and existing a marking $M \in R(G)$ such that $v'^T M > v'^T \kappa$ contradicts that fact that $\kappa(p)$ is a bound for the number of tokens in place p.

□

Example 4.2.5 (Re-writing the linear constraints)**.** Consider the PN depicted in Figure 4.6, where $\kappa(p_1) = \kappa(p_2) = 5$ and $\kappa(p_3) = 10$. We do not depict $\overline{p_1}$, $\overline{p_2}$ and $\overline{p_3}$, but assume that they are built using the complement place construction described in Definition 2.2.22. Given κ, we can conclude that $M_0(\overline{p_1}) = 2$, $M_0(\overline{p_2}) = 4$ and $M_0(\overline{p_3}) = 8$. The markings of this PN are of the form:

$$M = \begin{bmatrix} M(p_1) & M(p_2) & M(p_3) & M(\overline{p_1}) & M(\overline{p_2}) & M(\overline{p_3}) \end{bmatrix}^T \tag{4.28}$$

We give three examples on how to re-write the linear constraints so that they are of the form $v^T M \geq b$.

1. Consider the constraint $\begin{bmatrix}0 & 2 & 1 & 0 & 1 & 0\end{bmatrix} M \geq 8$:

$$\begin{aligned} 2M(p_2)+M(p_3)+M(\overline{p_2}) \geq 3 &\Leftrightarrow 2M(p_2)+M(p_3)+\kappa(p_2)-M(p_2) \geq 8 \\ &\Leftrightarrow M(p_2)+M(p_3) \geq 3 \end{aligned}$$

   Thus, the constraint can be re-written to $\begin{bmatrix}0 & 1 & 1 & 0 & 0 & 0\end{bmatrix} M \geq 3$

2. Consider the constraint $\begin{bmatrix}1 & 0 & 1 & 0 & 0 & 0\end{bmatrix} M \leq 6$:

$$\begin{aligned} M(p_1)+M(p_3) \leq 6 &\Leftrightarrow -M(p_1)-M(p_3) \geq -6 \\ &\Leftrightarrow -(\kappa(p_1)-M(\overline{p_1}))-(\kappa(p_3)-M(\overline{p_3})) \geq -6 \\ &\Leftrightarrow M(\overline{p_1})+M(\overline{p_3}) \geq 9 \end{aligned}$$

   Thus, the constraint can be re-written to $\begin{bmatrix}0 & 0 & 0 & 1 & 0 & 1\end{bmatrix} M \geq 9$

3. Consider the constraint $\begin{bmatrix}3 & 0 & -2 & 0 & 0 & 0\end{bmatrix} M \geq 4$:

$$\begin{aligned} 3M(p_1)-2M(p_3) \geq 4 &\Leftrightarrow 3M(p_1)-2(\kappa(p_3)-M(\overline{p_3})) \geq 4 \\ &\Leftrightarrow 3M(p_1)+2M(\overline{p_3}) \geq 24 \end{aligned}$$

   Thus, the constraint can be re-written to $\begin{bmatrix}3 & 0 & 0 & 0 & 0 & 2\end{bmatrix} M \geq 24$

We can now assume that we are dealing with constraints $v^T M \geq b$, where $v^T \in \mathbb{N}$ and $b \in \mathbb{N}$. Hence, if we find a way to reduce constraints of the form $v^T M \geq b$ to a single place constraint $M(p) \geq b$, we will be able to reduce any linear constraint to the form (4.24). For this, we introduce the notion of counter place for a given linear expression.

Definition 4.2.8 (Counter Place)**.** Let $G = \langle P, T, M_0, W^-, W^+, E, \ell, \kappa, ^- \rangle$ be a PN system model and and $v \in \mathbb{N}^{|P_b|}$. We add the counter place for v, denoted p_v, to G, yielding the PN $G^v = \langle P \cup \{p_v\}, T, M_0^v, W^{v-}, W^{v+}, E, \ell, \kappa^v, ^- \rangle$, where, for p_v:

- $M_0^v(p_v) = v^T M_0 = \sum_{p \in P_b} v(p) M_0(p)$
- $W^v(p_v, .) = v^T W$
- $\kappa^v(p_v) = v^T \kappa$

$W^v(p_v, .)$ represents the line added to incidence matrix $W = W^+ - W^-$, corresponding to p_v.

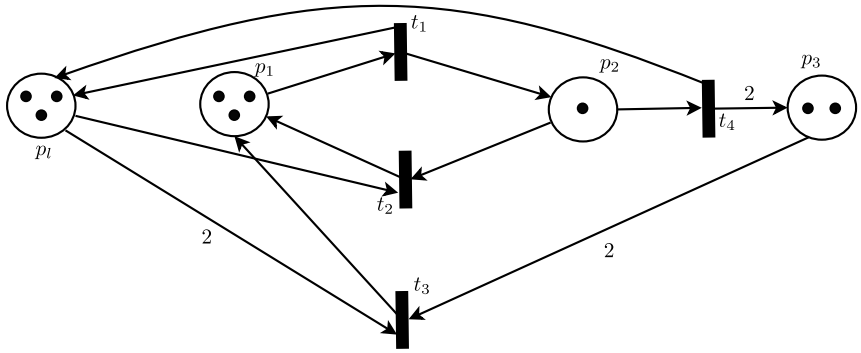


Figure 4.7: Illustration of the counter place construction.

Since p_v is used to count linear combinations of the number of tokens in other places, it does not contain self-loops. Hence, $W^v(p_v,t)$ uniquely defines $W^{v-}(p_v,t)$ and $W^{v+}(p_v,t)$ for all $t \in T$, as seen in equations (2.5) and (2.6).

Example 4.2.6 (Adding a Counter Place)**.** Consider again the PN depicted in Figure 4.6. The incidence matrix of the PN (including the complement places) is given by:

$$W = \begin{bmatrix} -1 & 1 & 1 & 0 \\ 1 & -1 & 0 & -1 \\ 0 & 0 & -2 & 2 \\ 1 & -1 & -1 & 0 \\ -1 & 1 & 0 & 1 \\ 0 & 0 & 2 & -2 \end{bmatrix} \tag{4.29}$$

We exemplify the construction of the counter place for the linear expressions of the previous example.

1. For $v^T = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$:
   - $M_0(p_v) = M_0(p_2) + M_0(p_3) = 3$
   - $W^v(p_v,.) = v^T W = \begin{bmatrix} 1 & -1 & -2 & 1 \end{bmatrix}$. Note that, since there are no self-loops, this is equivalent to $W^{v-}(p,.) = \begin{bmatrix} 0 & 1 & 2 & 0 \end{bmatrix}$ and $W^{v+}(p,.) = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}$.
   - $\kappa(p_v) = \kappa(p_2) + \kappa(p_3) = 15$

   We depict the addition of this counter place in Figure 4.7.

2. For $v^T = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$:
   - $M_0(p_v) = M_0(p_1) + M_0(p_3) = 5$
   - $W^v(p_v,.) = v^T W = \begin{bmatrix} -1 & 1 & -1 & 2 \end{bmatrix}$

- $\kappa(p_v) = \kappa(p_1) + \kappa(p_3) = 15$

3. For $v^T = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$:

   - $M_0(p_v) = 3M_0(p_1) + 2M_0(\overline{p_3}) = 9 + 16 = 25$
   - $W^v(p_v, .) = v^T W = \begin{bmatrix} -3 & 3 & 7 & -4 \end{bmatrix}$
   - $\kappa(p_v) = 3\kappa(p_1) + 2\kappa(\overline{p_3}) = 35$

As with the complement place addition, the PN G^v obtained by adding a counter place is well-defined, in the sense that the number of tokens in p_v in a given marking M is always equal to $v^T M$ and the behaviour of G^v is unaltered, as stated in the following proposition.

Proposition 4.2.8. *Let G be a PN system model, $v \in \mathbb{N}^{|P_b|}$, G^v the PN system model obtained by adding the counter place associated with v and $\tau \in T^*$. Then, $M_0 \xrightarrow{\tau} M$ if and only if $M_0^v \xrightarrow{\tau} M^v$, where:*

$$M^v(p) = \begin{cases} M(p) & \text{if } p \in P \\ \sum_{p \in P_b} v(p) M(p) & \text{if } p = p_v \end{cases} \tag{4.30}$$

Proof. Note that it is clear that if $M_0^v \xrightarrow{\tau} M^v$, then $M_0 \xrightarrow{\tau} M$. This is true because, by adding a place, we can only restrict the behaviour of the original PN. Furthermore, the initial markings in G and G^v coincide for $p \neq p_v$, as do the pre and postsets. Thus, we just need to show that, given $\tau \in T^*$, if $M_0 \xrightarrow{\tau} M$, then $M_0^v \xrightarrow{\tau} M^v$. We will show that $M_0^v \xrightarrow{\tau} M^v$ by induction on the length of τ.

1. $|\tau| = 0$, i.e., $\tau = \varepsilon$: In this case we need to analyse the initial marking M_0^v, which satisfies the requirement by definition.

2. $|\tau| = n + 1$, i.e., $\tau = \tau' t$, $\tau' \in T^*$, $t \in T$. By hypothesis, we have that:

   $$M_{\tau'}^v(p) = \begin{cases} M_{\tau'}(p) & \text{if } p \in P \\ \sum_{p \in P_b} v(p) M_{\tau'}(p) & \text{if } p = p_v \end{cases} \tag{4.31}$$

   Hence, we need to prove that:

   (i) If t is active in $M_{\tau'}$ for G, then it is also active for $M_{\tau'}^v$ for G^v, i.e., if $(M_{\tau'} \xrightarrow{t})$, then $(M_{\tau'}^v \xrightarrow{t})$. We assume that it is true that $(M_{\tau'} \xrightarrow{t})$, but not $(M_{\tau'}^v \xrightarrow{t})$. Hence, we have $W^{v-}(p,t) = W^-(p,t) \leq M_{\tau'}(p)$ for all $p \in P$. Thus, t not being active in $M_{\tau'}^v$ must be due

to place p_v, i.e., $W^{v-}(p_v,t) > M^v_{\tau'}(p_v) \geq 0$. Hence, we have the following:

$$\begin{aligned}
\sum_{p\in P_b} v(p)M_{\tau'}(p) &= M^v_{\tau'}(p_v) < W^{v-}(p_v,t) \\
&= -W^v(p_v,t) = -v^T W(.,t) \\
&= -\sum_{p\in P_b} v(p)W(p,t) \\
&= -\sum_{p\in P_b} v(p)(W^+(p,t) - W^-(p,t)) \\
&= \sum_{p\in P_b} v(p)(W^-(p,t) - W^+(p,t)) \\
&\leq \sum_{p\in P_b} v(p)W^-(p,t)
\end{aligned}$$

Thus, there must exist $p \in P_b$ such that $M_{\tau'}(p) < W^-(p,t)$, which contradicts the assumption that t is active in $M_{\tau'}$.

(ii) The marking obtained by firing t in $M^v_{\tau'}$ satisfies the equality stated in the proposition. For $p \in P$, the equality is obvious because W^v and W coincide for these places. For $p = p_v$, we have:

$$\begin{aligned}
M^v_{\tau' t}(p_v) &= M^v_{\tau'}(p_v) + W^{l+}(p_v,t) - W^{l-}(p_v,t) \\
&= \sum_{p\in P_b} v(p)M_{\tau'}(p) + W^{l+}(p_v,t) - W^{l-}(p_v,t) \\
&= \sum_{p\in P_b} v(p)M_{\tau'}(p) + W^v(p_v,t) \\
&= \sum_{p\in P_b} v(p)M_{\tau'}(p) + \sum_{p\in P_b} v(p)W(p,t) \\
&= \sum_{p\in P_b} v(p)\left(M_{\tau'}(p) + W(p,t)\right) \\
&= \sum_{p\in P_b} v(p)\left(M_{\tau'}(p) + W^+(p,t) - W^-(p,t)\right) \\
&= \sum_{p\in P_b} v(p)M_{\tau' t}(p)
\end{aligned}$$

Hence the proof is completed.

□

Corollary 4.2.1. *Let G be a PN system model and $v \in \mathbb{N}^{|P_b|}$. Then $\mathscr{L}(G) = \mathscr{L}(G^v)$.*

To summarize, we present the procedure that should be followed by the designer, in order to have the BA transitions labels in such a way that they can be composed with the PN transitions using the

notion of minimal satisfying transition we will present next. To be able to use the minimal satisfying transitions for the composition, the BA transition labels need to be formulas in the DNF such that each conjunctive clause is of the form (4.24). To achieve this, and given an LTL formula written over a set of atomic propositions of the form (4.25), i.e., an LTL formula φ written with the most general type of linear constraints, the following steps need to be followed for each DNF label in the BA obtained from φ:

1. For each negative literal not written over E, convert it into a positive literal by using the identities in Proposition 4.2.5.

2. For each (positive) literal not written over E, if it does not represent a LEQ or a GEQ constraint, convert it into one by using the identities in Proposition 4.2.6.

3. After applying the previous step, the formula might not be in the DNF, because literals representing "not equal" constraints are replaced by a disjunction of a LEQ and a GEQ constraint. Rewrite the formula in the DNF[3].

4. For each (positive) literal not written over E – which is now in form (4.26) – apply Algorithm 2 to turn it into a positive GEQ constraint.

5. For each (positive) literal not written over E – which is now in form (4.27) – build the corresponding counter place p_v using Definition 4.2.8, and turn it into a constraint of the form $M(p_v) \geq b$.

Thus, we will write linear constraints of form (4.25) in our LTL specifications, assuming that the PN system model and the BA transition labels are altered before applying the composition algorithm, by applying the steps presented above.

Remark 4.2.1. Before moving to the construction of the minimal satisfying transitions for this case, we present an important remark about the impact of the use of linear constraints on the size of the specifications. Consider a PN system model with symbolic state description. We already showed that this system model can be translated to a PN system model with algebraic state description. Furthermore, we can also re-write some propositional formulas written over D, by taking the following into account:

- A formula $\bigvee_{i=1}^{n} l_i$, with $l_i \in lit(D)$, can be represented as $\sum_{i=1}^{n} M(\mu(l_i)) \geq 1$
- A formula, $\bigwedge_{i=1}^{n} l_i$, with $l_i \in lit(D)$ can be represented as $\sum_{i=1}^{n} M(\mu(l_i)) \geq n$
- More generally, consider the so-called *cardinality constraints*, i.e., given $D' = \{l_1, ..., l_n\} \subseteq lit(D)$, constraints of the form "at least m of the n literals in D' are satisfied in marking M" or

[3]Note that, as already stated, this rewriting is always possible, by using the distribution and De Morgan's laws for propositional logic. In fact, given that we already turned all literals into positive literals, one only needs to use the distribution laws.

"at most m of the n literals in D' are satisfied in marking M". There has been a great deal of study in recent literature on how to codify this type of constraint in propositional logic and, to the best of our knowledge, the best lower-bound obtained for the length of a propositional logic codification is $O(n\log^2 m)$ [Asín et al., 2011]. With the algebraic state space representation, "at least" constraints can be represented by $\sum_{i=1}^{n} M(\mu(l_i)) \geq m$ and "at most" constraints can be represented by $\sum_{i=1}^{n} M(\mu(\neg l_i)) \geq n-m$, i.e., they can be represented by a single atomic proposition.

This reduction on the size of the specifications has an impact on the size of the supervisors, as we will see in Section 6.1. Furthermore, this new algebraic semantics can also be used for LTL based PN model checking and its reduction of the size of the formulas might allow one to speed up the model checking procedures. This can be subject of future work.

Now that we know how to reduce the BA transition labels written over general linear constraints into transition labels where the conjunctive clauses are of the form (4.24), we can proceed and show how to build a minimal satisfying transition for a conjunctive clause of that form. With this goal in mind, we define the set of places associated with $\psi = \psi_E \wedge \psi_P$, which are simply the places for which there is a constraint in ψ_P.

Definition 4.2.9. Let $G = \langle L, \kappa, {}^{-}\rangle$ be an algebraic PN system model, and $\psi = \psi_E \wedge \psi_P$ be a conjunctive clause written over $E \cup (P_b \times \mathbb{N})$. We define the set of places associated to ψ as:

$$P_\psi = \bigcup_{i=1}^{n} p_i \tag{4.32}$$

Note that $P_\psi \subseteq P_b$, i.e., the linear constraints are only written over places for which a bound exists and is defined. We also define a function $b : P_\psi \to \mathbb{N}$ that maps each p_i to the correspondent b_i, i.e., $b(p_i) = b_i$.

We can now define the notion of minimal satisfying transition.

Definition 4.2.10 (Minimal Satisfying Transition)**.** Let G be a PN system model, $t \in T$, and $\psi = \psi_E \wedge \psi_P$. If $\kappa(p) - {}^\bullet t(p) + t^\bullet(p) \geq b(p)$ for all $p \in P_\psi$ and $\{\ell(t)\} \Vdash \psi_E$, the minimal satisfying transition $t_{t,\psi}$ obtained from t, and ψ is defined as:

$${}^\bullet t_{t,\psi}(p) = \begin{cases} {}^\bullet t(p) & \text{if } p \notin P_\psi \text{ or } b(p) \leq t^\bullet(p) \\ b(p) + {}^\bullet t(p) - t^\bullet(p) & \text{if } p \in P_\psi \text{ and } b(p) > t^\bullet(p) \end{cases} \tag{4.33}$$

$$t^\bullet_{t,\psi}(p) = \begin{cases} t^\bullet(p) & \text{if } p \notin P_\psi \text{ or } b(p) \leq t^\bullet(p) \\ b(p) & \text{if } p \in P_\psi \text{ and } b(p) > t^\bullet(p) \end{cases} \tag{4.34}$$

In the cases where exists $p \in P_\psi$ such that $\kappa(p) - {}^\bullet t(p) + t^\bullet(p) < b(p)$ or $\{\ell(t)\} \nVdash \psi_E$, we say that the minimal satisfying transition is undefined.

This definition is a generalization of the definition of minimal satisfying transition for PN system models with symbolic description, but taking into account that now we can reason about more than the number of tokens in a place being 1 or 0. To do this, we need to use the bound function κ, thus these specifications can only be build for places for which a bound is known. Also, for ψ_E it is equivalent to the definition for PN system models with symbolic state description.

Example 4.2.7 (Building Minimal Satisfying Transitions)**.** Figure 4.8 (a) depicts a fragment of a PN, and Figure 4.8 (b) illustrates the construction of minimal satisfying transitions for the conjunctive clause:

$$\psi = (M(p_1) \geq 3) \wedge (M(p_2) \geq 1), \text{ where } \kappa(p_1) = 4 \text{ and } \kappa(p_2) = 1 \tag{4.35}$$

In this example, since $\kappa(p_1) - {}^\bullet t_2(p_1) + t_2^\bullet(p_1) = 4 - 2 + 0 = 2 < b(p_1) = 3$, the maximum amount of tokens that can be in p_1 immediately after the firing of t_2 is 2. Thus, given that our goal is to keep the number of tokens in p_1 greater or equal than 3, the minimal satisfying transition for t_2 is undefined. Since the definition of minimal satisfying transition for ψ_E is analogous to the symbolic case, we did not add events to the formula used for this example to facilitate its understanding.

The minimal satisfying transition for t and ψ is a transition that is active if and only if t is active and the firing of t drives the PN to a marking that satisfies all the linear constraints $M(p) \geq b(p)$, $p \in P_\psi$. Furthermore, when the minimal satisfying transition is not defined, then all the firings of t will drive the PN to a marking where there exists at least one $p \in P_\psi$ such that $M(p) \geq b(p)$ is not satisfied, i.e., $M(p) < b(p)$. We prove this in the following proposition.

Proposition 4.2.9. *Let G be a PN system model and $\psi = \psi_E \wedge \psi_P$ be a conjunctive clause written over E and ψ_P is written over $E \cup (P \times \mathbb{N})$. The minimal transition $t_{t,\psi}$ has the following properties:*

1. *For all $M \in R(G)$, $M \xrightarrow{t_{t,\psi}} M'$ if and only if $M \xrightarrow{t} M'$ and $M'(p) \geq b(p)$ for all $p \in P_\psi$, i.e., $(M, \ell(t)) \Vdash \psi$;*
2. *If $t_{t,\psi}$ is undefined, then for all $M' \in R(G)$ for which there exists $M \in R(G)$ such that $M \xrightarrow{t} M'$, there exists $p \in P_\psi$ such that $M'(p) < b(p)$, i.e., $(M, \ell(t)) \nVdash \psi$.*

Proof. The proof for ψ_E is analogous to the proof for the minimal satisfying transitions for PN system models with symbolic state description, hence we will only prove the proposition for ψ_P. This is a generalization of the proof for ψ_D presented in the symbolic state description case.

1. We start by noting that pre and postsets of t and $t_{t,\psi}$ coincide for all places p such that $p \notin P_\psi$ or $b(p) \leq t^\bullet(p)$. Hence, we only need to analyse places p such that $p \in P_\psi$ and $b(p) > t^\bullet(p)$. We now prove that for all $M \in R(G)$, if $M \xrightarrow{t_{t,\psi}} M'$ then $M \xrightarrow{t} M'$ and $M'(p) \geq b(p)$ for all $p \in P_\psi$. To prove that $M \xrightarrow{t} M'$, we need to prove that:

   (i) $(M \xrightarrow{t})$, i.e., $M \geq {}^\bullet t$. For all $p \in P_\psi$ such that $b(p) > t^\bullet(p)$, we have:

$$M(p) \geq {}^\bullet t_{t,\psi}(p) = b(p) + {}^\bullet t(p) - t^\bullet(p) > {}^\bullet t(p) \tag{4.36}$$

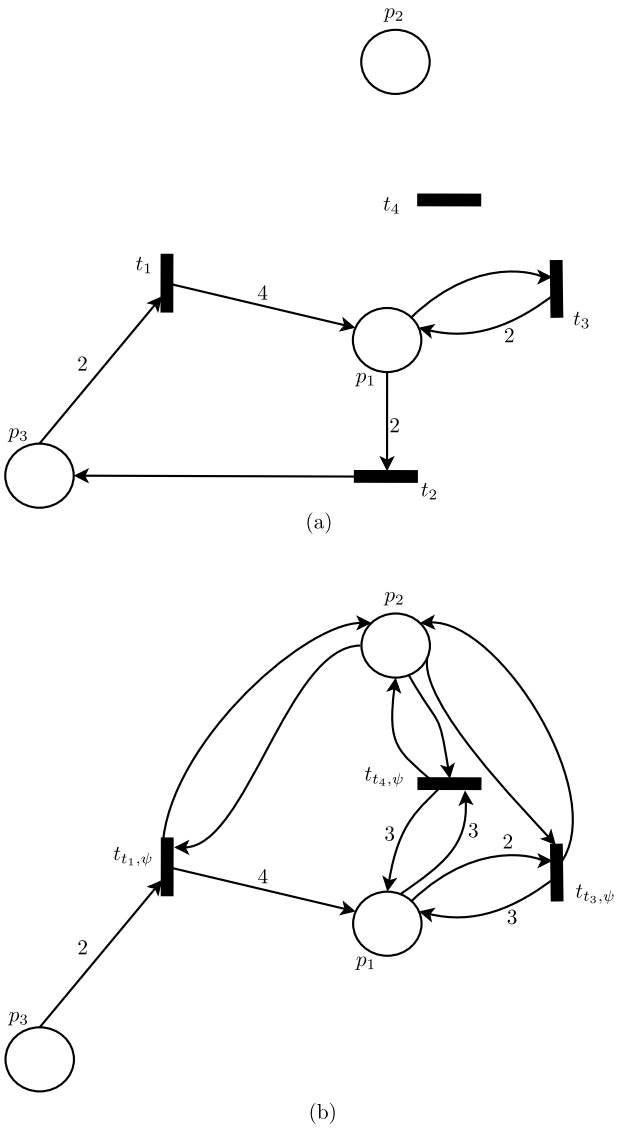


Figure 4.8: Illustration of the minimal satisfying transition construction. (a) A fragment of a PN. (b) The minimal satisfying transitions built for t_1, t_2, t_3 and t_4.

(ii) $M - {}^\bullet t + t^\bullet = M - {}^\bullet t_{t,\psi} + t^\bullet_{t,\psi}$. For all $p \in P_\psi$ such that $b(p) > t^\bullet(p)$, we have:

$$M(p) - {}^\bullet t_{t,\psi}(p) + t^\bullet_{t,\psi}(p) = M(p) - b(p) - {}^\bullet t(p) + t^\bullet(p) + b(p) = M(p) - {}^\bullet t(p) + t^\bullet(p) \tag{4.37}$$

To prove that $M'(p) \geq b(p)$ for all $p \in P_\psi$, we start by noting that, according to the PN firing rule, if $M \xrightarrow{t_{t,\psi}} M'$, then $M'(p) \geq t^\bullet_{t,\psi}(p)$ for all $p \in P$. Furthermore, by construction of $t_{t,\psi}$, $t^\bullet_{t,\psi}(p) \geq b(p)$ for all $p \in P_\psi$. Hence, it is clear that $M'(p) \geq b(p)$ for all $p \in P_\psi$.

We now prove that for all $M \in R(G)$, if $M \xrightarrow{t} M'$ and $M'(p) \geq b(p)$ for all $p \in P_\psi$, then $M \xrightarrow{t_{t,\psi}} M'$.

(i) $(M \xrightarrow{t_{t,\psi}})$, i.e., $M \geq {}^\bullet t_{t,\psi}$. For all $p \in P_\psi$ such that $b(p) > t^\bullet(p)$, we have:

$$\begin{aligned} M'(p) &= M(p) - {}^\bullet t(p) + t^\bullet(p) = M(p) - {}^\bullet t_{t,\psi}(p) + b(p) - t^\bullet(p) + t^\bullet(p) \\ &= M(p) - {}^\bullet t_{t,\psi}(p) + b(p) \end{aligned} \tag{4.38}$$

Given that, by hypothesis, $M'(p) \geq b(p)$, we can conclude that $M(p) - {}^\bullet t_{t,\psi}(p) + b(p) \geq b(p)$, thus $M(p) \geq {}^\bullet t_{t,\psi}(p)$.

(ii) $M - {}^\bullet t_{t,\psi} + t^\bullet_{t,\psi} = M - {}^\bullet t + t^\bullet$. This was already proven in point (ii) above.

2. Since $t_{t,\psi}$ is undefined, let $p \in P_\psi$ such that $\kappa(p) - {}^\bullet t(p) + t^\bullet(p) < b(p)$, i.e., $-{}^\bullet t(p) + t^\bullet(p) < b(p) - \kappa(p)$. Let $M,\ M' \in R(G)$ such that $M \xrightarrow{t} M'$. We have that:

$$M'(p) = M(p) - {}^\bullet t(p) + t^\bullet(p) < M(p) + b(p) - \kappa(p) \leq b(p) \tag{4.39}$$

□

4.2.2 Composition Algorithm

The contents of this subsection were first described in [Lacerda and Lima, 2011b, Lacerda and Lima, 2011a].

As with the case of FSA, to ensure that the Petri nets supervisor takes the initial marking of the system into account, we assume that G has place $init \in P_g$ and an *initialize_system* transition. The *init* place has initial marking equal to 1, while all other places have initial markings equal to 0. Also, the *init* place is not an output place on any transition. The system always starts by firing the *initialize_system* transition, which consumes the token of the *init* place and distributes tokens to the other places of G, according to the initial state of the system.

To solve Problem 4.1, we will define a composition function that, given the Petri net system model G and the (non-deterministic) BA $B_\varphi = \langle X, \Pi, f, x_0, X_f \rangle$, where $\Pi = \Pi_{symb}$ or $\Pi = \Pi_{alg}$, depending on the system model type, builds a Petri net system model G_φ that simulates the run of the PN system model and of the BA in parallel such that G only fires an event if this event, in conjunction

with the marking it drives G to, satisfies one of the active transition labels in the current state of the determinization of B_φ. The construction of this PN follows Algorithm 3.

The algorithm creates a PN that simulates a run in parallel of the PN model of the system and the determinization of the BA, where a transition t can only fire in parallel with a transition of the determinization of the BA labelled by ψ when we are ensured that the conjunction of the event corresponding with t and the marking to which the Petri net evolves satisfies ψ.

We start by analysing the initial state $\{q_0\}$ of the determinization, and only analyse states $current\_states \in 2^Q$, that are attained during the execution of the algorithm, i.e., states added to $states\_queue$ (lines 33–36). This is similar with what is done for the BA/FSA composition. In fact, the part of this algorithm related to the BA is very similar with Algorithm 1. The main difference is that the states of the determinization are now represented by places. At each marking of the supervisor, the "Büchi place" representing the current state of the determinization has one token and all other "Büchi places" have zero tokens.

When analysing a state $current\_states$ of the determinization, taken from $states\_queue$, we start by building, for each state q of the BA, the DNF formula ℓ_q (lines 7–9), which represents all the members of Π that can take us from a state in $current\_states$ to q. This DNF formula is simply the disjunction of the DNF formulas labelling transitions from elements of $current\_states$ to q. Then, for each $t \in T$, we start by defining ℓ'_q equal to ℓ_q (line 13), and then check the minimal satisfying transition obtained for each conjunctive clause in ℓ'_q (lines 12–22). We take two cases into account:

- If the minimal satisfying transition obtained from a conjunctive clause in ℓ'_q and t coincides with t, it means that the conjunction of $\ell(t)$ with any marking obtained immediately after the firing of t satisfies the conjunctive clause. Given that the conjunctive clause is in the DNF formula ℓ'_q, this means that the conjunction of $\ell(t)$ with any marking obtained immediately after the firing of t also satisfies ℓ'_q, i.e., whenever t fires and the determinization of the BA is in a state in $current\_states$, it is guaranteed to evolve to state q. Thus, we put q in the list $next\_guaranteed\_states$ and stop going through the conjunctive clauses in ℓ'_q.

- If the minimal satisfying transition obtained from a conjunctive clause in ℓ'_q and t is not defined, it means that the conjunction of $\ell(t)$ with any marking obtained immediately after the firing of t does not satisfy the conjunctive clause. Thus, we delete this conjunctive clause from ℓ'_q [4], because it will never contribute for the satisfaction of ℓ'_q from the firing of t. After checking all conjunctive clauses in ℓ'_q, if all of them were deleted, it means that the DNF formula ℓ_q can never be satisfied by the conjunction of $\ell(t)$ with a marking obtained immediately after the firing of t. Thus, we only add q to the list of $next\_possible\_states$ if at least one conjunctive clause was not deleted (lines 23–25). Intuitively, the set of $next\_possible\_states$ represent BA states that might be reached when t fires and the BA determinization is in state $current\_states$. A state q in $next\_possible\_states$ will be reached when the conjunction of $\ell(t)$ with the marking

[4] This is why we use ℓ'_q instead of using ℓ_q directly, because we need to take ℓ_q into account for each $t \in T$, thus we cannot delete its elements directly.

Algorithm 3 Büchi/PN System Model Composition

Input: PN system model $G = \langle P, T, W^{+'}, W^{-'}, M_0, E, \ell, SD \rangle$, where $SD = \langle D, \mu \rangle$ or $SD = \langle \kappa, {}^{-} \rangle$ and syntactically safe LTL formula φ, written over Π_{symb} or Π_{alg}

Output: PN $G_\varphi = \langle P', T', W^{+'}, W^{-'}, M'_0, E, \ell', SD \rangle$

1: $B_\varphi = \langle Q, 2^{E \cup D}, \delta, q_0, Q_f \rangle \leftarrow LTL2BA(\varphi)$
2: $P' \leftarrow P$; $M'_0 \leftarrow M_0$
3: $states\_queue.push(\{x_0\})$
4: add place p with label $\{q_0\}$ to P'; $M'_0(p) \leftarrow 1$
5: **while** $states\_queue \neq \emptyset$ **do**
6: $current\_states \leftarrow states\_queue.pop()$
7: **for all** $q \in Q$ **do**
8: $\ell_q \leftarrow \bigvee_{q' \in current\_states} \ell(q', q)$ {$\ell(q', q)$ is the label ψ of transition $\delta(q', \psi) = q$, if it is defined or $false$ otherwise}
9: **end for**
10: **for all** $t \in T$ **do**
11: $next\_guaranteed\_states \leftarrow \emptyset$; $next\_possible\_states \leftarrow \emptyset$
12: **for all** $q \in Q$ **do**
13: $\ell'_q \leftarrow \ell_q$
14: **for all** conjunctive clauses γ in ℓ'_q **do**
15: **if** ${}^\bullet t_{t,\gamma} = {}^\bullet t$ and $t^\bullet_{t,\gamma} = t^\bullet$ **then**
16: add x to $next\_guaranteed\_states$; break
17: **end if**
18: **if** $t_{t,\gamma}$ is not defined **then**
19: delete γ from ℓ'_q
20: **end if**
21: **end for**
22: **end for**
23: **if** $q \notin next\_guaranteed\_states$ and not all conjunctive clauses in ℓ'_q were deleted **then**
24: add q to $next\_possible\_states$
25: **end if**
26: **if** $next\_guaranteed\_states \cup next\_possible\_states \neq \emptyset$ **then**
27: **for all** $Y \in 2^{next\_possible\_states}$ **do**
28: $next\_possible\_label \leftarrow next\_guaranteed\_states \cup Y$
29: **if** $next\_possible\_label \neq \emptyset$ **then**
30: $\psi \leftarrow \bigwedge_{y \in Y} \ell'_y \wedge \bigwedge_{y \in next\_possible\_states \setminus Y} \neg \ell'_y$
31: **for all** conjunctive clauses $\gamma \in DNF(\psi)$ **do**
32: **if** $t_{t,\gamma}$ is defined **then**
33: **if** $\nexists p \in P'$ with label $next\_possible\_label$ **then**
34: $states\_queue.push(next\_possible\_label)$
35: add place p' with label $next\_possible\_label$ to P'; $M'_0(p') \leftarrow 0$
36: **end if**
37: add transition $t_{t,\gamma}$ to T'; $\ell'(t_{t,\gamma}) \leftarrow \ell(t)$
38: ${}^\bullet t_{t,\gamma}(p) \leftarrow 1$, where p is the place with label $current\_states$
39: $t^\bullet_{t,\gamma}(p) \leftarrow 1$, where p is the place with label $next\_possible\_label$
40: **end if**
41: **end for**
42: **end if**
43: **end for**
44: **end if**
45: **end for**
46: **end while**

obtained immediately after the firing of t (which depends on the marking from which t was fired) satisfies ℓ'_q. It will not be reached when the conjunction of $\ell(t)$ with the marking obtained immediately after the firing of t satisfies $\neg\ell'_q$.

After this analysis, if both lists *next_guaranteed_states* and *next_possible_states* are empty, it means that t can never occur when the BA determinization is in state *current_states*, i.e., when there is a token in the place representing state *current_states*, thus we simply analyse another transition. If at least one of them is not empty, it means that there are situations where t can fire when there is a token in the place representing that the BA determinization is in state *current_states*. The BA determinization will always evolve to a superset of *next_guaranteed_states*, because, by construction of this set, whenever t fires, there is a DNF transition label from a state in *current_states* to a state in *next_guaranteed_states* that is satisfied by the firing of t. Then, we check for each $Y \in 2^{next\_possible\_states}$, what are the markings from which we jump to Y when t is fired (lines 27–43). These markings are encoded by the DNF formula ψ (line 30), and satisfy all the labels that take us to a state in Y and the negation of all the labels that take us to a state in $next\_possible\_states \setminus Y$. Thus, we create a minimal satisfying transition for t and each conjunctive clause in ψ (lines 31–40), using either Definition 4.2.2 if we are dealing with a symbolic state description, or Definition 4.2.10 if we are dealing with an algebraic state description. Furthermore, we need to add arcs to this minimal satisfying transition that represent the evolution of the determinization of the BA from state *current_states* to state $next\_guaranteed\_states \cup Y$ (lines 38–39). In the cases where the place representing $next\_guaranteed\_states \cup Y$ was still not added to P', we add it and also add it to the queue of places to be analysed (lines 34–35).

To summarize, for each transition $t_{t,\gamma}$ we add 3 types of arcs:

1. *System arcs*, i.e., the same arcs as in t. These arcs take into account the evolution of the system after t is fired;

2. *Büchi arcs*, i.e., arcs that consume a token from the place representing the BA determinization state *current_states* and place one token in the place representing the BA determinization state $next\_guaranteed\_states \cup Y$. These arcs take into account the evolution of the determinization of the BA;

3. *Read-arcs*, obtained from the minimal satisfying transition construction. These arcs do not change the markings in the places they are pointing to, and guarantee that any marking obtained immediately after the firing of $t_{t,\gamma}$ satisfies γ (according to the properties of minimal satisfying transitions we proved).

Hence, our problem is solved for PN system models with both symbolic and algebraic state description, because the result of the composition simulates the running in parallel of the PN system model and the BA, where we add transitions that represent all the possible firings of transitions in the system model that can satisfy a DNF label in the BA, evolving both structures accordingly.

Note that for the case of PN system models with symbolic state description, we did not refer to the knowledge base K. This was done to simplify the presentation, given that for PN system models with algebraic state description, we did not define the notion of knowledge base. The knowledge base can be easily added, by creating minimal satisfying transitions $t_{t,\gamma,K}$ instead of simply $t_{t,\gamma}$. Also, for PN system models with algebraic state description, we assume all the conjunctive clauses are composed of positive literals of the form $M(p) \geq b$, as explained in the previous section. Note that we will negate some of these literals throughout the composition, which can require the addition of complement places that were still not defined for some of the counter places. This can be done easily "on-the-fly".

Example 4.2.8 (Büchi/PN System Model with Symbolic State Description Composition)**.** Consider the fragments of a Petri net system model and of a BA depicted in Figure 4.9 (a) and (b), respectively.

We will exemplify on how to compose the transitions from Büchi state 1 with transition t_0 of the Petri net. We start by checking the minimal satisfying transition obtained from t_0 and all BA transition labels from 1:

- For $\ell'_1 = e_1$, ${}^\bullet t_{t_0,e_1} = {}^\bullet t_0$ and $t^\bullet_{t_0,e_1} = t^\bullet_0$, because $\{\ell(t_0)\} \Vdash e_1$ and for the conjunctive clause e_1, $P_{e_1} = \emptyset$. Thus, we add state 1 to the set of guaranteed next states;

- For $\ell'_2 = (e_1 \wedge d_1) \vee (\neg d_2) \vee (e_2 \wedge d_2)$, the minimal satisfying transition is not defined for $e_2 \wedge d_2$, because $\{\ell(t_0)\} \nVdash e_2$. For $\gamma_1 = e_1 \wedge d_1$, ${}^\bullet t_{t_0,\gamma_1}(p_0) = 1 \neq 0 = {}^\bullet t_0(p_0)$ and for $\gamma_2 = e_2 \wedge d_2$, ${}^\bullet t_{t_0,\gamma_2}(p_2) = 1 \neq 0 = {}^\bullet t_0(p_2)$. Hence, we delete $e_2 \wedge d_2$ from ℓ'_2, obtaining $\ell'_2 = (e_1 \wedge d_1) \vee (\neg d_2)$, and add state 2 to the set of possible next states;

- For $\ell'_3 = d_3$, the minimal satisfying transition is not defined, because $t^\bullet_0(\mu(d_3)) - {}^\bullet t_0(\mu(d_3)) = -1$. Hence the minimal satisfying transition for the only conjunctive clause in ℓ'_3 is undefined, so we ignore state 3 in the evaluation.

From this reasoning, we conclude that whenever t_0 fires and the BA determinization is in state $\{1\}$, it will evolve to a state that contains $\{1\}$. Now, we need to check, for all combinations of subsets of the set of next possible states, in which conditions does the firing of t_0 drive us to that subset. Since the set of possible next states is a singleton, we only have 2 cases:

- *t_0 fires and we are guaranteed to stay in state* $\{1\}$. To guarantee that we are in this case, we need to use t_0 to build transitions that only fire when the conjunction of $\ell(t_0)$ with the marking obtained immediately after their firing satisfies the negation of the DNF label that leads the BA from state 1 to state 2, i.e., we need to build minimal satisfying transitions from t_0 for each

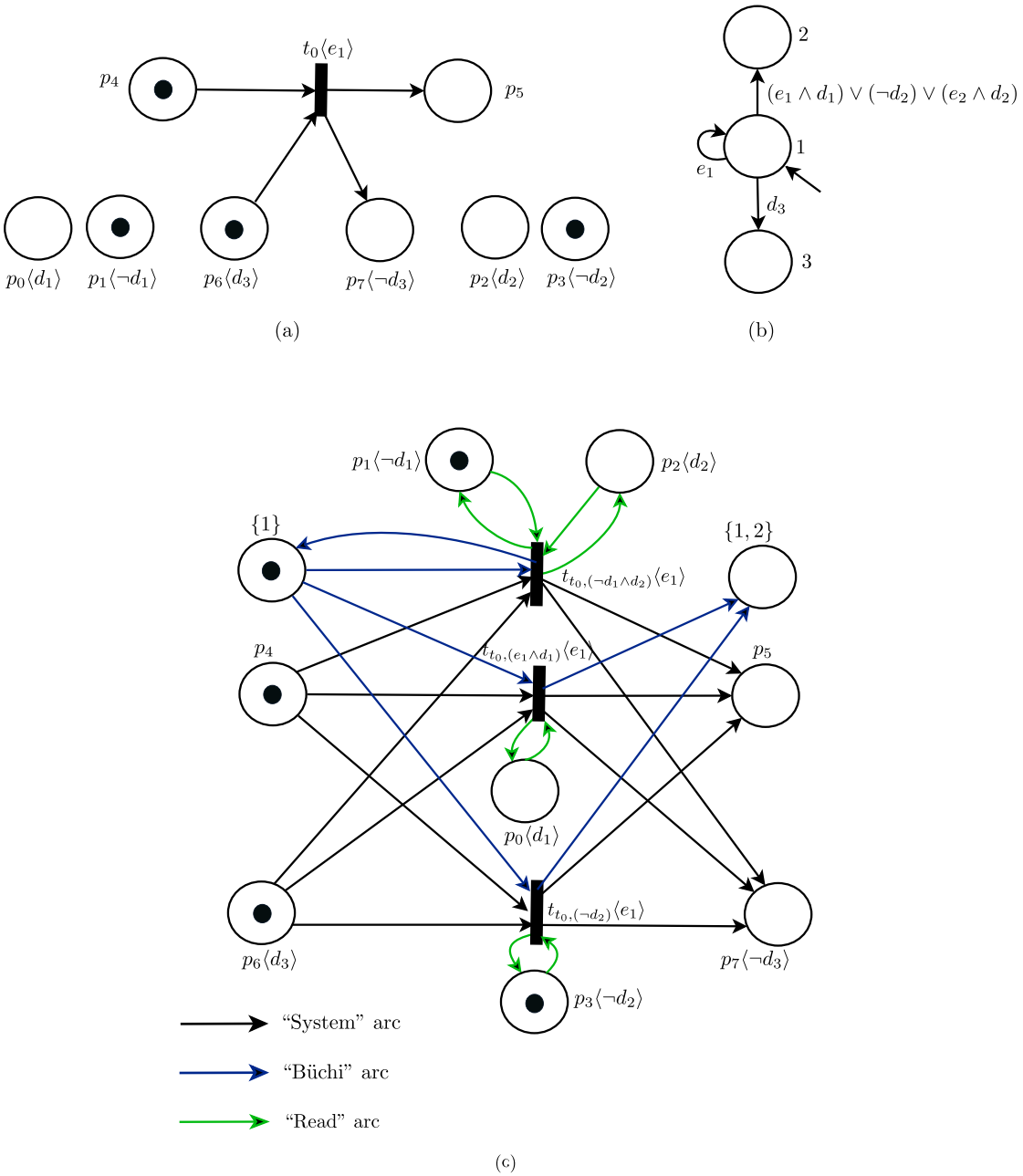


Figure 4.9: (a) A fragment of a PN system model (b) A fragment of a BA (c) The composition between the PN system model and the BA fragments.

conjunctive clause in the DNF formula $\neg\ell'_2$:

$$\begin{aligned}\neg\ell'_2 &= \neg((e_1 \wedge d_1) \vee (\neg d_2)) = \neg d_1 \wedge d_2 \\ &= \neg(e_1 \wedge d_1) \wedge d_2 \\ &= (\neg e_1 \vee \neg d_1) \wedge d_2 \\ &= (\neg e_1 \wedge d_2) \vee (\neg d_1 \wedge d_2)\end{aligned}$$

Hence, we build a minimal satisfying transition for each conjunctive clause. For $(\neg e_1 \wedge d_2)$, the minimal satisfying transition $t_{t_0,(\neg e_1 \wedge d_2)}$ is not defined. For $(\neg d_1 \wedge d_2)$ it is, and we also add to $t_{t_0,(\neg d_1 \wedge d_2)}$ an arc from the place representing the determinization state $\{1\}$ and an arc to the same place.

- t_0 *fires and we are guaranteed to go to state* $\{1,2\}$. To guarantee that we are in this case, we need to build minimal satisfying transitions for the conjunctive clause in the DNF formula ℓ'_2. The minimal satisfying transitions are defined for both conjunctive clauses $(e_1 \wedge d_1)$ and $(\neg d_2)$. Furthermore, to each minimal satisfying transition, we add an arc from the place representing the determinization state $\{1\}$ and an arc to the place representing the determinization state $\{1,2\}$.

Figure 4.9 (c) depicts the fragment of the composition obtained from the analysis above.

Example 4.2.9 (Büchi/PN System Model with Algebraic State Description Composition)**.** Consider the fragments of a Petri net system model and of a BA depicted in Figure 4.10 (a) and (b).

We will exemplify on how to compose the transitions from Büchi state 1 with transition t_0 of the Petri net. We start by checking the minimal satisfying transition obtained from t_0 and all BA transition labels from 1:

- For $\ell'_1 = M(p_1) \geq 2 \vee M(p_2) \geq 3$, ${}^\bullet t_{t_0,M(p_1)\geq 2} = {}^\bullet t_0$ and $t^\bullet_{t_0,M(p_1)\geq 2} = t_0^\bullet$, because $P_{M(p_1)\geq 2} = \{p_1\}$, and $t_0^\bullet(p_1) = 2 = b(p_1)$. Thus, we add state 1 to the set of guaranteed next states;

- For $\ell'_2 = M(p_3) \geq 2 \wedge M(p_4) \geq 4 \wedge M(p_1) \geq 2$ (note that ℓ'_2 is a single conjunctive clause, so we analyse it directly), the minimal satisfying transition is defined, and:

  - ${}^\bullet t_{t_0,\ell'_2}(p_3) = 2 \neq 0 = {}^\bullet t_0(p_3)$ and $t^\bullet_{t_0,\ell'_2}(p_3) = 2 \neq 0 = t_0^\bullet(p_3)$;
  - ${}^\bullet t_{t_0,\ell'_2}(p_4) = 2 \neq 0 = {}^\bullet t_0(p_4)$ and $t^\bullet_{t_0,\ell'_2}(p_4) = 4 \neq 2 = t_0^\bullet(p_4)$;
  - ${}^\bullet t_{t_0,M(p_1)\geq 2} = {}^\bullet t_0$ and $t^\bullet_{t_0,M(p_1)\geq 2} = t_0^\bullet$, as seen in the previous case.

  Thus, we add state 2 to the set of possible next states;

- For $\ell'_3 = (e_2 \wedge M(p_1) \geq 2) \vee M(p_2) \geq 3$, the minimal satisfying transition is not defined for $(e_2 \wedge M(p_1) \geq 2)$ because $\{\ell(t_0)\} \not\Vdash e_2$, and it is also not defined for $M(p_2) \geq 3$, because $b(p_2)+$

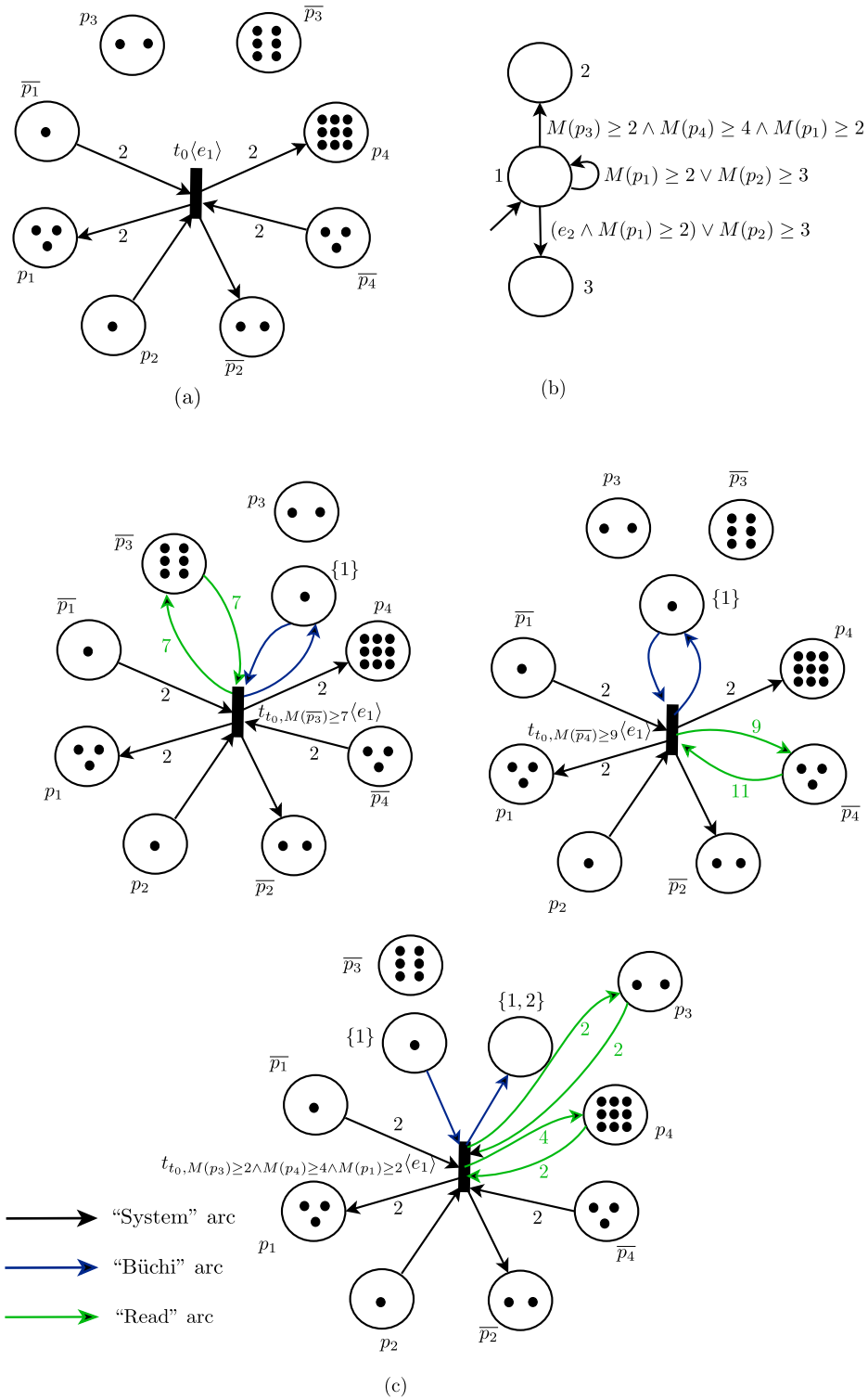


Figure 4.10: (a) A fragment of a PN system model (b) A fragment of a BA (c) The composition between the PN system model and the BA fragments.

${}^\bullet t_0(p_2) - t_0^\bullet(p_2) = 3+1-0 = 4 > 3 = \kappa(p_2)$. Thus, both conjunctive clauses of ℓ'_3 are deleted, which means that we will never evolve from state 1 to state 3 of the BA when firing t_0, i.e., we can ignore state 3 in the evaluation.

From this reasoning, we conclude that whenever t_0 fires and the BA determinization is in state $\{1\}$, it will evolve to a state that contains $\{1\}$. Now, we need to check, for all combinations of subsets of the set of next possible states, in which conditions does the firing of t_0 drive us to that subset. Since the set of possible next states is a singleton, we only have 2 cases:

- *t_0 fires and we are guaranteed to stay in state $\{1\}$.* To guarantee that we are in this case, we need to build minimal satisfying transitions for the conjunctive clause in the DNF formula $\neg\ell'_2$:

$$\begin{aligned}\neg\ell'_2 &= \neg(M(p_3) \geq 2 \wedge M(p_4) \geq 4 \wedge M(p_1) \geq 2)\\ &= \neg M(p_3) \geq 2 \vee \neg M(p_4) \geq 4 \vee \neg M(p_1) \geq 2)\\ &= M(\overline{p_3}) \geq \kappa(p_3) + 1 - 2 \vee M(\overline{p_4}) \geq \kappa(p_4) + 1 - 4 \vee M(\overline{p_1}) \geq \kappa(p_1) + 1 - 2\\ &= M(\overline{p_3}) \geq 7 \vee M(\overline{p_4}) \geq 9 \vee M(\overline{p_1}) \geq 3\end{aligned}$$

  Note that the minimal satisfying transition is not defined for $M(\overline{p_1}) \geq 3$, because $b(\overline{p_1}) + {}^\bullet t_0(\overline{p_1}) - t_0^\bullet(\overline{p_1}) = 3+2-0 = 5 > 4 = \kappa(\overline{p_1})$. Thus, we only add minimal satisfying transitions for the other 2 conjunctive clauses, also adding to them an arc from the place representing the determinization state $\{1\}$ and an arc to the same place.

- *t_0 fires and we are guaranteed to go to state $\{1,2\}$.* To guarantee that we are in this case, we need to build a minimal satisfying transition for the only conjunctive clause in the DNF formula ℓ'_2. The minimal satisfying transition is defined for ℓ_2, as we have seen before. Furthermore, we add an arc from the place representing the determinization state $\{1\}$ and an arc to the place representing the determinization state $\{1,2\}$ to the minimal satisfying transition.

Figure 4.10 (c) depicts the fragment of the composition obtained from the analysis above. We separated the places into 3 different copies to help the readability of the depiction, but in fact places with the same name in the figure represent the same place.

Trimming the Composition

In order to avoid the construction of the *reachability graph* of the PN modeling the system – which is known to be of exponential size in the size of the Petri net when the Petri net is bounded, and infinite for unbounded PNs – the described algorithm needs to analyse each transition of T against all the transitions in the output of each analysed state of the determinization of the BA. This can cause the creation of transitions in the PN realization of the supervisor that are never active, i.e., *dead* transitions. Hence, one can reduce the size of the supervisor by trimming these transitions.

Definition 4.2.11 (Set of Dead Transitions)**.** Let G be a PN system model. The set of dead transitions in the supervisor can be formally defined as:

$$T^{dead} = \{t \in T \mid \text{there is no } M \in R(G) \text{ such that } M \geq {}^\bullet t\} \tag{4.40}$$

Hence, a dead transition is a transition that has a preset that is not *coverable* by the evolution of the PN from its initial marking. The coverability property is known to be decidable, but the direct approach to the problem requires the construction of the *coverability tree*, which is known to have a complexity that is not upper-bounded in the size of the PN structure. Thus, we use an algebraic technique to delete at least a subset of T^{dead}. We note that the following is a necessary condition for a transition t not to be dead:

$$\text{Exists } w : P \to \mathbb{N} \text{ such that } M'_0 + w(W^{+'} - W^{-'}) \geq {}^\bullet t \tag{4.41}$$

The vector w is called the *firing count vector* [Murata, 1989], since its components specify the number of times that each transition must fire to reach a marking that covers ${}^\bullet t$. However, this condition is not sufficient, because one might not be able to map w into a firing sequence $\tau \in T^*$ that is feasible in the PN structure. Hence, if the following integer linear program (ILP) has no solution, we are guaranteed that t is a dead transition and we can delete it, but there might be some dead transitions that are not deleted using this method:

$$\begin{array}{ll} \text{find} & w \\ \text{subject to} & M'_0 + (W^{+'} - W^{-'})w \geq {}^\bullet t \\ & w \in \mathbb{N}^{|T|} \end{array} \tag{4.42}$$

Since ILP's are problem of very high complexity and we need to solve one for each transition in the supervisor, we relax the problem to a linear program (LP) with real variables, which can be solved much faster. For each transition t, if the LP has no solution, then the ILP does not have one either and we can delete t and all arcs from and to it. Using this relaxation, we might not delete some dead transitions for which there is a solution for the LP but there is none for the ILP but, from our experience, there are several instances of the problem above for which an ILP is not solved in a reasonable amount of time, and when it is, the LP relaxation gives the same result.

This is the most simple approach to deleting dead transitions, and other, more involved, approaches can be used, e.g., [Abdulla et al., 2004]. We did not follow dead transition deletion more deeply, however this simple approach already deletes a considerable amount of dead transitions, as we will see in Chapter 5.

4.3 Concluding Remarks

In this chapter, we presented the composition algorithms that are used to restrict a system model behaviour to an LTL specification. Before moving to the next chapter, where we will show how one can analyse the structures resulting from our compositions with regards to admissibility and deadlock-freeness, we present a short summary of the steps of the approach until now. This summary is illustrated in Figure 1.1.

1. The designer chooses one of the three presented options to model the uncontrolled behaviour of the system:

   - An FSA model – Definition 2.1.3, or;
   - A PN model with symbolic state description – Definition 2.2.13, or;
   - A PN model with algebraic state description – Definition 2.2.21.

   In general, the designer should use a modular approach for this design, using the different parallel composition functions defined for each model to "connect" the modules to each other and obtain the complete system model. We will follow this approach on the case studies on Chapter 6.

2. The designer states, in natural language, a set of rules he wants the system to fulfil. He/She then writes each of these rules as a syntactically safe LTL formula.

   - If the chosen system model is FSA or PN with symbolic state description, the LTL formulas are written over set Π_{symb} in Definition 3.1.3;
   - If the chosen system model is PN with algebraic state description, the LTL formulas are written over the set E plus linear constraints of form (4.25).

3. Each LTL formula is translated into a BA using the LTL2BA algorithm.

   - If the chosen system model is FSA or PN with symbolic state description, the BA transition labels will be DNF formulas written over set Π_{symb}, thus there is no need to re-write them;
   - If the chosen system model is PN with algebraic state description, the BA transition labels need to be re-written over set Π_{alg} in Definition 3.1.3. This can be done automatically by using the procedure described in Subsection 4.2.1.

4. Each BA is composed with the system model separately:

   - If the chosen system model is FSA, Algorithm 1 is used;
   - If the chosen system model is PN with symbolic state description, Algorithm 3 is used, where the minimal satisfying transitions are built using Definition 4.2.2;

- If the chosen system model is PN with algebraic state description, Algorithm 3 is used, where the minimal satisfying transitions are built using Definition 4.2.10;

5. Each resulting structure from the composition algorithms is checked for admissibility and deadlock-freeness, as will be described in Chapter 5.

Note that the designer is responsible for steps 1. and 2. of this procedure. The remaining steps can be executed automatically by using the algorithms we presented, and are guaranteed by construction to be correct. Thus, there are three possibilities for unwanted behaviour due to poor design:

- The system is not correctly modelled;
- The natural language specifications are not correctly stated;
- The LTL formulas are not written correctly from the natural language specifications.

We argue that these three possible sources of error are not problematic to deal with, because:

- Using FSA or PNs to model DES is a suitable choice[5] and there are many analysis tools that one can use to correct the system model.
- Natural language is how one thinks about the behaviour the system should have, thus correcting them is easy.
- LTL is a high-level formalism that allows, after a small learning curve, one to translate natural language statements to LTL in a correct way. Furthermore, one can refer to both states and events in the LTL formulas, which mirrors the way natural language specifications are usually stated. For example, for a robot system, one can refer to states of the system (e.g., the robot is grasping an object), and use events to refer to both actions issued to the robot (e.g., *start the grasping behaviour for a given object*) or changes in sensor readings (e.g., *the robot has lost track of the object it is trying to grasp*).

[5] This is supported by their widespread use on modelling these kind of systems.

CHAPTER 5

Supervisor Realization

In the previous chapter, we have seen how to compose a system model with a BA obtained from a syntactically safe LTL formula φ so that we restrict the language generated by the model to the sublanguage that satisfies the LTL formula. However, the result of this composition does not take into account some properties that are desirable in order to have a proper supervisor, as it is defined in the Ramadge-Wonham framework [Ramadge and Wonham, 1989]. In this chapter, we will see how to tackle this problem. We will start by presenting the basics of supervisory control (SC) theory, according to the Ramadge-Wonham framework, defining the requirements we want the supervisor to fulfil. We will then show how one can build supervisors with the desirable properties for FSA supervisors easily, by using the existing theory. For the case of PN system models, the solution is more cumbersome, and we introduce the limitations of PN system models in the scope of SC theory and also some existing approaches that try to deal with these problems. Taking a more pragmatic point-of-view, in order to be able to proceed with implementation, we will introduce an extra restriction over the LTL formulas that, if used, guarantees supervisor admissibility by construction. We will finish the chapter by introducing an initial approach for building a communication scheme that allows a decentralized deployment of the supervisors.

In the following, let let Σ be a finite alphabet, Σ^* the set of all finite strings built from Σ (including the empty string ε) and Σ^ω the set of all infinite strings that can be build from Σ.

5.1 Supervisory Control Basics

We start by presenting the fundamentals of supervisory control (SC) theory of discrete event systems (DES), as introduced in [Ramadge and Wonham, 1989], and define the requirements that our supervisors should fulfil. Our requirements will be less general than usual, and will only deal with

generated languages, because we did not introduce the notion of marked languages for our system models. However, one can easily generalize the notions we will present. Also, we will present the definitions already for the system models at hand, instead of the usual approach of reasoning in terms of languages. This choice is justified because it is more easily understandable and does not require the introduction of more notions about formal languages.We also do not take into account the notion of observable and unobservable events.

The purpose of SC is, given a DES model – in our case a FSA or PN system model G – of the open-loop uncontrolled behaviour of a system, to restrict its behaviour to an admissible language $\mathscr{L}_a \subseteq \mathscr{L}(G)$, through a feedback loop. We start by partitioning the event set E in two disjoint subsets $E = E_c \cup E_{uc}$. E_c is the set of controllable events, i.e., the events that can be prevented from happening by the supervisor and E_{uc} is the set of uncontrollable events, i.e., the events that cannot be prevented from happening. This partition is due to the fact that, in general, there are events that make a DES change its state that are not of the "responsibility" of the DES itself (e.g., for a DES model of a physical agent, one models changes in the environment which are not related to the agent's actions as uncontrollable events).

Example 5.1.1 (Controllable and Uncontrollable Events for Robot Systems)**.** In a robot scenario, we model the actions available to the robots, such as start or stop moving or starting a pass behaviour with a teammate as controllable events. Examples of uncontrollable events are meeting an obstacle, failing to grasp a given object or a communication failure. Uncontrollable events are related either to sensor readings or failures in performing actions.

We now formally define the notion of supervisor.

Definition 5.1.1 (Supervisor)**.** Let G be a system model. A supervisor for G is a function $S : \mathscr{L}^E_{fin}(G) \to 2^E$ that, given $s \in \mathscr{L}^E_{fin}(G)$, outputs the set of enabled events, i.e., the set of events that G can execute next. We will call the set $E \setminus S(s)$ as the set of disabled events. If G is an FSA, $\mathscr{L}^E_{fin}(G)$ is defined as:

$$\mathscr{L}^E_{fin}(G) = \{e_1 e_2 ... e_n \in E^* \mid \text{exists } q_1, ..., q_n \in Q \text{ such that } q_{i+1} = \delta(q_i, e_{i+1}) \text{ for all } i = 1, ..., n-1\} \tag{5.1}$$

If G is a PN, the finite event language generated by G is defined as:

$$\begin{aligned}\mathscr{L}^E_{fin}(G) = \{e_1 e_2 ... e_n \in E^* \mid \text{exists } M_1, ..., M_n \in R(G) \text{ and } t_1, ..., t_n \in T \\ \text{such that } M_i \xrightarrow{t_i} M_{i+1} \text{ and } \ell(t_i) = e_i \text{ for all } i = 1, ..., n-1\}\end{aligned} \tag{5.2}$$

The languages $\mathscr{L}^E_{fin}(G)$ used in the definition are the languages that are usually defined as generated languages by DES in the literature. We defined generated languages differently because (i) to evaluate LTL formulas we need languages of infinite strings and (ii) we also refer to states in our specifications. However, given that the supervisor is used in a feedback loop with the DES models, we need to define it over a prefix-closed language. We now define the language generated by a controlled system model.

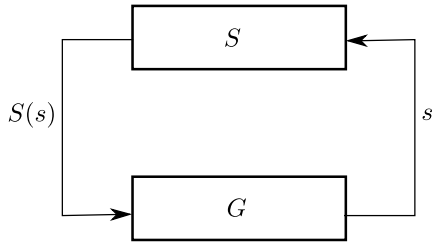


Figure 5.1: The feedback loop of monolithic supervisory control

Definition 5.1.2 (Generated Language by a Controlled System Model)**.** Let G be a system model and S be a supervisor. If G is an FSA, the language generated by G when controlled by S is given by:

$$\mathscr{L}(S/G) = \{(e_1,q_1)(e_2,q_2)... \in \mathscr{L}(G) \mid e_{i+1} \in S(e_1...e_i) \text{ for all } i \in \mathbb{N}\} \tag{5.3}$$

If G is an PN, the language generated by G when controlled by S is given by:

$$\mathscr{L}(S/G) = \{(e_1,M_1)(e_2,M_2)... \in \mathscr{L}(G) \mid e_{i+1} \in S(e_1...e_i) \text{ for all } i \in \mathbb{N}\} \tag{5.4}$$

In words, the supervisor controls the system by, after the firing of an event by G, "reading" the string s executed by G so far, and outputting a set of enabled events $S(s)$. When executing the next event, G can only execute an event which is active in its current state and which is enabled by S, i.e., which is in $S(s)$. This monolithic feedback loop is depicted in Figure 5.1.

There are two requirements that are desired for the supervisor: admissibility and deadlock-freeness. Intuitively, admissibility means that the supervisor never disables uncontrollable events and deadlock-freeness means that the supervisor never blocks the system. We formally define these two notions:

Definition 5.1.3 (Admissible Supervisor)**.** Let G be a DES model, with uncontrollable events $E_{uc} \subseteq E$, and $S : \mathscr{L}^E_{fin}(G) \to 2^E$. S is an admissible supervisor for G if, for all $s \in \mathscr{L}^E_{fin}(G)$:

$$E_{uc} \cap \Gamma_G(\delta(x_0,s)) \subseteq S(s), \text{ if } G \text{ is an FSA} \tag{5.5}$$

$$E_{uc} \cap \Gamma_G(M_{\ell^{-1}(s)}) \subseteq S(s), \text{ if } G \text{ is a PN} \tag{5.6}$$

Definition 5.1.4 (Deadlock-Free Supervisor)**.** Let G be a DES model and $S : \mathscr{L}^E_{fin}(G) \to 2^E$. S is a deadlock-free supervisor for G if, for all $s \in \mathscr{L}^E_{fin}(G)$:

$$\Gamma_G(\delta(x_0,s)) \cap S(s) \neq \emptyset, \text{ if } G \text{ is an FSA} \tag{5.7}$$

$$\Gamma_G(M_{\ell^{-1}(s)}) \cap S(s) \neq \emptyset, \text{ if } G \text{ is a PN} \tag{5.8}$$

Our goal will be to find admissible and deadlock-free supervisors from the coarse structure obtained by the composition functions presented in the previous chapter. Furthermore, we want these

supervisors to be least restrictive, in the sense we define next.

Definition 5.1.5 (Supervisor Restrictiveness)**.** Let G be a system model and $S_i : \mathscr{L}^E_{fin}(G) \to 2^E$, $i = 1,2$. We say that S_1 is less restrictive than S_2 if for all $s \in \mathscr{L}^E_{fin}(G)$, $S_2(s) \subseteq S_1(s)$. We say that S_1 is strictly less restrictive than S_2, if S_1 is less restrictive than S_2 and exists $s \in \mathscr{L}^E_{fin}(G)$ such that $S_2(s) \subset S_1(s)$.

We will slightly abuse the notation and write $S_2 \subseteq S_1$ when S_1 is less restrictive than S_2 and $S_2 \subset S_1$ when S_1 is strictly less restrictive than S_2. It is clear that if $S_2 \subseteq S_1$, then $\mathscr{L}(S_2/G) \subseteq \mathscr{L}(S_1/G)$.

We can now define the notion of least restrictive supervisor.

Definition 5.1.6 (Least Restrictive Supervisors)**.** Let G be a system model and $S : \mathscr{L}^E_{fin}(G) \to 2^E$. An $\mathscr{A}$-least restrictive supervisor obtained from G for S is a supervisor $S^a : \mathscr{L}^E_{fin}(G) \to 2^E$ such that:

- $S^a \subseteq S$;
- S^a is admissible;
- For all $S^a \subset S' \subseteq S$, S' is not admissible.

A $\mathscr{DF}$-least restrictive supervisor obtained from G for S is a supervisor $S^{df} : \mathscr{L}^E_{fin}(G) \to 2^E$ such that:

- $S^{df} \subseteq S$;
- S^{df} is deadlock-free;
- For all $S^{df} \subset S' \subseteq S$, S' is not deadlock-free.

An $\mathscr{A}/\mathscr{DF}$-least restrictive supervisor obtained from G for S is a supervisor $S^{a/df} : \mathscr{L}^E_{fin}(G) \to 2^E$ such that:

- $S^{a/df} \subseteq S$;
- $S^{a/df}$ is admissible and deadlock-free;
- For all $S^{a/df} \subset S' \subseteq S$, S' is not admissible or S' is not deadlock-free.

In the next sections, we will discuss the existence and how to obtain least restrictive supervisors for FSA and PN system models.

For analysis and implementation purposes, it is important to represent the supervisor in a convenient way. This representation is referred to as a *realization* of the supervisor. In our work, we will use the result of the composition functions defined in the previous chapter as a coarse structure G_φ from which a supervisor S_φ can be obtained by appropriately refining the structure so that it realizes a supervisor that is admissible and deadlock-free. Hence, in general, we will realize supervisors for FSA and PN models also as FSA and PNs respectively. The feedback loop depicted in Figure 5.1 is then implemented as follows: at each step, G executes an event e, according to the active events in its current state and the current enabled events by S_φ, evolving to a new state/marking. This event is then sent to the supervisor realization S_φ, which passively executes e[1], also evolving to a new

[1] We have the guarantee that e is active in S_φ, because it was enabled by it.

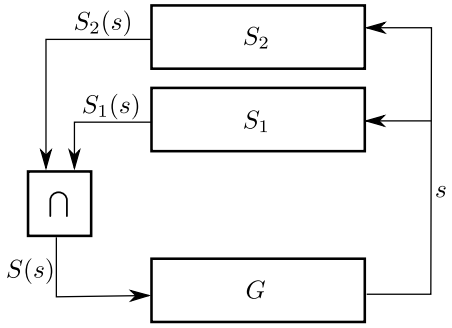


Figure 5.2: The feedback loop of modular SC, for 2 modules

state/marking. The set of enabled events after the execution of e is the set of active events of S_φ in the new state/marking. Thus, after the execution of a string $s \in E^*$, the set of enabled events by S_φ is given by $\Gamma_{S_\varphi}(\delta(x_0,s))$ if S_φ is realized by an FSA or $\Gamma_{S_\varphi}(M_{\ell^{-1}(s)})$ if S_φ is realized by a PN. In addition to simplifying the implementation, the fact that we realize supervisors using the same formalism that we use to model the system models also gives us analysis benefits. These benefits stem from the fact that, for this case, $\mathscr{L}(S_\varphi/G) = \mathscr{L}(S_\varphi)$, i.e., S_φ models the closed-loop behaviour of the system. Thus, we can use all the techniques available to analyse FSA or PNs to analyse the controlled system.

The fact that we are using FSA and PN to implement a function requires that they must be deterministic: if this is not true, then the value of $S_\varphi(s)$ is not uniquely defined, since it depends on the non-deterministic choices made when firing the events. Since we require the FSA and PN models of the system to be deterministic and we build the determinization of the BA during the composition, we are guaranteed that G_φ is deterministic. This will allow us to refine G_φ into an admissible and deadlock-free least restrictive supervisor which is realized by a deterministic FSA. However, as we will see later, the refinement of the deterministic PN G_φ so that an admissible and deadlock-free supervisor is obtained may not be closed for the class of deterministic PNs.

To finish this introduction to the basic notions of SC used in our work, we introduce *modular* SC [Wonham and Ramadge, 1988], where the supervisor S is realized by n supervisors $S_1, ...S_n$ and the set of enabled events for S is given by the intersection of the enabled events for each module, i.e., $S(s) = \cap_{i=1}^{n} S_i(s)$. This is depicted in figure 5.2.

From an implementation point of view, the use of modular supervision is preferable over using a single supervisor due the fact that we can then write a set of simpler LTL rules instead of a more complicated one, thus obtaining smaller supervisors. We write $\mathscr{L}(S_1, ..., S_n/G)$ to denote the language of the DES controlled by the supervisor S realized by the execution of the feedback-loop of modular SC using $S_1, ..., S_n$. As we will see, the use of modular SC is a benefit in terms of implementation, but makes it harder to guarantee deadlock-freeness. This is due to the following result that is a direct consequence of our definitions:

Proposition 5.1.1 ([Wonham and Ramadge, 1988]). *Let G be a system model,* $S_1, ..., S_n : \mathscr{L}^E_{fin}(G) \rightarrow$

2^E *be n supervisors and S be the supervisor obtained by execution of feedback-loop of modular SC using* $S_1, ..., S_n$*, i.e.:*

$$S(s) = \bigcap_{i=1}^{n} S_i(s) \tag{5.9}$$

Then:

- *If* $S_1, ..., S_n$ *are admissible supervisors then S is an admissible supervisor;*

- *If* $S_1, ..., S_n$ *are deadlock-free supervisors then S is a deadlock-free supervisor if and only if the monolithic supervisor* $S_{1\|2...\|n}$ *is a deadlock-free supervisor.*

Thus, to guarantee deadlock-freeness for modular supervision, one loses the computational advantage. Furthermore, the main bulk of work on related topics has do due with non-blockingness for modular SC when dealing with finite string marked languages, [Wonham and Ramadge, 1988, Hill and Tilbury, 2006, Chen et al., 2000]. Intuitively, a system model is non-blocking if at least one state in a set of *marked* states is always reachable. This notion, while related to the deadlock-freeness notion we presented, is not directly comparable with it:

- A system model might have a strongly connected component for which no marked state is reachable. While this system model is deadlock-free, it is blocking;

- A system model might have a marked state from which there is no active transition. While this system is nonblocking, it is not deadlock-free.

There is also some work related with the fundamentals of SC control for infinite string languages, which, for monolithic supervisors is more general than what we presented here (not only dealing with deadlock-freeness, but also with the Büchi acceptance condition) [Thistle, 1996, Thistle and Wonham, 1994]. However, to the best of our knowledge, there is no study on modular SC in this case. The fundamentals of modular SC for the specific case of PNs have also not been studied yet, to the best of our knowledge.

Given this limitations for modular SC, in the next section, we will only describe existing approaches on how to deal with admissibility and deadlock-freeness for the monolithic supervisory design. We also note that, when the modular supervisors are not too large, and their parallel composition can be calculated, we can use the result of this composition as the PN representation of the monolithic supervisor. This structure can be analysed using the standard techniques. Thus, the use of the modular approach allows us to handle larger system – in spite of not having formal guarantees on deadlock-freeness – and can be easily reduced to the monolithic approach whenever the size of the supervisors allow for it. We finalize by pointing out that these limitations present a lot of opportunities for future work.

5.2 Dealing with Admissibility and Deadlock-Freeness

5.2.1 Finite State Automata

In this section, we look at the construction of least restrictive supervisors for FSA system models. For this case, it is proven in the literature that the $\mathscr{A}/\mathscr{D}\mathscr{F}$-least restrictive supervisor exists and is unique. We refer the reader to, for example, [Cassandras and Lafortune, 2006] for a thorough study of SC for FSA models. In this section, we will present the algorithm for calculating the $\mathscr{A}/\mathscr{D}\mathscr{F}$-least restrictive supervisor for our case, i.e., for an FSA system model $G = \langle Q, E, \delta, q_0, D, \mu \rangle$ and the coarse supervisor $G_\varphi = \langle Q', E, \delta', q_0', D, \mu' \rangle$ obtained by the FSA system model/BA composition.

We start by stating two properties of G_φ that are a direct consequence of the fact that the composition algorithm simulates a run in parallel of G and the determinization of $B_\varphi = \langle Q^B, 2^\Pi, \delta^B, q_o^B, Q_F^B \rangle$, and will allow us to reduce the problem of finding least restrictive supervisors for FSA to the problem of finding and deleting a set of "bad" states in G_φ.

- The set of active events in a state (q, q') of G_φ, i.e., a pair where $q \in Q$ and $q' \in 2^{Q^B}$, is contained in the set of active events in state q of G, i.e.,:

$$\Gamma_{G_\varphi}((q, q')) \subseteq \Gamma_G(q) \tag{5.10}$$

- When using G_φ as a supervisor for G, at each step of the feedback-loop of SC, G is in state $q \in Q$ if and only if the element in Q of the pair that represents the current state of G_φ is also q, i.e.:

$$\begin{gathered} \text{For all } s \in E^*,\ \delta(q_0, s) = q \text{ and } \delta((q_0, \{q_0^B\}), s) \text{ is defined} \\ \text{if and only if} \\ \delta((q_0, \{q_0^B\}), s) = (q, q'), \text{ for some } q' \in 2^{Q^B} \end{gathered} \tag{5.11}$$

From these two results, we can easily characterize the non-admissibility and non-deadlock-freeness of G_φ in terms of the states in G_φ and G:

- If G_φ is not admissible for G, then there exists $q \in Q$ and $q' \in 2^{Q^B}$ such that $(q, q') \in Q'$ and:

$$\Gamma_{G_\varphi}((q, q')) \cap E_{uc} \subset \Gamma_G(q) \cap E_{uc} \tag{5.12}$$

- If G_φ is not deadlock-free for G, then there exists $(q, q') \in Q'$ such that :

$$\Gamma_{G_\varphi}((q, q')) = \emptyset \tag{5.13}$$

Thus, Algorithm 4, adapted from [Cassandras and Lafortune, 2006], yields an FSA S_φ that is admissible and deadlock-free for G. Furthermore, it is also proven in [Cassandras and Lafortune, 2006] that this FSA realizes the unique $\mathscr{A}/\mathscr{D}\mathscr{F}$-least restrictive supervisor for G_φ and G.

Algorithm 4 Calculation of the $\mathscr{A}/\mathscr{DF}$-least restrictive supervisor

Input: FSA system model G and FSA system model G_φ obtained by the system model/BA composition of G and B_φ

Output: FSA system model S_φ that realizes the $\mathscr{A}/\mathscr{DF}$-least restrictive supervisor for G_φ and G.

1: $S_\varphi = \langle Q^S, E, \delta^S, q_0^S, D, \mu^S \rangle \leftarrow G_\varphi$
2: $bad \leftarrow \{(q,q') \in Q^S \mid \Gamma_{G_\varphi}((q,q')) \cap E_{uc} \subset \Gamma_G(q) \cap E_{uc}\} \cup \{(q,q') \in Q^S \mid \Gamma_{G_\varphi}((q,q')) = \emptyset\}$
3: **if** $(q_0, \{q_0^B\}) \in bad$ **then**
4: WARNING: The least restrictive supervisor is the empty automaton
5: RETURN
6: **end if**
7: **while** $bad \neq \emptyset$ **do**
8: $Q^S \leftarrow Q^S \setminus bad$
9: $\delta^S((q,q'),e) \leftarrow \begin{cases} \delta^S((q,q'),e) & \text{if } \delta^S((q,q'),e) = (r,r') \text{ and } (q,q') \notin bad \text{ and } (r,r') \notin bad \\ \text{undefined} & \text{otherwise} \end{cases}$
10: $bad \leftarrow \{(q,q') \in Q^S \mid \Gamma_{G_\varphi}((q,q')) \cap E_{uc} \subset \Gamma_G(q) \cap E_{uc}\} \cup \{(q,q') \in Q^S \mid \Gamma_{G_\varphi}((q,q')) = \emptyset\}$
11: **if** $(q_0, \{q_0^B\}) \in bad$ **then**
12: WARNING: The least restrictive supervisor is the empty automaton
13: RETURN
14: **end if**
15: **end while**

The algorithm iteratively calculates and deletes states that satisfy either condition (5.12) or condition (5.13). At each step, after the deletion of these states, new states that satisfy one of the conditions might appear. Hence, we keep repeating the deletion until the automaton does not have states satisfying the conditions or its initial state is deleted[2]. In the latter case, a warning saying that the least restrictive supervisor is the empty automaton is given. In our approach, this means that there is no admissible and deadlock-free supervisor that restricts the behaviour of G so that it satisfies the safe LTL specification φ.

Example 5.2.1 (Calculating the $\mathscr{A}/\mathscr{DF}$-Least Restrictive Supervisor)**.** Consider the FSA system model G depicted in Figure 5.3, where $E_c = \{e_1, e_2, e_3\}$ and $E_{uc} = \{u_1, u_2\}$ and the LTL specification $\varphi = G(d_1 \Rightarrow (Xe_2))$. The BA B_φ is depicted in Figure 5.4, and the system model composition G_φ is depicted in Figure 5.5. We will apply the algorithm we just presented to G and G_φ. S_φ starts by being equal to G_φ. The set bad obtained from step 2 is $\{(q_4, \{2\})\}$, because $\Gamma_{S_\varphi}((q_4, \{2\})) = \{e_2\}$ and $\Gamma_G(q_4) = \{u_2, e_2\}$, thus $\Gamma_{S_\varphi}((q_4, \{2\})) \cap E_{uc} = \emptyset \subset \{u_2\} = \Gamma_G(q_4) \cap E_{uc}$. Thus, we delete from S_φ the state $((q_4, \{2\}))$ and all transitions from and to it. Now, in step 10, we get $bad = \{(q_3, \{1,2\})\}$ because the only transition from it was to the bad state $(q_4, \{2\})$, hence it was deleted. Then, $(q_3, \{1,2\})$ and all arcs from and to it are deleted. In the next iteration, $bad = \emptyset$, so we stop the loop and the algorithm returns S_φ, depicted in Figure 5.6.

[2]Note that we are performing a fixed point computation, using an operator on the states, given by the definition of set bad.

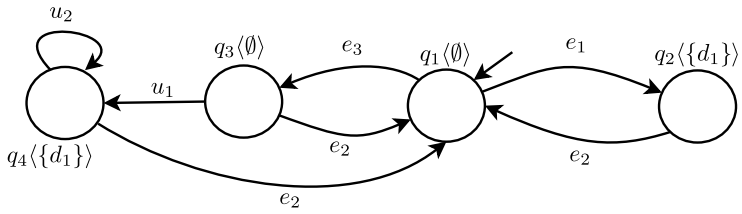


Figure 5.3: An FSA System Model

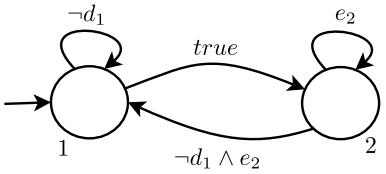


Figure 5.4: A Büchi automaton

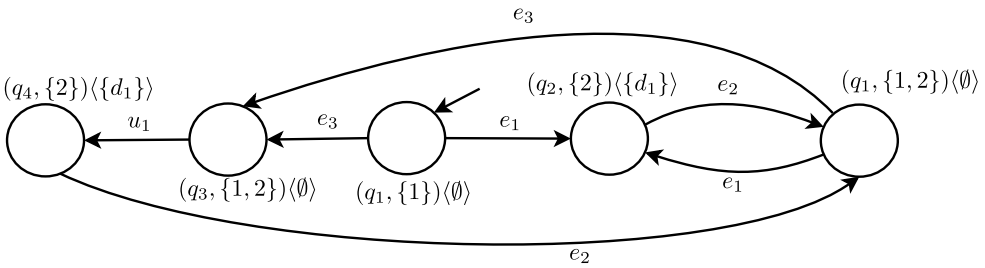


Figure 5.5: Composition between the FSA system model and the BA

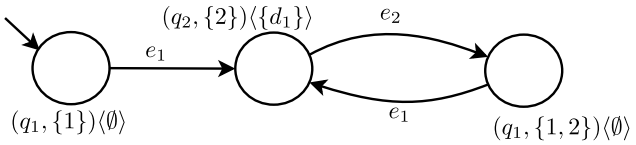


Figure 5.6: The $\mathscr{A}/\mathscr{D}\mathscr{F}$-least restrictive supervisor obtained after applying Algorithm 4, for $E_{uc} = \{u_1, u_2\}$

5.2.2 Petri Nets

For the case of PN system models, the results are not as positive as for FSA. In fact, in general it is impossible to build PN realizations $\mathscr{A}$-least restrictive supervisors (thus it is not possible to build PN realizations of $\mathscr{A}/\mathscr{DF}$-least restrictive supervisors either). However, it is possible to check PN supervisors for both admissibility and deadlock-freeness.

Admissibility

We start by briefly describing the negative result of non-existence of PN realizations of $\mathscr{A}$-least restrictive supervisors presented in [Giua and DiCesare, 1994a]. To prove the result, a counter-example, consisting of a PN G to be controlled and a non-admissible supervisor S realized by a PN such that the $\mathscr{A}$-least restrictive supervisor $S^{\mathscr{A}}$ cannot be realized by a PN, is presented. This means that $\mathscr{L}^E_{fin}(S^{\mathscr{A}}/G)$ is a language of finite strings that cannot be generated by a PN, i.e., there is no PN H such that $\mathscr{L}^E_{fin}(H/G) = \mathscr{L}^E_{fin}(S^{\mathscr{A}}/G)$. This fact is proved using the pumping-lemma for PN languages, proved in [Jantzen, 1987]. However, when the net is bounded, the $\mathscr{A}$-least restrictive supervisor exists. In [Giua, 2013] an algorithm for each construction is given. It relies on building the reachability graph of both the PN model and PN supervisor to find the *bad* markings, and then using a technique similar to the construction of minimal satisfying transitions we presented in this work to change the structure of the PN realization of the supervisor so that it never goes into the set of *bad* markings. Even though the final result is a PN built from the original supervisor, this approach requires the construction of the reachability graph to find the set of *bad* markings and the transitions that lead to them. Unfortunately, building the reachability graph is quite computationally expensive. A point to be further studied is how to build the set of *bad* markings without building the whole state space.

Given the impossibility of building least restrictive PN realizations of supervisors for the general case, we now move on to the problem of checking our BA/system model composition for admissibility. We will briefly address the issue of using control-laws that avoid "forbidden" markings in a least-restrictive way on top of the BA/system model composition again in the final remarks of this section. A very recent result [Giua, 2013] provided a procedure for checking admissibility for deterministic PN system models for the case where the specification is a deterministic PN language. However, the result is obtained under what we prove to be an invalid assumption on the so-called set of uncontrollable markings. We will point out a counter-example that shows that the set defined there (and in fact used previously in the literature) as the set of uncontrollable markings is not the set that should be used to check admissibility. Then, we will define a proper set of uncontrollable markings and show how the procedure for checking admissibility given in [Giua, 2013] can be adapted. We start by defining the problem in [Giua, 2013] in our framework.

Problem 5.1. Let $G = \langle P_G, T_G, W_G^-, W_G^+, M_{G,0}, E, \ell_G \rangle$ and $H = \langle P_H, T_H, W_H^-, W_H^+, M_{H,0}, E, \ell_H \rangle$ be deterministic labelled PNs (G represents the system model and H the PN language specification), where $E = E_c \cup E_{uc}$. Let $S = \langle P_S, T_S, W_S^-, W_S^+, M_{S,0}, E, \ell_S \rangle = G \parallel H$ be the result of the parallel composi-

tion defined in Chapter 2, without the merging of any places[3]. Decide if S is a PN realization of an admissible supervisor for G.

The set of events of G and H is the same, thus the language of S is the intersection of the languages of G and H (all transitions are synchronized). Thus, we have that, taking into account that the feedback-loop of SC also "implements" the intersection of the languages of the model and the supervisor:

$$\mathscr{L}(S/G) = \mathscr{L}(S) \cap \mathscr{L}(G) = \mathscr{L}(G) \cap \mathscr{L}(H) \cap \mathscr{L}(G) = \mathscr{L}(H) \cap \mathscr{L}(G) = \mathscr{L}(H/G) \tag{5.14}$$

Thus, from an implementation point-of-view, using S or H as the supervisor realization is equivalent. However, for analysis, S provides us with extra information: it contains two substructures, one corresponding to the structure of G and the other corresponding to the structure of H. In particular, $P_S = P_G \cup P_H$ and a marking $M \in \mathbb{N}^{|P_G|+|P_H|}$ reachable in S by an event sequence $s \in E^*$ represents the markings reachable in G and H by executing the same event sequence. Also, the firing of a transition $t = (t_G, t_H)$ in S represents the synchronous firing of transition t_G in G and transition t_H in H. With this in mind, the notion of uncontrollable marking has been introduced as early as in [Giua, 1992], as the set of markings M for which an uncontrollable transition $t = (t_G, t_H)$ in S is not active but for which t_G in G would be active for the projection of M to the places of G:

$$\begin{aligned} \mathscr{M}_b = \{M \in \mathbb{N}^{|P_G|+|P_H|} \mid & \text{ exists } t = (t_G, t_S) \in T_S \text{ such that } t_G \in T_{uc} \text{ and} \\ & \text{for all } p \in P_G,\ M(p) \geq {}^\bullet t(p) \text{ and} \\ & \text{exists } p \in P_H \text{ such that } M(p) < {}^\bullet t(p)\} \end{aligned} \tag{5.15}$$

Then, it is proven in Theorem 1 of [Kumar and Holloway, 1996] that S is admissible for G if and only if none of the markings in $\mathscr{M}_b$ is reachable in S. However, this proof does not take into account the fact that, when building the parallel composition $S = G \parallel H$, one might add more than one transition for each transition of G. Recall that if T_G^e is the set of transitions in G labelled with event e and T_H^e is the set of transitions in H labelled with the same event e, then the set of transitions in S labelled with e will be $T_G^e \times T_H^e$. Thus, for each transition $t_G \in T_G^e$, $|T_H^e|$ transitions will be created in S. We illustrate this in the following example.

Example 5.2.2. Consider the deterministic PNs G and H represented in Figure 5.7 (a) and (b) respectively, where event u is uncontrollable, i.e., t_1 is an uncontrollable transition. It is clear that both of them are deterministic. Their parallel composition $S = G \parallel H$ is depicted in Figure 5.8. It is also clear that S is admissible for G – in fact it does not even restrict the language generated by G. However, marking $\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$ is reachable in S and is clearly in $\mathscr{M}_b$: Transition $t = (t_1, t_2)$ is such that $M(p_1) = {}^\bullet t(p_1)$, where $p_1 \in P_G$ and $M(p_2) < {}^\bullet t(p_2)$, where $p_2 \in P_H$. Thus, $\mathscr{M}_b$, as defined in (5.15),

[3]This means that the places of each PN are considered independent from the places in the other. Using our terminology, in the case of system models with symbolic state description, there are no shared state description symbols and in the case of system models with algebraic state description, there are no shared places. Note that the notions of parallel composition defined for each of these system models coincide in this case.

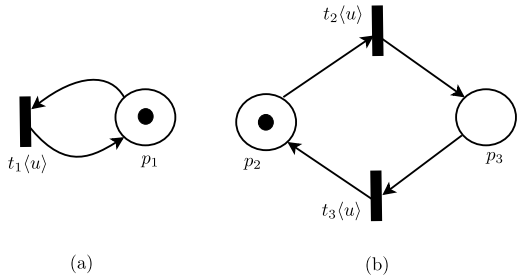


Figure 5.7: (a) A PN generator G, representing the system. (b) A PN generator H, representing the language specification.

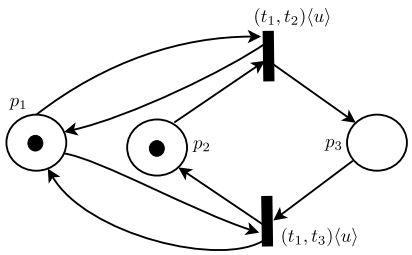


Figure 5.8: The parallel composition of G and H of Figure 5.7.

is not sound, because we need to take into account all transitions in S obtained from t_1. This fact is related to the creation of more than one transition in S corresponding to t_1.

This reasoning allows us to formally define the correct notion of uncontrollable marking in S.

Definition 5.2.1 (Uncontrollable Markings)**.** Let G and H be deterministic labelled PNs, $S = G \parallel H$ and $T_{uc} = \{t \in T_G \mid \ell_G(t) \in E_{uc}\}$. We define the set of uncontrollable markings as:

$$\begin{aligned}\mathscr{M}_{uc} = \{M \in \mathbb{N}^{|P_G|+|P_H|} \mid \text{exists } t_G \in T_{uc} \text{ such that} \\ \text{for all } p \in P_G,\ M(p) \geq {}^\bullet t_G(p) \text{ and} \\ \text{for all } t = (t_G, t_H) \in T_S \text{ exists } p \in P_H \text{ such that } M(p) < {}^\bullet t(p)\}\end{aligned} \tag{5.16}$$

A marking M is uncontrollable if exists an uncontrollable transition t_G that is active in G by the projection of M to P_G, but all the transitions created from t_G in S, i.e. pairs in $T_G \times T_H$ with the first component equal to t_G, are not active in H by the projection of M to P_H. Thus, the following proposition can be proven by a straightforward adaptation for the correct notion of uncontrollable markings of the proof of Theorem 1 in [Kumar and Holloway, 1996]:

Proposition 5.2.1. *Let G and H be two labelled PNs. $S = G \parallel H$ is an admissible supervisor for G if and only if none of the markings in $\mathscr{M}_{uc}$ is reachable in S.*

Thus, to solve Problem 5.1, we need to define a procedure to check if an element of $\mathscr{M}_{uc}$ is

reachable in S. To do that, we adapt the procedure given in [Giua, 2013], and represent $\mathscr{M}_{uc}$ in terms of partially covering markings for a given marking M.

Definition 5.2.2 (Partially Covering Markings)**.** Let G be a PN system model, $M \in \mathbb{N}^{|P|}$, and $P^{=} \subseteq P$. We define the set of partially covering markings as:

$$\mathscr{S}(M,P^{=}) = \{M' \in \mathbb{N}^{|P|} \mid M'(p) = M(p) \text{ for all } p \in P^{=} \text{ and } M'(p) \geq M(p) \text{ for all } p \in P\} \quad (5.17)$$

The set $\mathscr{S}(M,P^{=})$ is the (infinite) set of markings which are equal to M for places in $P^{=}$ and greater or equal than M for all other places. This set is a generalization of the set defined in [Giua, 2013], where $P^{=}$ is a singleton. However, in spite of being a generalization, checking the reachability of an element in this set is still decidable[4].

Proposition 5.2.2. *Let G be a PN system model, $M \in \mathbb{N}^{|P|}$, and $P^{=} \subseteq P$. Checking if a marking in $\mathscr{S}(M,P^{=})$ is reachable in G, i.e., $R(G) \cap \mathscr{S}(M,P^{=}) \neq \emptyset$ is decidable.*

Proof. The proof of this proposition relies on a simple adaptation of the construction given in [Giua, 2013].We will reduce the problem of determining if there exists a reachable marking in $\mathscr{S}(M,P^{=})$ to the problem of determining if a single reachable marking is reachable in a modified net $G' = \langle P', T', W^{-'}, W^{+'}, M'_0 \rangle$. The idea of the construction of G' is to have a PN that initially exactly mimics the behaviour of G. When G' gets to a marking M' that covers M, a special new transition becomes active. The firing of this transition changes the behaviour of G' to a behaviour that empties places in $P \setminus P^{=}$, and after all those places have zero tokens G' enters a deadlock. Hence, after this deadlock, the marking in places $p \in P^{=}$ is equal to $M(p)$, then $M' = M$, i.e., M is reachable. Thus, G' is such that:

- $P' = P \cup \{p_s, p_f\}$
- $T' = T \cup \{t_f\} \cup \{t_p \mid p \in P \setminus P^{=}\}$
- $W^{-'}$ is such that:

$$W^{-'}(p,t) = \begin{cases} W^{-}(p,t) & \text{if } p \in P \text{ and } t \in T \\ 1 & \text{if } p = p_s \text{ and } t \in T \\ 1 & \text{if } p = p_s \text{ and } t = t_f \\ 1 & \text{if } p = p_f \text{ and } t \in \{t_p \mid p \in P \setminus P^{=}\} \\ M(p) & \text{if } p \in P^{=} \text{ and } t = t_f \\ 1 & \text{if } p \in P \setminus P^{=} \text{ and } t = t_p \\ 0 & \text{otherwise} \end{cases} \quad (5.18)$$

[4]We point out that sets of partially covering markings can have infinite cardinality, and reachability is only proven for finite sets.

- $W^{+'}$ is such that:

$$W^{+'}(p,t) = \begin{cases} W^{+}(p,t) & \text{if } p \in P \text{ and } t \in T \\ 1 & \text{if } p = p_s \text{ and } t \in T \\ 1 & \text{if } p = p_f \text{ and } t = t_f \\ 1 & \text{if } p = p_f \text{ and } t \in \{t_p \mid p \in P \setminus P^{=}\} \\ 0 & \text{if } p \in P^{=} \text{ and } t = t_f \\ 0 & \text{if } p \in P \setminus P^{=} \text{ and } t = t_p \\ 0 & \text{otherwise} \end{cases} \tag{5.19}$$

- We were redundant in the definition of $W^{-'}$ and $W^{+'}$ to facilitate understanding the idea. We explain in words each of the possibilities:
  - $W^{-'}$ and $W^{+'}$ coincide with W^{-} and W^{+} for the original places and transitions in P and T.
  - Place p_s is self-looped with all transitions in T.
  - The firing of transition t_f removes one token from p_s and puts it in p_f.
  - Place p_f is self-looped with the transitions in set $\{t_p \mid p \in P \setminus P^{=}\}$.
  - Each place in $P \setminus P^{=}$ is in the preset of t_f, with weight $M(p)$.
  - Each place p in $P \setminus P^{=}$ is in the preset of the corresponding transition t_p.
- M'_0 is such that:

$$M'_0(p) = \begin{cases} M_0(p) & \text{if } p \in P \\ 1 & \text{if } p = p_s \\ 0 & \text{if } p = p_f \end{cases} \tag{5.20}$$

Thus, the evolution of G' is the following:

- Before t_f is fired, the evolution is the same as G.
- After t_f is fired, all the transitions in t cannot be fired any more, and transitions t_p can be fired until the places $p \in P \setminus P^{=}$ become empty. Afterwards, the PN enters a deadlock, where the number of tokens in places in $P^{=}$ is still the same as it was when t_f fired.
- Transition t_f only fires when the number of tokens in places $p \in P \setminus P^{=}$ is greater or equal than $M(p)$.

Thus, it is clear that the problem of reaching a marking in $\mathscr{S}(M, P^{=})$ in G is equivalent to the reachability problem of the (single) marking M' in G', where:

$$M'(p) = \begin{cases} M(p) & \text{if } p \in P^{=} \\ 1 & \text{if } p = p_f \\ 0 & \text{if } p \in P \setminus P' \text{ or } p = p_s \end{cases} \tag{5.21}$$

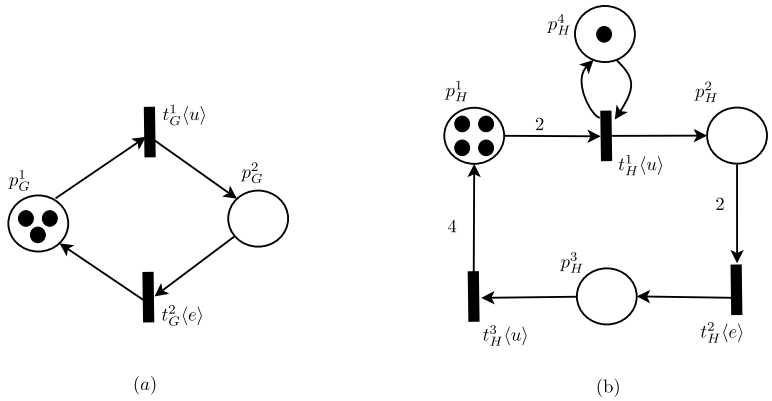


Figure 5.9: (a) A PN generator G, representing the system. (b) A PN generator H, representing the language specification.

□

Before we present the end decidability result, we introduce some helpful notation. We will use these sets to prove that checking for admissibility is equivalent to checking reachability of a finite union of partially covering markings.

Definition 5.2.3. Let $t_G \in T_G$. We define the set of transitions in H that will be synchronized with t_G as:

$$T_H(t) = \{t' \in T_H \mid \ell(t_G) = \ell(t')\} \tag{5.22}$$

Definition 5.2.4. We define the set of combinations of input places for the transitions of H that will be synchronized with t_G as:

$$P_H(t_G) = \{p \in P_H \mid {}^\bullet t_1(p) > 0\} \times \{p \in P_H \mid {}^\bullet t_2(p) > 0\} \times ... \times \{p \in P_H \mid {}^\bullet t_r(p) > 0\} \tag{5.23}$$

Definition 5.2.5. Let $t_G \in T_G$, $T_H(t_G) = \{t_1, ..., t_r\}$ and $(p_1, ..., p_r) \in P_H(t_G)$. We define the set of bad k's for t_G as:

$$K(t_G, (p_1, ..., p_r)) = \{(k_1, ..., k_r) \in \mathbb{N}^r \mid \text{for all } i, j = 1, ..., r,\ k_i < {}^\bullet t_i(p_i) \text{ and if } p_i = p_j \text{ then } k_i = k_j\} \tag{5.24}$$

Example 5.2.3. Figure 5.9 depicts a system model G (a), and a language specification H (b).

Following the definitions above, we have the following:

- $T_H(t_G^1) = \{t_H^1, t_H^3\}$
- $P_H(t_G^1) = \{(p_H^1, p_H^3), (p_H^4, p_H^3)\}$
- $K(t_G^1, (p_H^1, p_H^3)) = \{(0,0), (1,0)\}$

- $K(t_G^1, (p_H^4, p_H^3)) = \{(0,0)\}$

The set $K(t_G, (p_1, ..., p_r))$ represents all the possible combinations of tokens in the places $p_1, ..., p_r$ that will not make any of the transitions of $T_H(t_G)$ active. Thus, if t_G is uncontrollable, a marking M in $R(S)$ where (i) $M(p) \geq {}^\bullet t(p)$ for all $p \in P_G$ and (ii) there exists $(p_1, ..., p_r) \in P_H(t)$ and $(k_1, ..., k_r) \in K(t, (p_1, ..., p_r))$ such that $M(p_i) = k_i$ for all $i = 1, ..., r$, is an uncontrollable marking. From this reasoning, we have the following result, adapted from [Giua, 2013]:

Proposition 5.2.3. *Problem 5.1 can be reduced to checking reachability of a finite union of partially covering markings.*

Proof. To check admissibility, we need to check if there exists a marking in $\mathscr{M}_{uc}$ which is reachable in $S = G \parallel H$. To do that, we note that $\mathscr{M}_{uc}$ can be represented by a finite union of partially covering markings. Let $t_G \in T_{uc}$, $P^= = (p_1, ..., p_r) \in P_H(t_G)$ and $K^= \in K(t_G, P^=)$, and consider the marking $M_{t_G, P^=, K^=}$, defined over $P_S = P_G \cup P_H$ as:

$$M_{t_G,P^=,K^=}(p) = \begin{cases} k_i & \text{if } p = p_i \text{ for some } i = 1, ..., r \\ {}^\bullet t_G(p) & \text{if } p \in P_G \\ 0 & \text{otherwise} \end{cases} \tag{5.25}$$

Note that $M_{t,P^=,K^=}$ is well-defined because $K^= \in K(t, P^=)$, and we assume that members of $K(t, P^=)$ are such that if $p_i = p_j$, then $k_i = k_j$. Furthermore, as we previously discussed, the partially covering markings given by $\mathscr{S}(M_{t,P^=,K^=}, P^=)$ represent an infinite set of markings that are uncontrollable. Also, all the uncontrollable markings can be represented by partially covering markings, by going through all possible uncontrollable transitions $t_G \in T_{uc}$, all elements $P^=$ of $P_H(t_G)$ and all elements $K^=$ of $K(t_G, P^=)$. Thus, checking supervisor admissibility is equivalent to checking that the following finite union of partially covering markings is not reachable in S:

$$\bigcup_{t_G \in T_{uc}} \bigcup_{P^= \in P_H(t_G)} \bigcup_{K^= \in K(t_G, P^=)} \mathscr{S}(M_{t_G,P^=,K^=}, P^=) \tag{5.26}$$

□

This proposition gives us a procedure to check if a given supervisor $S = G \parallel H$ is admissible: check reachability in S for all the sets of partially covering markings in the finite union represented in equation (5.26). If one of them is reachable, then S is not admissible.

We now move on to specialize these result to the composition algorithms we presented. Our approach, based on the BA/system model composition is a bit different from the traditional approach based on calculating $S = G \parallel H$. However, we can frame it so we are in the conditions of applying Proposition 5.2.3. To do that, we simply take H to be G_φ, and perform the following modifications on it:

- We rename the places or state description symbols in G_φ so that there are no shared places or state description symbols between G and G_φ. Thus, no place will be merged by the parallel composition of G and G_φ.

- We add a place named *dead* with no preset and initial marking equal to zero, and for each $e \in E_{uc}$ such that there is no transition labelled with e in G_φ, we add to G_φ a transition with label e and only with place *dead* in its preset (we need this structure because in our case, if e does not appear in G_φ it will always be disabled by G_φ and it is necessary to include this information in $G \parallel G_\varphi$).

Thus, we can check if G_φ is admissible for G using Proposition 5.2.3. This reasoning also sheds a new light over our approach: we are presenting an LTL-based approach to build deterministic PN languages specifications. An interesting open question is what is the class of deterministic PN language specifications that we can specify through syntactically safe LTL.

However, this reasoning for applying Proposition 5.2.3 in our framework requires us to calculate an extra parallel composition $G \parallel G_\varphi$. In the standard case, a parallel composition between the system model G and the PN language specification H is needed because the structure of H does not have any information about the structure of G, and we need that information to check for admissibility. However, given that G_φ is obtained by the BA/system model composition, that information is already present in G_φ. This means that, we can adapt the result to use G_φ directly, avoiding the parallel composition.

The first step of the adaptation is to define the set of uncontrollable markings for G_φ.

Definition 5.2.6 (Uncontrollable Marking)**.** Let $G = \langle P_G, T_G, W_G^+, W_G^-, M_{G,0}, E, \ell_G, SD\rangle$ be a PN system model, $G_\varphi = \langle P_{G_\varphi}, T'_{G_\varphi}, W^+_{G_\varphi}, W^-_{G_\varphi}, M_{G_\varphi,0}, E, \ell'_{G_\varphi}, SD\rangle$ be a BA/system model composition and $T_{uc} = \{t \in T \mid \ell(t) \in E_{uc}\}$. We define the sets:

$$\begin{aligned}\mathscr{M}_{uc} = \{M \in \mathbb{N}^{|P_{G_\varphi}|} \mid \text{exists } t_G \in T_{uc} \text{ such that} \\ \text{for all } p \in P_G,\ M(p) \geq {}^\bullet t_G(p) \text{ and} \\ \text{for all } t = t_{t_G,\psi} \in T_{G_\varphi} \text{ exists } p \in P_{G_\varphi} \text{ such that } M(p) < {}^\bullet t(p)\}\end{aligned} \tag{5.27}$$

The definition is quite similar to the one made for the general case, however, there are two details that will change the procedure to check if an uncontrollable marking is reachable:

- The BA/system model composition also adds "read-arcs" between the transitions and places in G. As explained when describing the composition, these arcs are used to restrict the behaviour of G, avoiding its evolution to markings that do not satisfy any DNF transition labels in the BA. Thus, transitions $t_{t_G,\psi}$ can be disabled even if they are enabled by the substructure of G_φ corresponding to the BA.

- The BA/system model composition can outright delete transitions of G, if their firing is guaranteed not to satisfy the DNF transition labels of the BA. Thus, it is possible that there is no

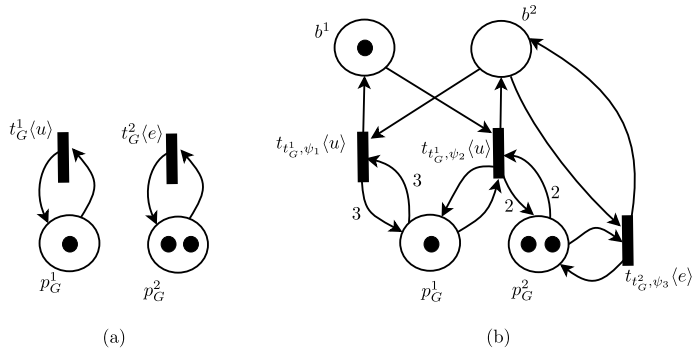


Figure 5.10: (a) A PN generator G, representing the system. (b) A PN obtained from the BA/system composition, for some BA B_φ.

transition $t_{t_G,\psi}$ in T_{G_φ}, for some transitions t_G.

Thus, taking these 2 facts into account, we adapt Proposition 5.2.3, obtaining Proposition 5.2.4. Before stating these results, we introduce some helpful notation, adapted from the definitions given for the standard case.

Definition 5.2.7. Let $t_G \in T_G$. We define the set of transitions in G_φ obtained from t_G as:

$$T_{G_\varphi}(t_G) = \{t' \in T_{G_\varphi} \mid t' \text{ is of the form } t_{t_G,\psi}, \text{ for some BA transition label } \psi\} \tag{5.28}$$

Definition 5.2.8. Let $t_G \in T_G$ and $T_{G_\varphi}(t_G) = \{t_1,...,t_r\}$. We define the set of combinations of input places for the transitions obtained from t_G in G_φ as:

$$P_{G_\varphi}(t_G) = \{p \in P_{G_\varphi} \mid {}^\bullet t_1(p) - {}^\bullet t_G(p) > 0\} \times ... \times \{p \in P_{G_\varphi} \mid {}^\bullet t_r(p) - {}^\bullet t_G(p) > 0\} \tag{5.29}$$

Note that for Büchi places, i.e., places $p \in P_{G_\varphi} \setminus P_G$, we consider ${}^\bullet t_G(p) = 0$.

Definition 5.2.9. Let $t_G \in T_G$, $T_{G_\varphi}(t_G) = \{t_1,...,t_r\}$ and $(p_1,...,p_r) \in P_{G_\varphi}(t_G)$. We define the set of bad k's for t_G as:

$$\begin{aligned} K(t_G,(p_1,...,p_r)) = \{(k_1,...,k_r) \in \mathbb{N}^r \mid &\text{ for all } i,j = 1,...,r, \\ &{}^\bullet t_G(p_i) \leq k_i < {}^\bullet t_i(p_i) \text{ and if } p_i = p_j \text{ then } k_i = k_j\} \end{aligned} \tag{5.30}$$

Example 5.2.4. Figure 5.10 depicts a system model G (a), and the PN G_φ resulting from the composition between G and a BA obtained from a formula φ (b). Note that G_φ has 2 places b_1 and b_2 which we assume represent Büchi places.

Following the definitions above, we have the following:

- $T_{G_\varphi}(t^1_G) = \{t_{t^1_G,\psi_1}, t_{t^1_G,\psi_2}\}$
- $P_{G_\varphi}(t^1_G) = \{p^1_G, b_1\} \times \{p^2_G, b_2\} = \{(p^1_G, p^2_G), (p^1_G, b_2), (b_1, p^2_G), (b_1, b_2)\}$

- $K(t_G^1,(p_G^1,p_G^2)) = \{(1,0),(2,0),(1,1),(2,1)\}$
- $K(t_G^1,(p_G^1,b_2)) = \{(1,0),(2,0)\}$
- $K(t_G^1,(b_1,p_G^2)) = \{(0,0),(0,1)\}$
- $K(t_G^1,(b_1,b_2)) = \{(0,0)\}$

With this new definitions, we are in conditions to present a result analogous to Proposition 5.2.3, adapted to the BA/system model composition.

Proposition 5.2.4. *Let G be a PN system model and G_φ the result of the BA/system model composition for a syntactically safe LTL formula φ. Checking if G_φ is admissible for G can be reduced to checking reachability of a finite union of partially covering markings.*

Proof. To check admissibility, we need to check if there exists a marking in $\mathscr{M}_{uc}$ – defined for the BA/system model composition – which is reachable in G_φ. We once again note that $\mathscr{M}_{uc}$ can be represented by a finite union of partially covering markings. Let $t_G \in T_{uc}$, $P^= = (p_1,...,p_r) \in P_{G_\varphi}(t_G)$ and $K^= \in K(t_G,P^=)$, and consider the markings $M_{t_G,P^=,K^=}$ and M_{t_G}, defined over P_{G_φ} as:

$$M_{t_G,P^=,K^=}(p) = \begin{cases} k_i & \text{if } p = p_i \text{ for some } i = 1,...,r \\ {}^\bullet t_G(p) & \text{if } {}^\bullet t_G(p) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.31}$$

$$M_{t_G}(p) = \begin{cases} {}^\bullet t_G(p) & \text{if } p \in P_G \\ 0 & \text{otherwise} \end{cases} \tag{5.32}$$

$M_{t_G,P^=,K^=}$ is defined analogously to the previous definition – the only difference is on how one defines the sets $P^=$ and $K^=$. In this case, we also define marking M_{t_G}, that will be used for the case of transitions $t_G \in T_G$ for which no transition of the form $t_{t_G,\psi}$ were created in G_φ. These markings are used to reduce the coverability problem for markings for which t_G would be active to the reachability problem of partially covering markings. Thus, checking supervisor admissibility is equivalent to checking that the following finite union of partially covering markings is not reachable in G_φ:

$$\begin{gathered} \bigcup_{t_G \in T_{uc} \mid T_{G_\varphi}(t_G) \neq \emptyset} \; \bigcup_{P^= \in P_H(t_G)} \; \bigcup_{K^= \in K(t_G,P^=)} \mathscr{S}(M_{t_G,P^=,K^=},P^=) \\ \cup \\ \bigcup_{t_G \in T_{uc} \mid T_{G_\varphi}(t_G) = \emptyset} \mathscr{S}(M_{t_G},\emptyset) \end{gathered} \tag{5.33}$$

□

This approach for checking admissibility can be easily adapted to calculate the set of reachable uncontrollable markings itself, thus it can also provide a finite representation of all the reachable

markings in the supervisor that will disable uncontrollable events that are active in G. Thus, if we can also calculate the deadlocked markings in the same way, we will have a finite representation of the set equivalent to set *bad* defined for FSA, i.e., the set of all the markings in G_φ that should be avoided in order to guarantee admissibility.

Deadlock-Freeness

Continuing from the discussion above, we will show that in fact, the deadlocked markings can also be represented by sets of partially covering markings. We start by defining the set of deadlocked markings for an arbitrary PN structure N. Given that, in the case of the BA/system model composition, the active events in G_φ are always a subset of the active events in G, it will be enough for our case to check G_φ for deadlocked markings.

Definition 5.2.10 (Deadlocked Marking)**.** Let $N = \langle P, T, W^-, W^+, M_0 \rangle$ be a PN structure. The set of deadlocked markings for N is defined as:

$$\mathscr{M}_{dead} = \{M \in \mathbb{N}^{|P|} \mid \text{for all } t \in T \text{ exists } p \in P \text{ such that } M(p) < {}^\bullet t(p)\} \tag{5.34}$$

It is clear from the definition that if a marking in $\mathscr{M}_{dead}$ is reachable in N, then N is not deadlock-free. Before we provide the reduction from checking for deadlock-freeness to reachability of a finite union of partially covering markings, we, again, present some useful notation. This definitions are analogous to the ones presented for admissibility, but taking into account that fort he case of deadlock-freeness we are interested in representing markings for which none of the transitions is active.

Definition 5.2.11. Let $N = \langle P, T, W^-, W^+, M_0 \rangle$ be a PN structure, where $T = \{t_1, ..., t_m\}$. The set of tuples of places associated to a deadlock marking in N is given by:

$$P^{dead} = \{p \in P \mid {}^\bullet t_1(p) > 0\} \times \{p \in P \mid {}^\bullet t_2(p) > 0\} \times ... \times \{p \in P \mid {}^\bullet t_m(p) > 0\} \tag{5.35}$$

For each tuple of places in P^{dead}, we also define the set of bad k's.

Definition 5.2.12. Let $N = \langle P, T, W^-, W^+, M_0 \rangle$ be a PN structure, where $T = \{t_1, ..., t_m\}$, and $(p_1, ..., p_m) \in P^{dead}$. The set ofbad k's for $(p_1, ..., p_m)$ is given by:

$$K((p_1, ..., p_m)) = \{(k_1, ..., k_m) \in \mathbb{N}^m \mid \text{for all } i, j = 1, ..., m,\ k_i < {}^\bullet t_i(p_i) \text{ and if } p_i = p_j \text{ then } k_i = k_j\} \tag{5.36}$$

Example 5.2.5. Figure 5.11 depicts a PN structure N.

Following the definitions above, we have the following:

- $P^{dead} = \{p^1, p^2\} \times \{p^2, p^3\} = \{(p^1, p^2), (p^1, p^3), (p^2, p^2), (p^2, p^3)\}$
- $K((p^1, p^2)) = \{(0,0), (1,0), (0,1), (1,1)\}$

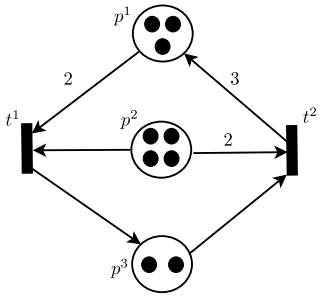


Figure 5.11: (A PN structure N.

- $K((p^1,p^3)) = \{(0,0),(1,0)\}$
- $K((p^2,p^2)) = \{(0,0)\}$
- $K((p^2,p^3)) = \{(0,0),(0,1)\}$

Note that for $K((p^2,p^2))$ we did not add $(0,1)$ because that would require for $M(p^2)$ to have two different values at the same time.

We are now in conditions to state the main result of this subsection.

Proposition 5.2.5. *Let $N = \langle P,T,W^-,W^+,M_0\rangle$ be a PN structure, where $T = \{t_1,...,t_m\}$. Checking if N is deadlock-free can be reduced to checking reachability of a finite union of partially covering markings.*

Proof. To check deadlock-freeness, we need to check if there exists a marking in $\mathscr{M}_{dead}$. Let $P^= = (p_1,...,p_m) \in P^{dead}$ and $K^= = (k_1,...,k_m) \in K(P^=)$, and consider the markings $M_{P^=,K^=}$, defined over P as:

$$M_{P^=,K^=}(p) = \begin{cases} k_i & \text{if } p = p_i \text{ for some } i = 1,...,m \\ 0 & \text{otherwise} \end{cases} \tag{5.37}$$

Note that $M_{P^=,K^=}$ is well-defined because $K^= \in K(t,P^=)$, and we assume that members of $K(t,P^=)$ are such that if $p_i = p_j$, then $k_i = k_j$. Furthermore, it is easy to see that the partially covering markings given by $\mathscr{S}(M_{P^=,K^=},P^=)$ represent an infinite set of deadlocked markings. Also, all the deadlocked markings can be represented by partially covering markings, by going through all elements $P^=$ of P^{dead} and all elements $K^=$ of $K(P^=)$. Thus, checking supervisor deadlock-freeness is equivalent to checking that the following finite union of partially covering markings is not reachable:

$$\bigcup_{P^= \in P^{dead}} \bigcup_{K^= \in K(P^=)} \mathscr{S}(M_{P^=,K^=},P^=) \tag{5.38}$$

□

There are many other approaches for deadlock-freeness checking, we presented this one to keep ourselves consistent with what we presented for admissibility checking and also to provide motivation for further work. It is known that checking deadlock-freeness is an EXPSPACE-hard problem for arbitrary PNs [Cheng et al., 1993], thus we cannot expect to solve it efficiently for bigger systems. There are many approaches for checking deadlock freeness, we refer the reader to [Girault and Valk, 2002] which covers an array of state-of-the-art analysis techniques for this problem.

The enforcement of deadlock-freeness in PN models through SC is still a big challenge for the PN community and there are many different approaches for solving the problem, though the main bulk of them does not deal with uncontrollable events. In general, there is a trade-off between the non-restrictiveness of a supervisor and the amount of computational effort that is needed to calculate it, and there are several surveys exclusively dedicated on comparing the wide array of different approaches (some of them for specific classes of PNs). We refer the reader to, for example, [Li et al., 2012, Li et al., 2008, Li and Zhou, 2009] to surveys on this subject. Most of these approaches do not deal with uncontrollable events, thus we highlight [Iordache et al., 2002] which takes this notion into account.

5.3 Specification Language Restriction for Admissibility

As mentioned in the previous section, given the limitations of modular SC theory and the SC of PNs, we present a restriction to the syntactically safe LTL formulas that guarantees supervisor admissibility. We will present the restriction for all system models.

We start by defining the set of controllable literals which is composed of literals for which the truth value is not changed by the occurrence of uncontrollable events. This means that our supervisor can always enforce that an element of D_c must be satisfied, while being admissible.

Definition 5.3.1 (Controllable State Description Literals – FSA System Models)**.** Let $G = \langle Q, E, \delta, q_0, D, \mu \rangle$ be an FSA system model. The set lit_c of controllable state description literals is defined as:

$$lit_c = \{l \in lit(D) \mid \text{For all } q \in Q \text{ and } e \in E_{uc} \cap \Gamma_G(q), \text{ if } \mu(q) \Vdash l \text{ then } \mu(\delta(q,e)) \Vdash l\} \tag{5.39}$$

To check is a given state description literal l is controllable, we need to analyse all states where l is satisfied. If for all of these states there is no uncontrollable event that can occur and drive the FSA to a state where l is not satisfied, then l is controllable.

Example 5.3.1 (Controllable State Description Literals – FSA)**.** Consider the FSA system model in Figure 5.12. In order to simplify the example, δ is only defined for state x, but in general one needs to do this analysis for all states that satisfy a given literal. We assume that event e is controllable and event u is uncontrollable. It is possible to see that literal d_2 is not controllable: $\mu(x) \Vdash d_2$ but $\mu(z) \nVdash d_2$ and $\delta(x,u) = z$. On the other hand, literal $\neg d_3$ is controllable: $\mu(x) \Vdash \neg d_3$, $\mu(z) \Vdash \neg d_3$ and, in spite of $\mu(y) \nVdash \neg d_3$, the event that drives x to y is controllable, thus can be disabled. With a similar

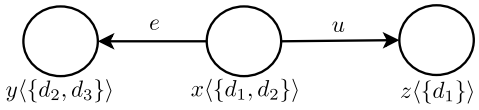


Figure 5.12: Checking which state description literals are controllable – FSA.

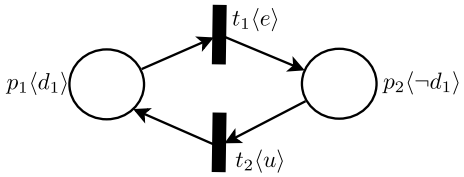


Figure 5.13: Checking which state description literals are controllable – PN with symbolic state description.

reasoning, we can check that literal d_1 is also controllable. All the other literals, i.e., $\neg d_1$, $\neg d_2$ and d_3 are not satisfied in x, hence are also controllable.

Definition 5.3.2 (Controllable State Description Literals - PN with Symbolic State Description)**.** Let $G = \langle P, T, W^+, W^-, M_0, E, \ell, D, \mu \rangle$ be a PN system model. The set lit_c of controllable state description literals is defined as:

$$lit_c = \{l \in lit(D) \mid \text{for all } t \in T_{uc}, \text{ if } {}^\bullet t(\mu(l)) = 1 \text{ then } t^\bullet(\mu(l)) = 1\}$$

The set lit_c contains the literals associated to places that are not in the preset of an uncontrollable transition or, if so, are also in the postset of that transition. This means that our supervisor can always enforce that the truth value of elements l of lit_c cannot be altered while being controllable, because we can disable all transitions that can change the marking of the place corresponding to l, $\mu(l)$, since such transitions correspond to controllable events.

Example 5.3.2 (Controllable State Description Literals - PN with Symbolic State Description)**.** In Figure 5.13, we show an example of a PN system model, where event e is controllable and event u is uncontrollable. The set of uncontrollable transitions is $T_{uc} = \{t_2\}$. The state description literal d_1 is controllable because ${}^\bullet t_2(p_1) = 0$, and $\neg d_1$ is uncontrollable because ${}^\bullet t_2(p_2) = 1$ (i.e., p_2 is in the preset of t_2) and $t_2^\bullet(p_2) = 0$ (i.e., p_2 is not in the postset of t_2).

Definition 5.3.3 (Controllable State Description Literals - PN with Algebraic State Description)**.** Let $G = \langle P, T, W, M_0, E, \ell, \kappa, {}^- \rangle$ be a PN system model. The set lit_c of controllable state description literals is defined as:

$$lit_c = \{(p, b) \in P_b \times \mathbb{N} \mid \text{for all } t \in T_{uc}, \text{ if } t^\bullet(p) - {}^\bullet t(p) < 0 \text{ then } t^\bullet(p) \geq b\}$$

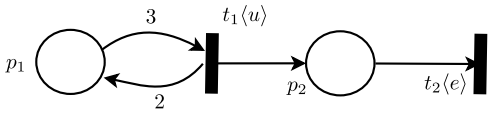


Figure 5.14: Checking which state description literals are controllable – PN with algebraic state description.

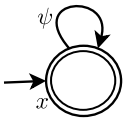


Figure 5.15: BA that generates the language satisfying type 1 formulas.

The set lit_c corresponds to pairs of places and positive integers p,b for which, if the firing of an uncontrollable transition consumes tokens from it, then that firing puts in the place an amount of tokens greater or equal than b.

Example 5.3.3 (Controllable State Description Literals - PN with Algebraic State Description)**.** In Figure 5.14, we show an example of a PN system model, where event e is controllable and event u is uncontrollable. The set of uncontrollable transitions is $T_{uc} = \{t_1\}$. Literals $M(p_1 \geq 1)$ and $M(p_1 \geq 2)$ are controllable, because though $t_1^\bullet(p_1) - {}^\bullet t_1(p_1) = -1 < 0$, $t_1^\bullet(p_1) = 2$.This means that immediately firing t_1, p_1 will have at least 2 tokens. Also, for all $b \geq 3$, $M(p_1) > b$ is an uncontrollable literal. Since no uncontrollable transition takes tokens from p_2, the literals $M(p_2) \geq b$ are controllable for all $b \in \mathbb{N}$.

Definition 5.3.4 (LTL Formula Restrictions for Supervisor Admissibility)**.** LTL specifications φ must be of one of the following types:

Type 1. $G\psi$, where ψ is a propositional formula in the DNF where only literals in lit_c that are satisfied in the initial state, or literals $\neg e$ with $e \in E_c$ can occur. This type of formula can be encoded as the deterministic BA in Figure 5.15.

Type 2. $G(\gamma \Rightarrow X\psi)$, where γ is any propositional formula in the DNF and ψ is a propositional formula in the DNF where only literals in lit_c that also appear in γ, or literals $\neg e$ with $e \in E_c$ can occur. This type of formula can be encoded as the deterministic BA in Figure 5.16.

Type 3. $G(\gamma \Rightarrow X(\psi W \gamma'))$, where γ and γ' are any propositional formulas in the DNF and ψ is a propositional formula in the DNF where only literals in lit_c that also appear in γ, or literals $\neg e$ with $e \in E_c$ can occur. This type of formula can be encoded as the deterministic BA in Figure 5.17.

The BA were obtained by determinizing the BA obtained from LTL2BA [Gastin and Oddoux, 2001], but if one carefully analyses them, it is clear that they accept exactly the ω-strings that satisfy the formulas.

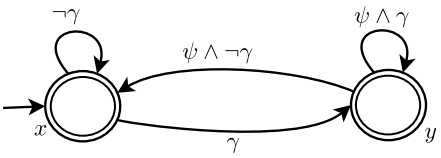


Figure 5.16: BA that generates the language satisfying type 2 formulas.

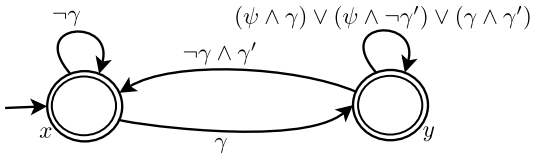


Figure 5.17: BA that generates the language satisfying type 3 formulas.

Formulas of type 1 are formulas that specify propositional logic relations between state descriptions and controllable events that should always be satisfied in all markings of the supervised PN. For example, we might want that two given robots do not move at the same time or that a certain controllable event never occurs. To ensure admissibility, we only allow the occurrence in ψ of literals in lit_c or the negation of controllable events.

Formulas of type 2 specify that the propositional logic formula ψ must be satisfied exactly after condition γ is met. For example, we may specify that if a robot is moving forward and there is no obstacle in front of it, it should continue moving forward:

$$G((moving\_forward \wedge no\_obstacle) \Rightarrow (X continue\_moving)) \tag{5.40}$$

Since γ is a condition, it can have occurrences of all atomic propositions. To ensure admissibility, we require that only literals in lit_c that also appear in γ or the negation of controllable events can occur in ψ. This means that, in this type of formula, one can require that a given controllable state description symbol maintains its truth value but cannot enforce the occurrence of an event that changes it.

Formulas of type 3 specify that immediately after condition γ is satisfied, the propositional formula ψ must keep being satisfied until condition γ' is met. For example, if a robot is moving forward and there is no obstacle in front of him, it should continue moving forward until the goal region is reached.

$$G((moving\_forward \wedge no\_obstacle) \Rightarrow (X(continue\_movingW goal\_reached))) \tag{5.41}$$

The restrictions for the literals appearing in ψ are the same as for formulas of type 2. We also note that, using this kind of formula, one can state that, after condition γ is met, a given controllable event

should be the next controllable event to occur. This can be seen as stating that a given controllable event should occurs as soon as possible. Let $e \in E_c$ be that event. The following formula of type 3 is a translation to LTL of the previous natural language statement:

$$G(\gamma \Rightarrow X((\bigwedge_{e' \in E_c \setminus \{e\}} \neg e')We)) \tag{5.42}$$

These 3 types of formulas already allow us to specify a wide array of different behaviours for a system to fulfil. In fact, in our experience, all the natural language specifications we defined for several different examples can be translated into LTL formulas of one of these types. We note that an LTL formula of one of these types ensures admissibility of the supervisor because:

- We never enforce that a token must remain in a place that is in the preset of an uncontrollable transition;
- We never enforce that a token must not be put in a place that is in the postset of an uncontrollable transition;
- We never enforce the occurrence of an event or set of events, we may only enforce that a given subset of E_c must not happen at a given moment.

We now prove this statement using the BA obtained for each type of formula.

Proposition 5.3.1. *Let G be a PN system model and φ a formula of type 1, 2 or 3. Then G_φ is admissible for G.*

Proof. We first start by analysing what are the labels that cannot be satisfied in each state of the BA for each type of formula.

- For type 1 formulas, it is clear that any transition firing that satisfies $\neg\psi$ will be disabled when the BA is in state x.
- For type 2 formulas, if the BA is in state x, then any transition firing that does not satisfy none of the DNF labels of arcs from state x, i.e., when it satisfies $\neg(\neg\gamma \vee \gamma) = \gamma \wedge \neg\gamma = false$. Thus, when the BA is in state x, all transitions can fire. When the BA is in state y, transitions that do not satisfy either $\psi \wedge \gamma$ or $\psi \wedge \neg\gamma$ are disabled, that is, the transitions that will be disabled will be the ones for which the firing satisfies:

$$\begin{aligned}
\neg((\psi \wedge \gamma) \vee (\psi \wedge \neg\gamma)) &= \neg(\psi \wedge \gamma) \wedge \neg(\psi \wedge \neg\gamma) \\
&= (\neg\psi \vee \neg\gamma) \wedge (\neg\psi \vee \gamma) \\
&= (\neg\psi \wedge (\neg\psi \vee \gamma)) \vee (\neg\gamma \wedge (\neg\psi \vee \gamma)) \\
&= (\neg\psi \wedge \neg\psi) \vee (\neg\psi \wedge \gamma) \vee (\neg\gamma \wedge \neg\psi) \vee (\neg\gamma \wedge \gamma) \\
&= \neg\psi \vee (\neg\psi \wedge \gamma) \vee (\neg\psi \wedge \neg\gamma) \\
&= \neg\psi
\end{aligned}$$

Thus, any transition firing that satisfies $\neg\psi$ will be disabled when the BA is in state y.

- For type 3 formulas, the reasoning for state x is analogous to the reasoning for type 2 formulas, and we can conclude that no transition will be disabled. When the BA is in state y, transitions that do not satisfy either $\neg\gamma \wedge \gamma'$ or $(\psi \wedge \gamma) \vee (\psi \wedge \neg\gamma') \vee (\gamma \wedge \gamma')$ are disabled. With the same type of calculation as above, we get that transitions for which the firing satisfies $(\neg\gamma' \wedge \neg\psi)$ are disabled. Thus, the set of transitions which are disabled in state y is a subset of the transitions for which the firing satisfies $\neg\psi$.

From the reasoning above, we can conclude that it is sufficient to show that the firing of an uncontrollable transition will never satisfy $(\neg\psi)$. This is a direct consequence of the restrictions we put on ψ:

- Given that we can only write negations of controllable events in ψ, $\neg\psi$ the only elements of $lit(E)$ in $\neg\psi$ will be positive literals $e \in E_c$. The label of an uncontrollable transition is in E_{uc}, hence it will never satisfy any of the event literals.

- We can only write controllable literals in ψ, where, for formulas of type 1, those literals are satisfied in the initial marking and, for formulas of type 2 and 3, those literals also appear in γ. Furthermore, for formulas of type 2 and 3, γ needs to be satisfied to make the BA go to state y. Thus, a firing of a transition satisfies $\neg\psi$ when it changes the PN from a marking M that satisfies all the controllable literals in ψ to a marking M' for which at least one of the controllable literals in ψ is not satisfied. Given our definition of controllable literal, it is clear that none of the uncontrolled transitions will have this behaviour, hence, no uncontrollable transition satisfies $\neg\psi$, i.e., no uncontrollable transition is disabled.

□

5.4 Decentralized Approach

In the particular case of multi-robot systems, a drawback of the presented methodology is related to its centralized nature, which induces high complexity and low tolerance to failures during execution. To mitigate this problem, we present a first approach for a decentralized version, where each robot runs an individual PN controller and an individual set of PN realizations of supervisors. This approach is based on the individual robot PN system models with symbolic state description and the LTL specifications for each individual robot. It relies on augmenting the individual PN system models and creating a communication scheme. Robots communicate both changes in the environment and occurrence of events that are relevant for the coordination (the relevant data to be communicated is obtained from the LTL specifications), and the PN models of each robot are augmented so that the markings can be updated by the occurrence of these communication events. Thus, one can avoid building the parallel composition of the individual models to obtain a model of the multi-robot team, while still being

able to use the BA/system model composition presented in the previous chapter for each individual PN model and each LTL specification. One can then deploy, for each robot, the corresponding PN supervisors. The work presented in this section was first described in [Lacerda and Lima, 2011c].

In the following, we assume a team of n robots, with individual PN system models with symbolic[5] state description $G_i = \langle P_i, T_i, W_i^-, W_i^+, M_{0i}, D_i, \mu_i \rangle$, $i = 1, ..., n$. Each robot model event set is partitioned into controllable and uncontrollable events, i.e., $E_i = E_{c,i} \cup E_{uc,i}$. The set E of events of the system is the union of the events of each individual model and the set D of state description symbols of the system is the unions of the state description symbols for each individual model.

We start by defining the sets of shared events and shared state description symbols, for each robot i.

Definition 5.4.1 (Shared Events)**.** The set of shared events for robot i is defined as:

$$E_i^s = \bigcup_{\substack{j=1 \\ j \neq i}}^{n} E_i \cap E_j \tag{5.43}$$

Shared events represent the triggering of collaborative actions between the robots.

Definition 5.4.2 (Shared State Description Symbols)**.** The set of shared state description symbols for robot i is defined as:

$$D_i^s = \bigcup_{\substack{j=1 \\ j \neq i}}^{n} D_i \cap D_j \tag{5.44}$$

Shared state description symbols represent environmental changes that are present in the model of more that one robot and/or can happen due to more than one robot. In this case, we need to add a new restriction to the set of controllable state description literals used to restrict the specification language. Due to the decentralized nature of this solution, each robot cannot guarantee that its teammates will not alter the truth value of its shared description symbols. Hence, we also exclude it from the set of controllable state description literals.

Definition 5.4.3 (Controllable State Description Literals)**.** The set $lit_{c,i}$ of controllable state description literals for robot i is defined as:

$$lit_{c,i} = \{l \in lit(D_i) \mid l \notin lit(D_i^s) \text{ and for all } t \in T_{i,uc},\ {}^\bullet t(\mu_i(l)) = 0\} \tag{5.45}$$

Remark 5.4.1. To guarantee that a shared state description symbol d remains unaltered, we need to check if all the robots that share d have similar LTL specifications regarding it. This would imply analysing the LTL specifications for all the robots sharing d, which is subject of future work.

We will write n sets of specifications, one for each robot.[6] For robot i, the formulas are written

[5] This approach was developed only for this type of system model.

[6] A subject of future research is following the more elegant approach of writing global specifications – that can include shared description symbols in the set of controllable literals for the team – to be satisfied in a local and decentralized way.

over $\Pi = E \cup D$, i.e., over the events and state description symbols of the whole system. We comply with the admissibility rules but, for each robot i, we take into account the sets $E_{c,i}$ and $lit_{c,i}$ instead of the sets E_c and lit_c taken into account in the centralized version. This is because we are building local supervisors for each robot, and we cannot have a supervisor for robot i disabling actions that are related to robot j (but a supervisor of robot i might change the behaviour of i after j executes an action, for example). Hence the writing of specifications over $E \cup D$.

Formally, we are interested in solving the following problem:

Problem 5.2. Given n individual PN models $G_1, ...G_n$ and sets $\Phi_1, ...\Phi_n$, where $\Phi_i = \{\varphi_i^1, ..., \varphi_i^{m_i}\}$ is the set of LTL specifications for each robot, written over the set $\Pi = E \cup D$, and satisfying Definition 5.3.4 (using the sets $E_{c,i}$ and $lit_{c,i}$ instead of E_c and lit_c) build, for each robot i:

- An augmented model G_i^{comm} that handles the required communications between robot i and other robots;
- m_i admissible PN realizations of supervisors $S_{\varphi_i^1}, ..., S_{\varphi_i^{m_i}}$

such that $\mathscr{L}^D(G_i^{comm}/S_{\varphi_i^1}, ...S_{\varphi_1^{m_i}})$ is the largest language contained in $\mathscr{L}^D(G_i^{comm})$ such that:

$$\text{If } \sigma \in \mathscr{L}^D(G_i^{comm}/S_{\varphi_i^1}, ..., S_{\varphi_i^{m_i}}) \text{ then } \sigma \Vdash \varphi_i^j \text{ for all } j = 1, ..., m_i \tag{5.46}$$

To solve this problem, we will define the augmentation of each individual PN model, which is based in the sets of shared events and shared state description symbols and in the LTL specifications. Afterwards, we compose the augmented PNs with the BA obtained from the formulas, following Algorithm 3.

We describe the process for a given robot i. The approach is analogous for all other members of the team. For each robot i, we start by defining the sets of robots that share events $e \in E_i^s$ and state description symbols $d \in D_i^s$ with i, i.e., robots that need to communicate with i to execute e or to inform i that the truth value of d has changed.

Definition 5.4.4 (Robots Sharing an Event with i)**.** Let $e \in E_i^s$. The set of robots sharing an event with i is defined as:

$$comm_e^i = \{j \in \{1, ..., n\} \mid j \neq i \text{ and } e \in E_i \cap E_j\} \tag{5.47}$$

The set of shared events with robot i represents the triggering of collaborative actions between the robots for which robot i must be involved, e.g., in a robot soccer scenario, a pass from another robot to robot i. To fire transitions associated to shared events, the robots must communicate and make sure that all the involved robots are ready to execute the action. Since all the models already include these shared actions, there is no need of adding anything to the individual PNs.

Definition 5.4.5 (Robots Sharing a State Description Symbol with i)**.** Let $d \in E_i^s$. The set of robots

sharing a state description symbol with i is defined as:

$$comm_d^i = \{j \in \{1,...,n\} \mid j \neq i \text{ and } d \in D_i \cap D_j\} \tag{5.48}$$

The set of shared state description symbols for robot i represents environmental changes that can happen or be observed by robot i and also by other robots. Whenever the truth value of a shared state description symbol d is changed by a team member, i.e., whenever one of the robots $j \in comm_d^i$ goes from a marking M to a marking M' such that $M(\mu(d)) \neq M'(\mu(d))$ (equivalently, $M(\mu(\neg d)) \neq M'(\mu(\neg d))$), it needs to send a message to robot i saying that d has changed and what is its truth value. The individual PN model of robot i needs to be augmented to handle this change in d in the cases where it did not observe it. To do this, we add two new uncontrollable events to E_i, named $change\_d\_true$ and $change\_d\_false$. We also add 4 transitions to T_i to cover the following cases:

1. Message from $j \in comm_d$ stating that the truth value of d changed to $true$, and it is already $true$ in the current marking of i.;
2. Message from $j \in comm_d$ stating that the truth value of d changed to $true$, and it is $false$ in the current marking of i;
3. Message from $j \in comm_d$ stating that the truth value of d changed to $false$, and it is already $false$ in the current marking of i;
4. Message from $j \in comm_d$ stating that the truth value of d changed to $false$, and it is $true$ in the current marking of i;

The first two cases deal with the situations where when the robot receives the message, it already observed the change of the truth value of d. The last two cases deal with the situations where the message states a change of the truth value of d that the robot did not observe. In Figure 5.18 (a) we depict an individual PN system model, and in Figure 5.18 (b), we depict the model after adding the transitions for a shared state description symbol d_2.

We also need to take into account the information needed to fulfil the LTL specifications, which, as stated before, are written over the set $\Pi = E \cup D$. We start by defining the sets of occurrences of events and state description symbols for a given formula φ.

Definition 5.4.6 (Occurrences of Symbols in φ)**.** Let φ be an LTL formula for robot i, written over $E \cup D$. The sets of events and state description symbols occurring in φ are defined as:

$$occ_\varphi^E = \{e \in E \mid e \text{ occurs in } \varphi\} \tag{5.49}$$

$$occ_\varphi^D = \{d \in D \mid d \text{ occurs in } \varphi\} \tag{5.50}$$

These sets contain, respectively, the elements of E that occur in φ and the elements of D that occur in φ. We will need to augment G_i so that it can handle the events and state description symbols that

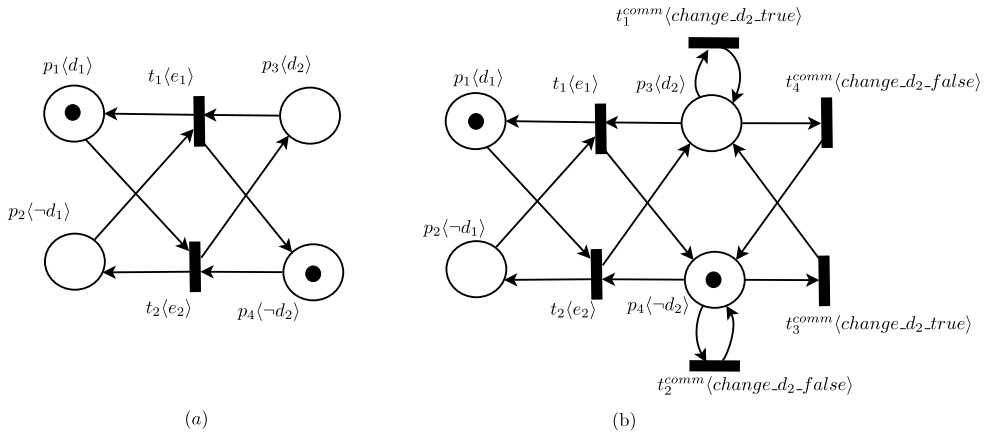


Figure 5.18: (a) A PN system model for an individual robot. (b) The individual model after adding the transitions to handle communication, for shared state description symbol d_2.

occur in φ but are not in G_i. Hence, we define the sets of external events and external state description symbols in ext_φ^E and ext_φ^D.

Definition 5.4.7 (External Symbols in φ)**.** Let φ be an LTL formula for robot i, written over $E \cup D$. The sets of external events and external state description symbols of φ are defined as:

$$ext_\varphi^E = occ_\varphi^E \setminus E_i \tag{5.51}$$

$$ext_\varphi^D = occ_\varphi^D \setminus D_i \tag{5.52}$$

External events and external state description symbols represent information about the global state of the system that is needed for robot i to satisfy φ, but that are not included in the individual PN model for robot i. The elements d of ext_φ^D are handled in a similar manner as the shared state description symbols. The difference is that we also need to add d to D_i and add two places to P_i corresponding to $\mu(d)$ and $\mu(\neg d)$ and do not need to add transitions that represent that the robot already observed a change on the truth value of d before receiving the communication, as depicted in Figure 5.19. The initial marking of these places can be obtained from the initial marking of a PN G_j such that $d \in D_j$[7]. Also, we add i as a communication target for the change of the truth value of d in the PNs G_j such that $d \in D_j$.

For the elements of ext_φ^E, we simply add e to E_i as an uncontrollable event and a transition t with label e that is always active to T_i and fires without changing the marking of G_i. For example, Figure 5.20 depicts the place and transitions added to G_i if $ext_\varphi^E = \{e_4, e_5, e_7\}$. Furthermore, whenever a robot j such that $e \in E_j$ fires a transition with label e, it sends a message to i which also fires its added transition with label e – this maintains G_i in the same marking, but will change the marking of the supervisor obtained from φ, thus changing the set of enabled events.

[7]We continue assuming that the initial markings are consistent.

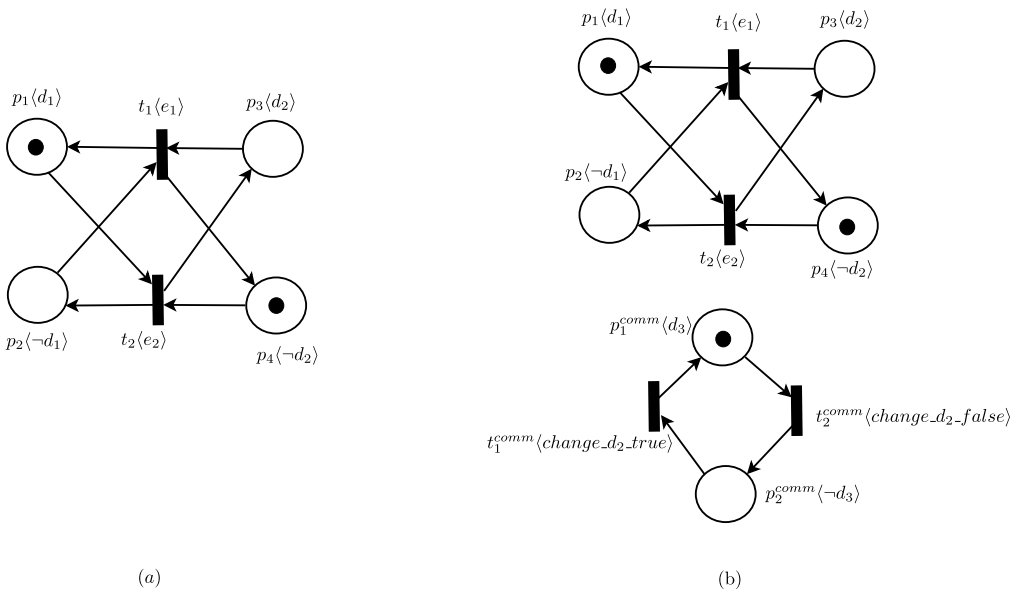


Figure 5.19: (a) A PN system model for an individual robot. (b) The individual model after adding the places and transitions to handle communication, for external state description symbol d_3.

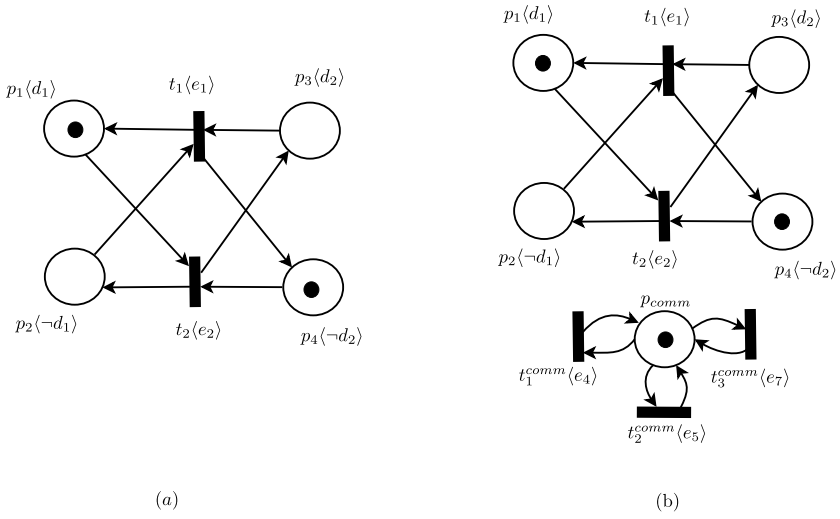


Figure 5.20: (a) A PN system model for an individual robot. (b) The individual model after adding the place and transitions to handle communication, for external events $\{e_4, e_5, e_7\}$.

Following this procedure for each robot i, we obtain the augmented PN models G_i^{comm} to be composed with the BA. Note that in these augmented models we added exactly the external events and state description symbols that are required to evaluate the LTL specifications. In the centralized case, where we model multi-robot systems by using parallel composition, the model to be composed with the BA is larger, since it takes into account all events and state description symbols of all robots. We will see in the next Chapter that the decentralized approach is a relevant improvement in terms of supervisor size executed for each robot, not only because only relevant external events and state description symbols are added to the individual models but also because they can be executed in a decentralized manner, thus each robot executes the same number of modular supervisors regardless of the number of robots in the team.

This procedure is yet to be tested in a real scenario and can still be subject to a number of improvements. An obvious improvement is abstracting, when possible, which robot sent the communication. This would be done by having external events that can be fired whenever there is an incoming communication of the occurrence of an event for which the receiving robot does not need to know the id of the robot that executed it to fulfil the specification. By doing this, we would be able to add and subtract elements to the team without the need to change the models and the specifications. Also, this approach can be generalized for PN models with algebraic state description.

5.5 Generalized Mutual Exclusion Constraints

We finish the chapter with a brief remark on an alternative approach for SC of PNs. The first is from a theoretical point-of-view. The most studied and used approach for SC using PN is arguably the so called generalized mutual exclusion constraints (GMEC) approach. There has been many works on the use of this approach, as was mentioned in the section about related work, e.g., [Giua et al., 1992, Iordache and Antsaklis, 2006b, Wu et al., 2002, Iordache and Antsaklis, 2002, Moody and Antsaklis, 1998, Iordache and Antsaklis, 2006a]. The problems we addressed in this section have all been studied for GMECs, and there are methods to build (non least-restrictive) supervisors that ensure admissibility and deadlock freeness. They also address other issues such as unobservable events and liveness, though for some of these cases – particularly the ones related to deadlock prevention and liveness enforcement – the approach can be quite inefficient, in the sense that its complexity is quite high and solutions can only be found for more or less small cases. Contrary to our approach, the GMEC approach is not based on language specifications. It is based on the addition of places – called monitors – that create place invariants in the PN. These monitors allow the designer to write specifications similar to our specifications for the algebraic state description (in the basic form, without taking events into account). In fact, the construction of the counter place we presented here has similarities with the basic construction of monitors. The main differences between the approaches is that, while our approach allows the use of LTL specifications written over linear combinations of the number of tokens in bounded places, the GMEC approach allows writing specifications over

linear combinations of the number of tokens in places that must be satisfied in all the markings of the PN. Loosely speaking, a "GMEC specification" $l^T M \geq b$ is equivalent to $G(l^T M \geq b)$ in our case (assuming that the specifications are only over bounded places). The GMEC approach has the advantage of being able to deal with unbounded places. However, this is the case because, since the idea of GMEC is to always keep the number of tokens in the places inside given bounds, being only applicable when the initial marking already satisfies the linear constraint, there is no need to "check" the number of tokens in the places being controlled. In our case that kind of checking is necessary, because we always need to check what is the marking after the firing of a transition in order to make the BA evolve to the correct state. Thus, since it is known that any extension of the PN model that allows checking an unbounded place for 0 tokens increases the modelling power of the extended PN model to the power of Turing machines (which is not desired, since for Turing machines most non-trivial problems are undecidable), we cannot hope to extend our approach to deal with unbounded places. Even if there is the possibility to deal with unbounded places in some cases – for example the specification $G(l^T M \geq b)$ has the same flavour as a GMEC specification, thus for this case it is possible to build a supervisor – most of the LTL formulas will require some checking on the number of markings in the places it refers to. The approach has also been extended in order to also take into account linear constraints on the firings of transitions. Furthermore, there has been a small reference on how this approach can be used to find admissible supervisors for PN language specifications, but, to the best of our knowledge, the amount of research on this topic is limited to just one small section in [Iordache and Antsaklis, 2006b]. Furthermore, it would require the calculation of $G \parallel G_\varphi$, which is something that we want to avoid. We argue that, the algorithm we presented to directly verify G_φ for admissibility is a better approach in our case.

CHAPTER 6

Case Studies and Results

In this chapter, we present three application scenarios of the methodologies we presented. We will start with a simulated soccer robot team. This is a simple example where the individual models are not very intricate. It is presented with the goal of providing a simple illustration of the methodology and comparing the different approaches we developed in terms of scalability. We will then present implementations on a social robot and on a multi-robot surveillance scenario, which illustrate the application of our methodology to real robot systems. Before describing the scenarios themselves, we present some practical remarks about their implementation.

As we discussed in Section 5.2, there is still considerable work to be done in order to efficiently obtain admissible and deadlock-free PN supervisors. This is a point for future work, however, in order to be able to implement our methodology in concrete scenarios, we took a more pragmatic point-of-view. Given that the state-of-the-art still does not allow one to build admissible and deadlock-free supervisors for complex systems, in order to implement our work in real systems, our approach will be the following:

- In Sections 6.1 and 6.3, will use the extra restriction to the syntactically safe LTL specifications – presented in Section 5.3 – that guarantees supervisor admissibility by construction. It has the drawback of mixing the specification and model levels, which is not always desired. For example, when the writing of the specifications depends directly on the model, any change in it can have a large impact on the specification. However, when the designer feels it is appropriate – for the case of our application scenarios for example, we had no problems using the restriction – the admissibility restriction provides a guarantee that, as we have just discussed, is very computationally expensive to check for complex systems.

- In Section 6.2, the admissibility restriction for the specifications is not used, because at the time

the example was implemented we had not defined the language restriction. In this case, we add an extra rule to the evolution of the supervisor: always enable all uncontrollable events and if the system executes one which is not active in the current state of the supervisor, simply ignore it (i.e., the supervisor does not execute an event). Of course, this might lead to unpredictable behaviours. However, in this particular example, the impact of ignoring non-active uncontrollable events on the final behaviour of the system was not noticeable.

- For deadlock-freeness, one can either try and apply a deadlock freeness algorithm or, as was our choice, proceed with simulations before deployment and check if blocking occurs. If so, the specifications should be rewritten. This is obviously a very naive approach, which does not provide guarantees, but in our cases we managed to get supervisors which never deadlocked, after a small amount of simulation-specification rewrite iterations.

In the application scenarios where we used the language restriction to guarantee admissibility, we could translate our natural language specification to formulas of type 1, 2 or 3 in a intuitive way, so we argue that the restriction already allows the definition of a good array of behaviours. However, our ultimate goal is to continue developing the methodology so that we can eventually drop the use of this language restriction.

In terms of execution, whenever there are active controllable events, we immediately choose one randomly to be executed. When there are only uncontrollable events, we wait until one of them becomes active and, when it does, we fire it accordingly, update the system and supervisors states and check for active controllable events to fire next.

To finalize this overview of the practical implementation of our methodology, we just mention another approach that can be used when dealing with homogeneous multi-robot teams: do the analysis for a small amount of robots (e.g., 3 robots) and, if needed, modify the specifications to guarantee supervisor admissibility and deadlock-freeness. Then, since the addition of more robots only entails the addition of another module similar to the ones already in the model, and a small change in the specifications, it might be possible to guarantee that if we have supervisory admissibility and deadlock-freeness for a given set of specifications, the adding of more robots does not have an impact on these properties. This can be another topic for further work.

6.1 Simulated Soccer Robots

Consider a team of n robots playing a soccer match. The objective is to reach a situation in which one of the robots is close enough to the goal to shoot and score. When a robot does not have the ball in its possession, it has two options:

- Move to the ball until it is close enough to take its possession;
- Get ready to receive a pass from a teammate.

When a robot has the possession of the ball, it can:

- Shoot the ball (if it is close enough to the goal);
- Take the ball to the goal, if it is far from the goal and there is no opponent blocking its path;
- Choose a teammate to pass the ball and, when it is ready to receive, pass it.

For simplicity, we assume that, when a robot shoots the ball, the team loses its possession (we do not differentiate the situation where the robot scores from the situation where the robot does not score since the team will lose the ball possession in both) and that the opponents do not steal the ball (they are only able to block paths, at which point our robot will try to pass to a teammate).

Figure 6.1 depicts a possible FSA system model for a robot i situated in a soccer match, where we describe the states by stating if the robot is moving to the ball and if the robot has the ball in its possession. Hence, the state description symbol set is $D_i = \{moving2getball_i, hasball_i\}$ and:

- $\mu_i(without\_ball_i) = \mu_i(receiving\_ball_i) = \emptyset$;
- $\mu_i(moving\_to\_ball_i) = \mu_i(preparing\_to\_get\_ball_i) = \{moving2getball_i\}$;
- $\mu_i(with\_ball_i) = \mu_i(preparing\_to\_kick_i) = \mu_i(moving\_to\_goal_i) = \mu_i(choosing\_receiver_i) = \mu_i(passing\_ball_i) = \{hasball_i\}$.

The events $close\_to\_ball_i$, $close\_to\_goal_i$ and $blocked\_path_i$ are caused by changes in the environment around the robots and not by the robots themselves. Therefore, they are considered uncontrollable events. The controllable events correspond to the actions available to each robot.

- $E_{c,i} = \{move\_to\_ball_i,\ get\_ball_i,\ move\_to\_goal_i,\ kick\_ball_i,\ start\_passing_{i,j},\ start\_receiving_i,\ pass_{i,j} \mid i,j = 1,...,n,\ j \neq i\}$
- $E_{uc,i} = \{close\_to\_ball_i,\ blocked\_path_i,\ close\_to\_goal_i \mid i = 1,...,n\}$

An FSA system model for the whole team is given by the parallel composition of the FSA models for each robot. Note that the $start\_passing_{i,j}$, $pass_{i,j}$ and $pass_{j,i}$ labels in the model for robot i are in reality representing a set of events $\{start\_passing_{i,j} \mid j = 1,...,n, j \neq i\}$, $\{pass_{i,j} \mid j = 1,...,n, j \neq i\}$ and $\{pass_{j,i} \mid j = 1,...,n, j \neq i\}$ respectively. Also, each event $pass_{i,j}$ is a shared event, i.e., it must be synchronized between robot i (the passing robot) and robot j (the receiving robot).

In Figure 6.2, we present the PN model for one of the robots, which is a direct translation to a PN of the FSA defined for above.

The state propositional description is according to the following rules:

- $moving2getball_i$ is $true$ when there is a token on places $moving\_to\_ball$ or $preparing\_to\_get\_ball$;
- has_ball_i is $true$ when there is a token on $with\_ball$, $preparing\_to\_kick$, $moving\_to\_goal$, $choosing\_receiver$ or $passing\_ball$.

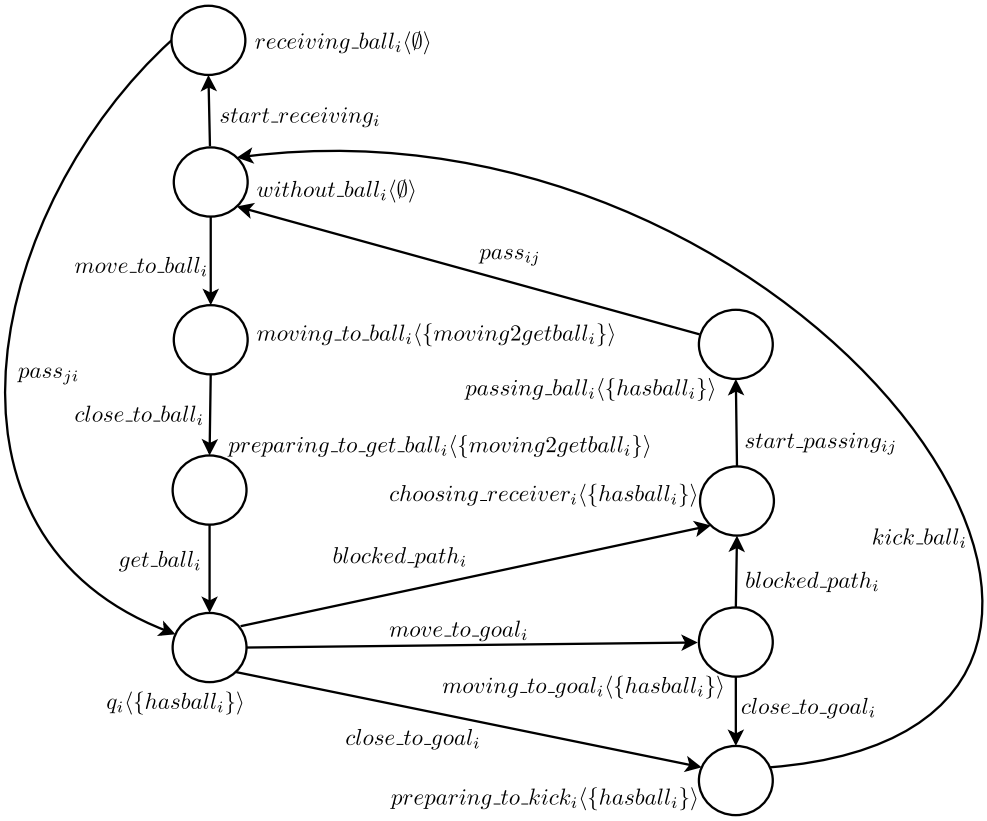


Figure 6.1: FSA model for robot *i*

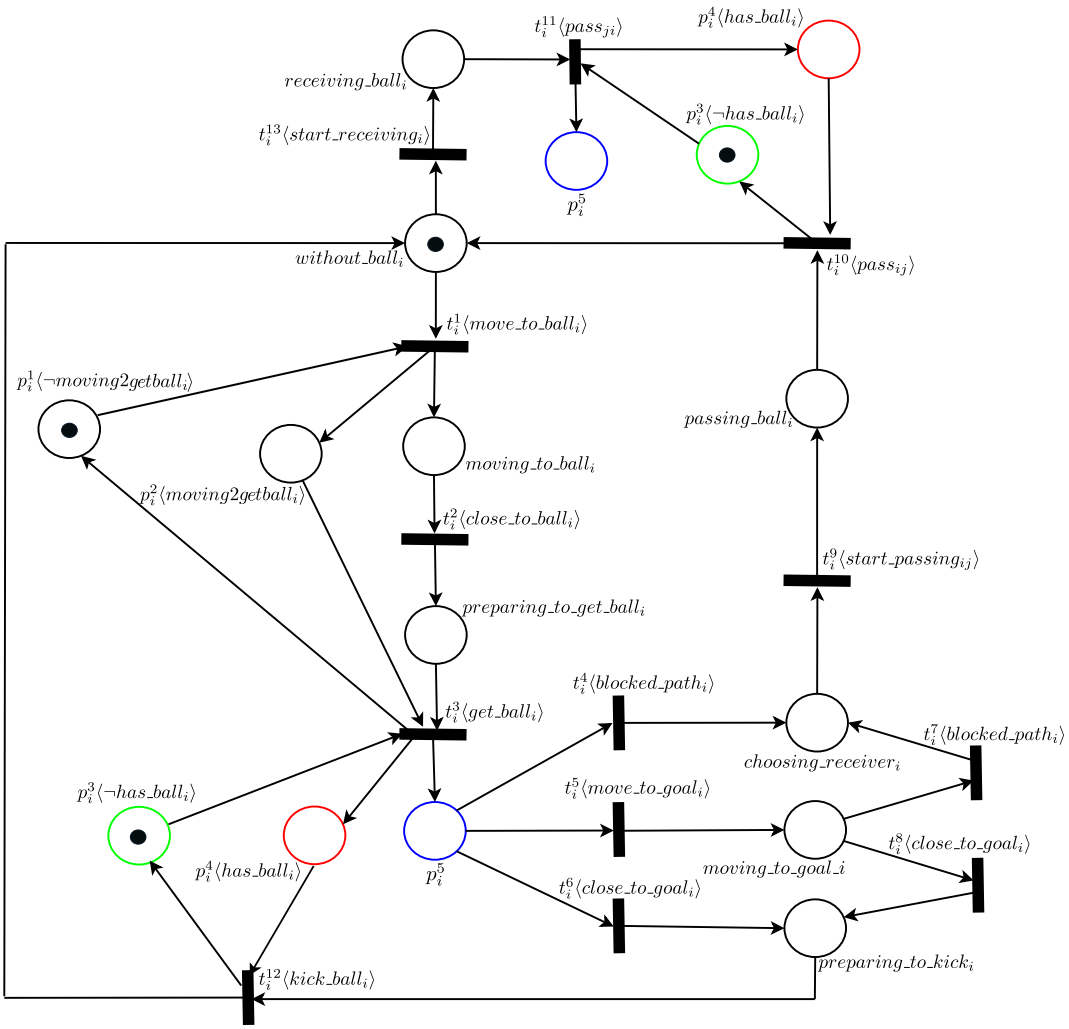


Figure 6.2: PN model for robot i. Places with the same color represent the same place, we separated them to improve readability.

We also add the following facts to K:

- When a robot is moving to get the ball it does not have the ball;
- When a robot has the ball, it is not moving towards it.

Hence, we have:

$$K = \{(moving2getball_i, \neg has\_ball_i), (has\_ball_i, \neg moving2getball_i) \mid i = 1, ..., n\} \tag{6.1}$$

A PN model for the centralized system is given by the parallel composition of the PN models of each robot. One may define the following specifications, which are useful to improve the team's performance:

- For the whole team, a robot will move to the ball if and only if the ball is not in the team's possession and no other teammate is moving towards it. This specification is written in a different way, whether we are using the symbolic state description or the algebraic state description. for the symbolic state description, we have:

$$\varphi = G((\bigvee_{i=1}^{n}(moving2getball_i \vee hasball_i)) \Rightarrow (X(\bigwedge_{i=1}^{n} \neg move\_to\_ball_i))) \tag{6.2}$$

For the algebraic state description, we can re-write the disjunction in the first half of the formula as a constraint on the sum of tokens in the corresponding places, yielding the following specification:

$$\varphi = G((\sum_{i=1}^{n}(M(p_i^2)+M(p_i^4))) \geq 1 \Rightarrow (X(\bigwedge_{i=1}^{n} \neg move\_to\_ball_i))) \tag{6.3}$$

As we will see, the fact that we can re-write the disjunction as a linear constraint on the number of tokens in a set of places will greatly reduce the size of the supervisors.

- For each robot i, it will not get ready to receive a pass if none of its teammates wants to pass it the ball:

$$\psi_i = G((\bigwedge_{\substack{j=1 \\ j\neq i}}^{n} \neg start\_passing_{j,i}) \Rightarrow (X\neg start\_receiving\_i)) \tag{6.4}$$

- For each robot i, when one of the teammates decides to pass it the ball, it will be ready to receive the pass as soon as possible:

$$\gamma_i = G((\bigvee_{\substack{j=1 \\ j\neq i}}^{n} start\_passing_{j,i}) \Rightarrow (X((\bigwedge_{e' \in E_c^i \setminus \{start\_receiving_i\}} \neg e')W start\_receiving_i))) \tag{6.5}$$

Note that in Figure 1.1, the fragment of the supervisor depicted is obtained by the same reasoning as in Example 4.2.8, analysing the firing of t_1^2 when the Büchi automaton is in state x (this analysis will create 3 more transitions similar to the one depicted in yellow in the figure, but we omit them to improve readability).

Hence, we will build $2n+1$ supervisors, one for formula φ, which deals with the team as a whole, one for each formula ψ_i and one for each formula γ_i.

We ran simulations for this example, for 3 robots, for both FSA and PN. As expected the behaviour is the same in both versions. In simulation 1, we emphasize the role of formula φ - only 1 robot goes to the ball at each time - and in simulation 2 we emphasize the roles of formulas ψ_i and γ_i - each robot only gets ready to receive a pass immediately after one of its teammates decides to pass it the ball . In Simulation 3 we show the uncontrolled behaviour of the system. The lack of restrictions imposed for this system allows it to regularly evolve to a *deadlock* situation.

Simulation 1 - *move_to_ball*$_1$ - *close_to_ball*$_1$ - *get_ball*$_1$ - blocked_path$_1$ - start_passing$_{13}$ - start_receiving$_3$ - pass$_{13}$ - move_to_goal$_3$ - blocked_path$_3$ - start_passing$_{31}$ - start_receiving$_1$ - pass$_{31}$ - blocked_path$_1$ - start_passing$_{13}$ - start_receiving$_3$ - pass$_{13}$ - move_to_goal$_3$ - close_to_goal$_3$ - *kick_ball*$_3$ - *move_to_ball*$_2$ - *close_to_ball*$_2$ - *get_ball*$_2$ - blocked_path$_2$ - start_passing$_{23}$ - start_receiving$_3$ - pass$_{23}$ - blocked_path$_3$ - start_passing$_{32}$ - start_receiving$_2$ - pass$_{32}$ - close_to_goal$_2$ - *kick_ball*$_2$ - *move_to_ball*$_1$ - *close_to_ball*$_1$ - *get_ball*$_1$ - close_to_goal$_1$ - *kick_ball*$_1$ - *move_to_ball*$_2$ - *close_to_ball*$_2$ - *get_ball*$_2$ - close_to_goal$_2$ - kick_ball$_2$ - *move_to_ball*$_3$ - *close_to_ball*$_3$ - *get_ball*$_3$ - move_to_goal$_3$ - blocked_path$_3$ - start_passing$_{31}$ - start_receiving$_1$ - pass$_{31}$ - close_to_goal$_1$ - kick_ball$_1$ - ...

Simulation 2 - move_to_ball$_3$ - close_to_ball$_3$ - get_ball$_3$ - move_to_goal$_3$ - close_to_goal$_3$ - kick_ball$_3$ - move_to_ball$_1$ - close_to_ball$_1$ - get_ball$_1$ - blocked_path$_1$ - *start_passing*$_{12}$ - *start_receiving*$_2$ - *pass*$_{12}$ - blocked_path$_2$ - *start_passing*$_{21}$ - *start_receiving*$_1$ - *pass*$_{21}$ - move_to_goal$_1$ - blocked_path$_1$ - *start_passing*$_{13}$ - *start_receiving*$_3$ - *pass*$_{13}$ - move_to_goal$_3$ - close_to_goal$_3$ - kick_ball$_3$ - move_to_ball$_3$ - close_to_ball$_3$ - get_ball$_3$ - move_to_goal$_3$ - blocked_path$_3$ - *start_passing*$_{31}$ - *start_receiving*$_1$ - *pass*$_{31}$ - close_to_goal$_1$ - kick_ball$_1$ - move_to_ball$_1$ - close_to_ball$_1$ - get_ball$_1$ - blocked_path$_1$ - *start_passing*$_{13}$ - *start_receiving*$_3$ - *pass*$_{13}$ - blocked_path$_3$ - *start_passing*$_{31}$ - *start_receiving*$_1$ - *pass*$_{31}$ - move_to_goal$_1$ - blocked_path$_1$ - *start_passing*$_{12}$ - *start_receiving*$_2$ - *pass*$_{12}$ - close_to_goal$_2$ - kick_ball$_2$ - move_to_ball$_3$ - close_to_ball$_3$ - get_ball$_3$ - ...

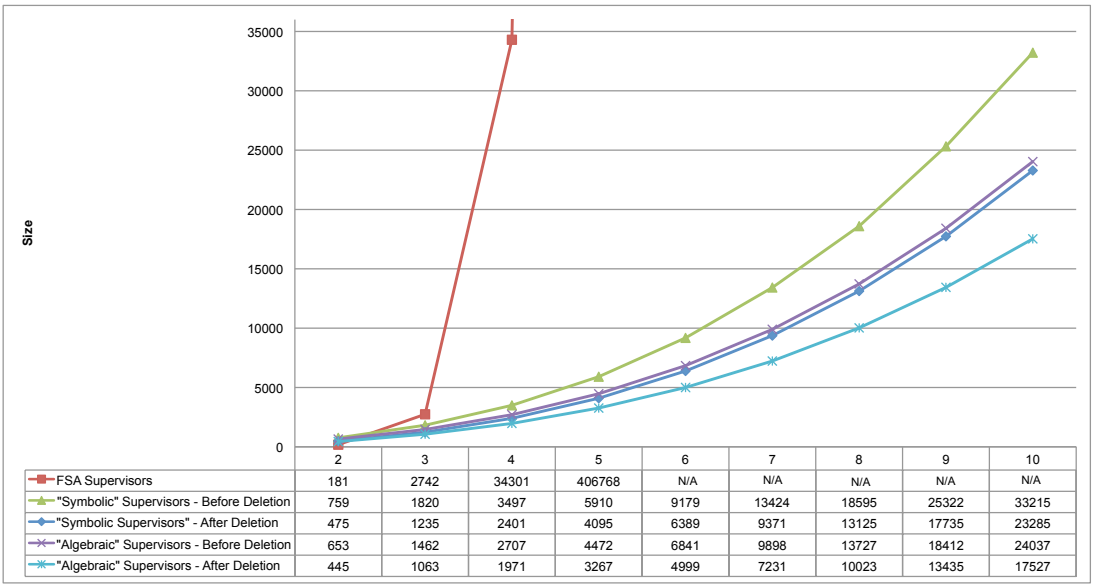
Simulation 3 - start_receiving$_3$ - move_to_ball$_1$ - move_to_ball$_2$ - close_to_ball$_2$ - close_to_ball$_1$ - get_ball$_1$ - blocked_path$_1$ - start_passing$_{12}$ - get_ball$_2$ - blocked_path$_2$ - pass$_{13}$ - move_to_ball$_1$ - move_to_goal$_3$ - start_passing$_{21}$ - close_to_goal$_3$ - close_to_ball$_1$ - get_ball$_1$ - kick_ball$_3$ - close_to_goal$_1$ - kick_ball$_1$ - move_to_ball$_3$ - close_to_ball$_3$ - move_to_ball$_1$ - get_ball$_3$ - blocked_path$_3$ - start_passing$_{32}$ - close_to_ball$_1$ - get_ball$_1$ - move_to_goal$_1$ - blocked_path$_1$ - start_passing$_{12}$

A video of the simulations ran in matlab can be found at `http://users.isr.ist.utl.pt/~blacerda/SocRob%202.wmv`. In Figure 6.3, we show the size (defined as the number of states for FSA, and the sum of the number of places and transitions for PNs) of the sum of the sizes of all the supervisors before deleting the dead transitions and the sum of the sizes of all the supervisors after deleting the dead transitions, ranging from a team of 2 to a team of 10 robots.

We can see that the size of the supervisors in the FSA version grows very fast, making it only possible to calculate the supervisors for 5 robts. For the PN case the rate of growth is much less. In spite of not using a complete method to delete dead transitions, we are able to efficiently reduce the size of the supervisors. Furthermore, we can see that the impact of the use of the algebraic state description is significant. This is due to the fact that the number of transitions in the BA is greatly reduced when we do not have disjunctions on the LTL specifications.

To apply the decentralized version, we need to substitute formula φ, because it is defined for the whole system, and in this case formulas must be written for specific robots. So, we define the following specifications instead of φ:

- For each robot *i*, only move to the ball if the ball is not in the team's possession and no other



	2	3	4	5	6	7	8	9	10
FSA Supervisors	181	2742	34301	406768	N/A	N/A	N/A	N/A	N/A
"Symbolic" Supervisors - Before Deletion	759	1820	3497	5910	9179	13424	18595	25322	33215
"Symbolic Supervisors" - After Deletion	475	1235	2401	4095	6389	9371	13125	17735	23285
"Algebraic" Supervisors - Before Deletion	653	1462	2707	4472	6841	9898	13727	18412	24037
"Algebraic" Supervisors - After Deletion	445	1063	1971	3267	4999	7231	10023	13435	17527

Figure 6.3: Size of the FSA supervisors (number of states) and the PN supervisors (number of places plus number of transitions) – before and after deleting dead transitions.

teammate is moving towards it:

$$\varphi_i = G(\bigvee_{\substack{j=1 \\ j\neq i}}^{n} (moving2getball_j \vee has\_ball_j) \Rightarrow (X\neg move\_to\_ball_i)) \tag{6.6}$$

The other specifications are already written for a given robot, so they remain the same. In this case, we will build 3 supervisors for each robot, 1 per specification. Figure 6.4 shows the size (defined as the sum of the number of places and transitions) of sum of the sizes of the supervisors for the decentralized approach, and the centralized approach for PN with symbolic state description, ranging from a team of 2 to a team of 10 robots.

We can see that, as expected, the decentralized approach yields smaller supervisor realizations, enabling us to handle bigger systems. In fact, given the simplicity of this example, we were able to find a closed formula for the size of the PN supervisors for the symbolic state representation. For the centralized version, we have a cubic growth on the number of robots, while in the decentralized approach we have a quadratic growth on the number of robots.

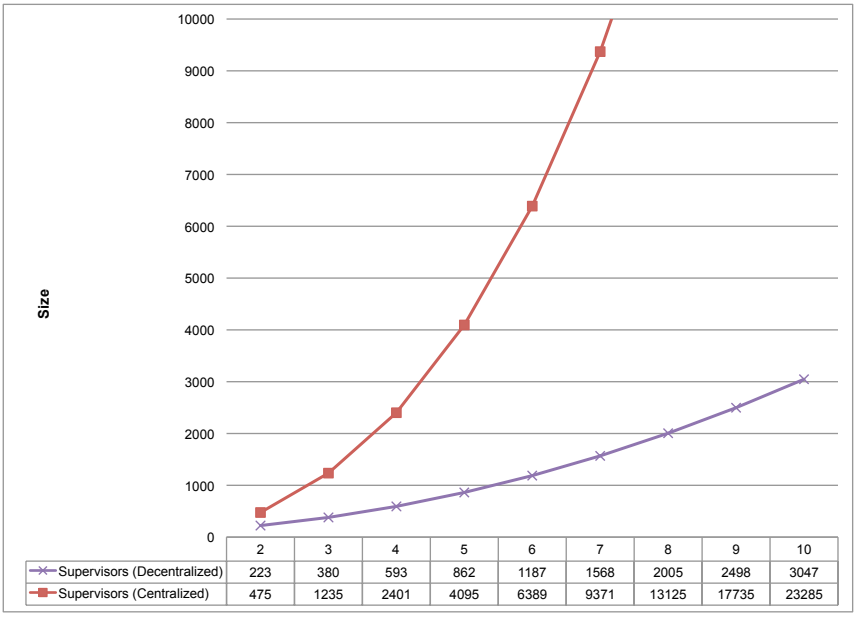


Figure 6.4: Size (sum of number of places and number of transitions) of the supervisors to be run in each robot.

6.2 Maggie, The Social Robot

The method was also implemented in *Maggie*, a social robot developed at the Robotics Lab in Universidad Carlos III de Madrid[1]. A complete description of the robot is provided in [Salichs et al., 2006]. The implementation of our method in Maggie is fully described in [Lacerda et al., 2011]. We will give a brief overview of the robot's features used in our implementation and present the PN models used for the actuators and sensors.

Maggie, depicted in Figure 6.5, is a social robot developed at the RoboticsLab of Universidad Carlos III de Madrid to interact mainly with children. In this section, we provide a brief description of its capabilities, further details can be found in [Salichs et al., 2006].

Maggie is a 1.35m tall girl-like doll. Her base is equipped with two differentially driven wheels and a caster wheel on each side. The arms and the eyelids have 1 DOF: up/down, while the neck has 2 DOF: up/down and left/right. She also possesses tactile sensors, including on the shoulders and on top of her head. These are the sensors and actuators that we took into account in our implementation. Maggie possesses many other capabilities, such as infrared and ultrasound sensors used for navigation, a color camera for people tracking and a mouth shape with invisible web-cam and coloured lights synchronized with the speech. These capabilities result in a platform well suited to study human-

[1]This implementation was done before we defined the language restrictions to guarantee supervisor admissibility. The LTL specifications used in the implementation do not satisfy the restrictions, but it is possible to build "restricted" formulas that yield a similar behaviour. Furthermore, we circumvented the admissibility problems by having the supervisors ignore the occurrence of uncontrollable events which are not active in their current state. While not theoretically sound, this approach was successful in this case.

Figure 6.5: Maggie, the social robot, interacting with a child.

robot interaction and robot learning by training and teaching.

The control architecture is based on the automatic-deliberative (AD) control architecture proposed in [Barber and Salichs, 2001]. In this architecture, the robot skills are divided in two levels. In the automatic level, we find the skills related with robot sensors and actuators. In the deliberative level, we find higher-level skills, such as a planner or our implementation of the feedback-loop of SC. The skill we developed subscribes to events representing changes in sensor readings, for which an event handler is implemented. Also, it can order the execution of skills related to performing simple actions (e.g., raising an arm or start spinning).

We will use PNs to model Maggie behaving freely in the environment. In the PN models used here both places and transitions have a specific interpretation:

- *Places* represent the value of binary variables that are used to define the state of the system, i.e., all places correspond to the truth value of some state description symbol;
- *Transitions* represent orders to execute actions or changes in sensor readings, i.e., the firing of a transition represents a communication between our implementation and the automatic level of the architecture.

In Figure 6.6 we depict the PN model for Maggie. Contrary to the other examples, where the individual models already have some structure, representing the basic behaviour of each robot, and we use LTL to specify coordination rules for the team, in this case we start with a very unstructured model, where the robot can perform any of its available actions randomly, and use the LTL formulas to restrict the available actions so that a given behaviour is achieved.

We note that, for the base, when a *start* action is issued, the robot starts performing that action

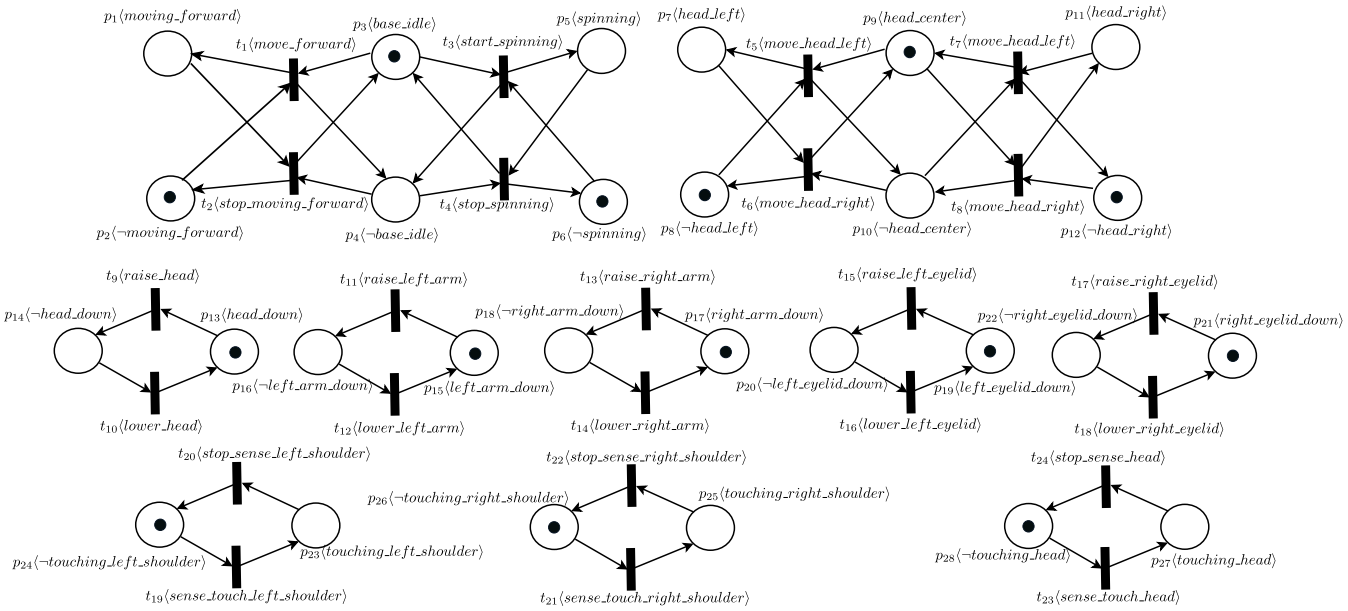


Figure 6.6: The PN model for Maggie's different actuators and sensors.

until a *stop* action is issued, that is, the action continues until it is explicitly stopped. For the other models, we assume that the actuator moves for a fixed amount and then stops, hence no *stop* action is required.

We implemented a *Petri net executor* in C++, so that the robot is able to run its system PN. We start by dividing the transitions into transitions corresponding to actions - t_1 to t_{18} - and transitions corresponding to changes in sensor readings - t_{19} to t_{24}. The implementation is a loop that, in each step, randomly selects one of the active transitions corresponding to actions and fires it, ordering the execution of the corresponding action and updating the marking. Also, when one of the sensors changes state, the handler interrupts the loop, and the transition corresponding to that sensor change is immediately fired and the marking is updated. By running this PN without supervising it, Maggie simply executes random actions, displaying an unrestricted behaviour.

We use LTL to specify 3 different behaviours for Maggie. The first behaviour is a simple sequence triggered by touching the head of the robot. The sequence can be stated in natural language as "*When the head is touched raise the left arm, then raise and lower the right arm. When the head stops being touched, lower the left arm*". This sequence can be translated into the following simple rules:

- The left arm must only be raised when the head is being touched:

$$G(touching\_head \Leftrightarrow (X(\neg left\_arm\_down))) \tag{6.7}$$

- The right arm must only be raised after the left arm is raised:

$$G(raise\_left\_arm \Leftrightarrow (X raise\_right\_arm)) \tag{6.8}$$

- After raising the right arm, it must be immediately lowered again:

$$G(raise\_right\_arm \Rightarrow (X lower\_right\_arm)) \tag{6.9}$$

The second behaviour shows how to specify different reactions to different sensor readings. We can state it has "*If the left shoulder is touched, wait until the right shoulder is touched and then raise your right arm. If the right shoulder is touched, wait until the left shoulder is touched and raise your left arm. After raising an arm, lower it again and return to the initial state*". To keep track of which shoulder is touched, we use the state of the eyelids. If both eyelids are down, then no shoulder has been touched yet. If one of the eyelids is up, then the corresponding shoulder was touched and Maggie is waiting for the other shoulder to be touched. Hence, the behaviour can be implemented by the following rules:

- If the left shoulder is touched and both eyelids are down (i.e., no shoulder was touched yet), raise the left eyelid:

$$G((sense\_left\_shoulder \wedge right\_eyelid\_down \wedge \ left\_eyelid\_down) \Leftrightarrow (X raise\_left\_eyelid)) \tag{6.10}$$

- If the right shoulder is touched and the left eyelid is up (i.e., the left shoulder was previously touched), raise the right arm:

$$G((sense\_right\_shoulder \wedge \neg left\_eyelid\_down) \Leftrightarrow \ (X raise\_right\_arm)) \tag{6.11}$$

- After raising the right arm, return to the initial state by lowering the left eyelid and the right arm:

$$G(raise\_right\_arm \Leftrightarrow (X lower\_left\_eyelid)) \tag{6.12}$$

$$G(lower\_left\_eyelid \Leftrightarrow (X lower\_right\_arm)) \tag{6.13}$$

- If the right shoulder is touched and both eyelids are down (i.e., no shoulder was touched yet), raise the right eyelid:

$$G((sense\_right\_shoulder \wedge right\_eyelid\_down \wedge \ left\_eyelid\_down) \Leftrightarrow (X raise\_right\_eyelid)) \tag{6.14}$$

- If the left shoulder is touched and the right eyelid is up (i.e., the right shoulder was previously

touched), raise the left arm:

$$G((sense\_left\_shoulder \wedge \neg right\_eyelid\_down) \Leftrightarrow \ (X raise\_left\_arm)) \tag{6.15}$$

- After raising the left arm, return to the initial state by lowering the right eyelid and the left arm:

$$G(raise\_left\_arm \Leftrightarrow (X lower\_right\_eyelid)) \tag{6.16}$$

$$G(lower\_right\_eyelid \Leftrightarrow (X lower\_left\_arm)) \tag{6.17}$$

Note that in this behaviour, touching the same shoulder more than one time in a row does influence the arm to be raised. After touching one of the shoulders, Maggie ignores all other sensor readings until the other shoulder is touched. When that happens, she raises the corresponding arm.

The third behaviour is also triggered by touching the head, but we allow the robot to perform random actions during the behaviour. It can be stated as "*Move arms randomly. When the head is touched, start spinning until both arms are up*". This can be translated into the following rules:

- Only start spinning when the head is touched, you are not spinning yet and both arms are not already up:

$$G((sense\_head \wedge (\neg spinning) \wedge \neg(\neg left\_arm\_down \wedge \neg right\_arm\_down)) \Leftrightarrow (X start\_spinning)) \tag{6.18}$$

- After starting to spin, continue spinning until both arms are up:

$$G(start\_spinning \Rightarrow (X(spinning U(\neg left\_arm\_down \wedge \neg right\_arm\_down)))) \tag{6.19}$$

- When both arms are up, stop spinning (or continue stopped if you are not spinning):

$$G(\neg left\_arm\_down \wedge \neg right\_arm\_down) \Rightarrow (X \neg spinning)) \tag{6.20}$$

All of these examples also include an additional formula which avoids the execution of the actions that are not referred to in the specification, i.e., in each example, a formula of the form $G\left(\bigwedge_{e \in E_o} \neg e\right)$, where E_o is the set of actions that is not mentioned in that example is also added.

In the video available at `http://bit.ly/dQqVQK`, we show both the uncontrolled behaviour of Maggie and its behaviour when being supervised by the PNs obtained from the specifications described here.

Figure 6.7: The e-puck robot (re-printed from [Mondada et al., 2009]).

6.3 Surveillance Scenario with E-Pucks

In this section, we describe the larger scale implementation of out methodology. This was an implementation of a surveillance scenario in a team of real mobile robots. We will start by discussing the scenario implementation details, and then show the LTL rules used for the coordination.

6.3.1 Scenario Description

We implemented this scenario using the e-puck robots [Mondada et al., 2009]. The e-puck (Figure 6.7) is a small circular, differential driven robot, with $7cm$ of diameter, $6cm$ of height and $660g$, developed at the École Polytechnique Fédérale de Lausanne.

More details about the robot can be found in its homepage `http://www.e-puck.org/`. Given that our goal was to test the high-level LTL specifications, we only used the e-pucks for navigation, using an overhead camera for both robot localization and event detection. Before describing the set-up, we show the scenario, which is a $2.1m \times 1.6m$ rectangular representation of a floor, composed of an inner corridor and 10 rooms, as depicted in Figure 6.8.

The goal is to have a team of 4 e-pucks doing surveillance in the scenario, looking for 3 different anomalous situations, which we will represent by different coloured rectangles in the rooms[2]:

1. A dirty room, represented by a blue rectangle;
2. An abandoned object, represented by a green rectangle;
3. A fire, represented by a red rectangle.

As we already stated, we use an overhead *AXIS P13* camera for both individual robots localization and room events detection. Furthermore, we use a central computer to process the images, run the PN supervisors and send move commands to the e-pucks, by Bluetooth wireless communications. Figure 6.9 depicts a diagram of the different modules used.

We give a brief overview on each of the modules. We used the robot operating system (ROS) to manage the camera drivers and the messages passed between different modules:

[2]We assume that the anomalous situations can only occur inside the rooms.

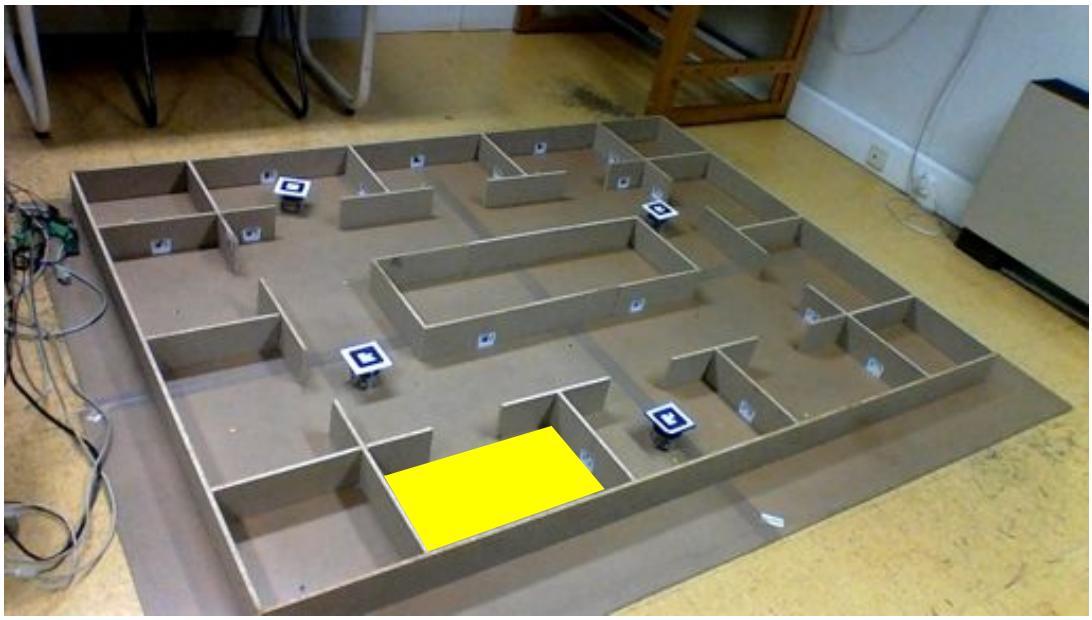


Figure 6.8: The scenario with the 4 e-pucks

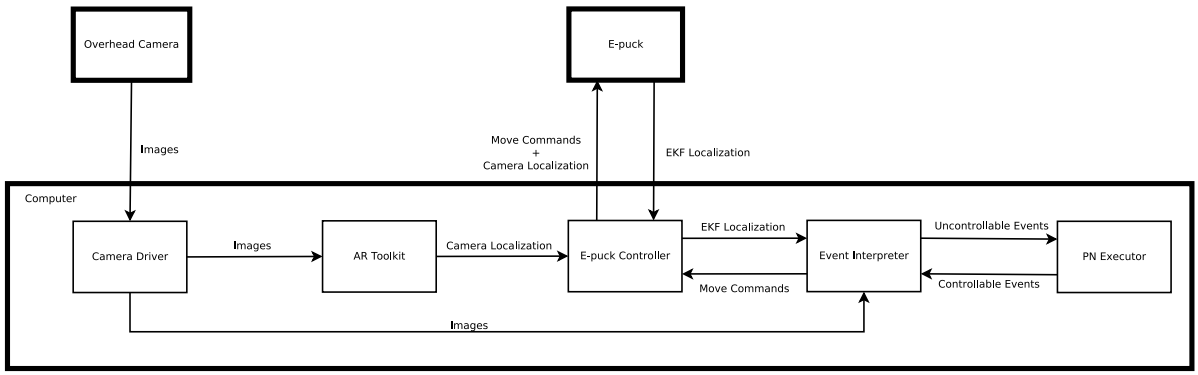


Figure 6.9: Diagram of the scenario implementation

- The *AXIS P13* camera retrieves images of the whole scenario and sends them to the central computer by cable Ethernet;

- In the computer, the ROS *camera driver* receives the image stream and publishes it in a ROS topic;

- In the computer, *ARToolKit*[3] [Kato and Billinghurst, 1999] subscribes to the image topic and is used to get a 3D position for each robot, which is translated to 2D via a simple projection. To use ARToolkit, we placed a small marker on top of each e-puck, which is used both to identify the robot and to find its orientation. The values (x, y, θ) for each robot are then published in another ROS topic;

- In the computer, the *e-puck controller* subscribes to the topic published by ARToolKit, and also receives move and stop commands from the PN executor. Both the camera positions and the commands are sent to the e-pucks via bluetooth. Furthermore, the e-puck controller also receives, via Bluetooth, updated positions from the e-pucks and publishes them in a ROS topic;

- The *e-pucks* execute the move commands received from the e-puck controller and run an extended Kalman filter (EKF) using their odometry and the ARToolkit localization data to update their localization. This approach is used to mitigate camera localization errors, the camera delay, and to reduce the communication frequency between the central computer and the e-pucks, thus avoiding latency in the Bluetooth network;

- The *event interpreter* subscribes to the e-puck localizations topic and the image topic, and uses its information to compute the uncontrollable events that occurred. The image topic is used to fire the room events (using a simple color detection algorithm) and the e-puck localization is used for all other uncontrollable events, which have a relation to the e-puck positions. For example, if robot i is inspecting a room and the camera detected a blue rectangle in the room corresponding to the position of robot i, then uncontrollable event $found\_dirty\_room_i$ is fired. It also receives controllable events from the PN executor and transforms them into move commands to be sent for the robot. For example, if the fired event is $continue\_in\_corridor_i$, it computes the point in front of the next door and creates a move command to be sent for robot i, with that position as the goal.

- The *PN executor* is used to run the PN supervisors. It receives the uncontrollable events from the event interpreter and fires them immediately, updating the states of the supervisors, and also randomly chooses controllable events enabled by all supervisors to be fired, also updating the supervisors states. When all the controllable events are disabled, it waits for a new uncontrollable event to be fired and change the PN supervisors state.

[3]The choice for ARToolKit was due to its integration in ROS.

The overhead camera detects the room events. However, we simulate that these detections are made by the robots. In fact, initially we started by using the e-puck on-board cameras for color detection, but this approach was less reliable, and also made the occurrence of these events harder to understand in the videos. This happened because we needed to put the rectangles standing in front of the e-pucks, which did not make them very visible in the overhead camera video. Thus, a room event is only detected when a robot is in the room where that event has occurred.

6.3.2 PN Models and LTL Specifications

We now proceed to the high-level task implementation details. Our decision for this example was to modularly model the individual robot basic behaviours using PNs with algebraic state description, and to use LTL to specify in which situations a given behaviour should be executed. We start by describing the individual robot possible behaviours. The PN models for these behaviours are presented in Appendix A. Each robot is able to:

- Move (counter-clockwise) around the corridor, stopping whenever a new room door is found;
- When in front of a room door, randomly decide whether to enter room or continue;
- Go (directly) to a special security room – the room in yellow in Figure 6.8;
- When inside the special security room, leave it;
- Go towards a teammate position;
- When a dirty room is found, clean it;
- When an abandoned object is found, pick it up;
- When holding an object, drop it;
- When a fire is detected, extinguish it;
- When a room is clear, leave it.

We will simulate some of these behaviours in the following way:

- While cleaning a room, the robot simply stands still until the room is clean (i.e., the blue rectangle is removed) or it decides to stop cleaning the room;
- Picking an object just changes the PN marking, the robot does not do anything. Whenever a robot picks up an object, we will remove the green rectangle
- When a robot starts extinguishing a fire, it moves to one of the sides of the room, if one is available (to allow other robots to enter the room). It then simply stands still until the fire is extinguished (i.e., the red rectangle is removed) or it decides to stop extinguishing the fire.

In the video showing a run of the system we will use color codes on top of the robots to allow the viewer to understand what each robot is doing.

Note that the first two behaviours are the "default" behaviours for the robot, and are executed whenever none of the other behaviours is being executed. Also, while some of the behaviours, such as extinguishing a fire, can only occur when a fire is detected, initially we allow the robots to randomly choose to go to the special security room or towards a teammate. The LTL formulas will be used to specify when these behaviours should be executed, and when they should be stopped. The list of natural language specifications we want the system to fulfil, and its translations to LTL formulas[4] is the following:

- *Spread around the scenario*: For each robot i, if it just passed in front of a room that has another robot inside, it must not enter other rooms until it passes in front of at least 2 rooms without entering or it starts moving towards a teammate[5]:

$$G(room\_occupied_i \Rightarrow (X(\neg start\_entering\_room_i W \\ (M(rooms\_passed_i) \geq 2 \vee \bigvee_{j \neq i} start\_going\_towards\_teammate_{j,i})))) \tag{6.21}$$

- *Avoid spending too much time without checking rooms*: For each robot i, if it passes in front of 2 or more rooms without entering, and the current room is not occupied, do not continue in the corridor:

$$G(M(rooms\_passed_i) \geq 2 \Rightarrow (X \neg continue\_in\_corridor_i)) \tag{6.22}$$

- For each robot i, only go to the safe room if it is holding a bag:

$$G(\neg holding\_bag_i \Rightarrow (X \neg start\_going\_to\_safe\_room_i)) \tag{6.23}$$

- For each robot i, only drop a bag you are holding if it is in the safe room:

$$G(\neg in\_safe\_room_i \Rightarrow (X \neg drop\_bag_i)) \tag{6.24}$$

- For each robot i, never leave the safe room if it is holding a bag:

$$G(holding\_bag_i \Rightarrow (X \neg start\_leaving\_safe\_room_i)) \tag{6.25}$$

[4]To facilitate the reading of the formulas, for places with bound 1, we will write p instead of $M(p) \geq 1$ (which is equivalent to $M(p) = 1$, i.e., "p is satisfied") and $\neg p$ instead of $M(\overline{p}) \geq 1$, i.e., "$\overline{p}$ is satisfied".

[5]We also reason about the robot starting moving towards a teammate in this case because moving towards a teammate implies entering the room where the teammate is. Thus, if we do not include this condition and the robot starts moving towards a teammate after passing in front of an occupied room, it will not be able to enter the room where the teammate is, because it would falsify the specification.

- If robot i is not next to a fire, then no other robots should go towards it:

$$G(\neg next\_to\_fire_i \Rightarrow (X \bigvee_{j \neq i} \neg start\_going\_towards\_teammate_{i,j})) \tag{6.26}$$

- If there is not a fire, then the robots should not stop cleaning dirty rooms:

$$G(\neg fire \Rightarrow (X \bigwedge_{i} \neg stop\_cleaning\_room_i)) \tag{6.27}$$

- For each robot i, if it is next to a fire, then the other robots should not stop going towards it:

$$G(\neg next\_to\_fire_i \Rightarrow (X \bigvee_{j \neq i} \neg start\_going\_towards\_teammate_{i,j})) \tag{6.28}$$

- For each robot i, no more than two teammates should move towards it (there can be a maximum number of 3 robots in each room, due to space constraints)

$$G \sum_{j \neq i} M(going\_towards\_teammate_{i,j}) \leq 2 \tag{6.29}$$

- There should not be more that one robot going towards the safe room at each time (to avoid collisions between robots in that room)

$$G \sum_{i} M(going\_to\_safe\_room_i) \leq 1 \tag{6.30}$$

A video showing a run of the robots that satisfies this set of rules is available at `https://dl.dropboxusercontent.com/u/8409501/output.avi`. Note that one can change the behaviour of the team easily by simply changing some of these rules. Thus makes the addition of new behaviours easy. For example, if we assume that the robots cannot extinguish fires, one could simply delete all the "fire related" rules and allow the robot to go to the security room to warn a human also when it detects a fire.

CHAPTER 7

Conclusions and Further Work

7.1 Conclusions

In this thesis, we described methodologies to build an FSA or PN realization of a supervisor that is guaranteed by construction to restrict the behaviour of an uncontrolled system (modelled as an FSA or a PN, respectively) so as to fulfil an LTL specification. The similarities between natural language and temporal logic, and the fact that we can write formulas over both the events of the system and a set of atomic propositions used to describe the states, allow us to build supervisors that realize intricate/complex behaviours that are guaranteed by construction to be consistent with the temporal logic specification. These methods provide the designer with a framework that requires him/her to:

1. Model the system as an FSA or PN;
2. Specify the intended behaviour for the system as a set of LTL formulas.

FSA and PN are two suitable, well known and well accepted frameworks to model robot tasks. Also, LTL is a suitable, well known and well accepted formalism to reason about properties of a system (hence, to synthesize supervisors that fulfil those properties). Therefore, the combination of the two leads to methodologies that reduce design errors, and enable the implementation of larger and more complex systems, from the modular composition of simple models and specifications.

We illustrated the application of the methodologies using a simulated multi-robot scenario, an implementation on a real social robot and a real multi-robot surveillance scenario. From these experiments, we draw the following conclusions:

- the use of LTL (written over the set of events plus symbols describing the state of the world) as the specification language enables one to bridge the gap between a natural language specification for a robot system and a supervisor that enforces it;

- the use of FSA models induces very large supervisors for multi-robot systems, due to the need to enumerate all the possible states of a heavily distributed system;
- PN models are more suited for the modelling of multi-robot systems, due to the distributed nature of PN state representation (i.e., markings). The advantage of using PNs is twofold, facilitating the task of the designer and also inducing smaller supervisors, thus improving the computational efficiency of the method;
- a further improvement to the computational efficiency of the methodologies described here is defining a decentralized version, where we simply augment the PN models of each robot such that they are able to handle communication between each other.

These experiments, in particular the multi-robot surveillance scenario, also show the application of SC theory in an mobile multi-robot scenario of a considerable size. This is another important contribution of this work. However, the implementation of this scenario also showed some of the limitation of the approach when dealing with intelligent robot systems. These limitations are due to two main reasons: the difficulty between mapping the robot's sensor readings into the occurrence of uncontrollable events and the lack of ability to handle uncertainty. In the next section, where we discuss further work, we will provide a possible research route to mitigate these limitations. Furthermore, we also note that these limitations are inherent to the application of SC theory in this field, and the main goal of this thesis was to provide a general framework for the use of LTL specifications in SC of DES modelled as PNs.

7.2 Further work

Our introduced methods constitute a novel approach to the SC of PNs. Moreover, the theory for SC of PNs is still not completely studied. Thus, there is a significant amount of open problems and improvements that can be subject of future work.

- Dealing with dead transitions in the BA/PN system model composition. As referred is Section 4.2.2, since the BA/PN system model composition is based on analysing the PN structure, without building the reachability graph, some transitions are created that are never active in the reachable markings of the PN resulting from the composition. These dead transitions do not affect the behaviour of the system, but their deletion is desirable because they increase the size of the model and, as seen in Section 6.2, there can be a large amount of dead transitions. This larger size of the model has an impact on analysis and also on execution, because there are transitions that can never be active but nonetheless are checked at each step of the feedback loop. The simple approach we presented already deletes a large percentage of dead transitions, but more effective approaches should be used. In a complementary approach to solve this problem, it might be possible that one can detect at least a subset of the dead transitions during the composition, and this will be investigated.

- We presented the use of a knowledge base K for PN system models with symbolic state description that provides extra information which allows for smaller supervisors. We briefly mentioned that PN analysis techniques can be used to create the knowledge base, but the version we presented still relies on the designer writing information that he knows is true in all reachable markings of the PN. Creating knowledge bases automatically from the PN structure using invariant analysis, and extending the notion of knowledge base for PN system models with algebraic state description will be subject of future work.

- Modular supervision for systems generating ω-languages has not been studied in the literature. In particular, it is relevant to define methodologies to check deadlock-freeness under modular supervision without composing all the supervisors (thus effectively ruling out the argument for using modular SC). In our case, since the specifications are initially given in LTL, it would be useful to find a way of checking the different formulas used for each specifications and guarantee that their conjunction does not originate deadlock situations. We believe it is possible to do this using the BA obtained from the formulas. Furthermore, generalizing deadlock prevention algorithms so that they can deal with the concurrent execution of a set of PNs might be both interesting from a theoretical point-of-view, and to provide a more efficient solution for the problem.

- Studying the class of deterministic PN languages generated by our approach. In Section 5.2.2, when discussing how to apply the standard algorithm for checking admissibility for our methodology, we considered G as a deterministic PN representation of the system behaviour, and $G \parallel G_\varphi$ as the deterministic PN language specification. Thus, we argued that our methodology can be seen as a way to create deterministic PN language specifications, in the style of the Ramadge-Wonham framework. This raises a set of interesting questions which are worth being studied, e.g., what is the class of deterministic PN languages that can be specified using syntactically safe LTL and PN system models with algebraic state description?

- Given that the result of the BA/system model composition always has a substructure that represents the determinization of the BA, it is expectable that taking the properties of that substructure into account (e.g., at any marking one and only one place of the Büchi determinization will be marked) one can improve the computational efficiency of the algorithms for checking admissibility and deadlock-freeness. It might also be possible that the substructure can help in trimming the composition so that in becomes deadlock-free and admissible.

- An important result of Section 5.2.2 is that we are able to represent all the markings which are deadlocked or uncontrollable in a finite structure. The goal now will be to take advantage of this structure to create approaches that ensure admissibility and deadlock-freeness of the supervisor. The marking-based control-laws are the most direct approach to do this, but there might be others.

- Given the difficulty on handling arbitrary PNs, we can consider special classes of PN (e.g., workflow nets, free-choice nets, etc.) and use their properties to improve the algorithms for admissibility and deadlock-freeness. Furthermore it would be interesting to analyse which classes of PNs are closed for the BA/system model composition.

- One can also define an equivalent approach for other classes of LTL. The safety fragment of LTL is quite well-behaved and the more attentive reader will notice that all the big challenges in this methodology stem from the use of PNs. However, we can also think of increasing the expressibility of the LTL specifications. One immediate class of LTL formulas that can be worked on is the class of co-safe languages, i.e., languages which are the complement of safe languages. These languages are also accepted by deterministic BA, and they have a single accepting state which, once the reached for the first time, cannot be exited anymore. Furthermore, after the accepting state is reached, all the labels of the BA alphabet become enabled. Thus, we can use co-safe LTL formulas to specify reachability behaviours. However, guaranteeing that the accepting state of the BA is eventually reached in finite time for all the executions of the system will require extra operations on the composition.

- The decentralized approach presents a large set of opportunities to extend our proposals:

  - Define a way to incorporate shared description symbols in the specification;
  - Allow the designer to write global specifications, which are then satisfied locally and in a decentralized way;
  - Abstract the id's of the robots when that is possible;
  - Extend the approach to system models with algebraic state description;
  - Framing this decentralization methodology within the existing theory of distributed SC can be studied.

- A final and particularly interesting direction of further work in the context of intelligent robot systems is related to adding uncertainty to the models. The methodologies we presented are purely qualitative, which can be seen as a limitation in the context of robotics, where there is an inherent uncertainty about the state of the world that stems from sensor's inaccuracy. Thus, adding uncertainty to the models is both interesting in theoretical terms and has a very good motivation in terms of the method's applicability. One particularly interesting idea for future work is using generalized stochastic Petri nets (GSPN) [Viswanadham and Narahari, 1992] to model the systems and take advantage of recent work that connects GSPN models with interactive Markov chains (IMC) [Katoen, 2012]. The IMCs can then be analysed, using recent developments in the analysis of real-time stochastic games to obtain several quantitative objectives such as expected time to reach a marking, time spent in a given marking in the long run, or time-bounded reachability objectives. Furthermore, these analysis tools yield marking based policies that minimize or maximize each of these objectives, or a mix between them. Given this,

it is our claim that these approaches can be further investigated to see how one can find policies that maximize the probability of satisfying objectives given in LTL, i.e., we can adapt this approach to also do planning. Also, one can use modelling and identification approaches using GSPNs and specially tailored for robot systems, e.g., [Costelha and Lima, 2012]. Thus, when working towards making our methodology more applicable in robot systems, this direction is very relevant.

The fact that so many new challenges arose from this work also indicates that this line of research has the potential for improvement, being a relevant topic to be further tackled.

Appendices

APPENDIX A

PN Models for the Surveillance Scenario

We present the PN models used for the implementation described in Section 6.3. In the end od the appendix, we also list the set of uncontrollable events for robot i. All the places in the models have bound 1, except for places $rooms\_passed_i$ and $\overline{rooms\_passed_i}$, which have bound 10.

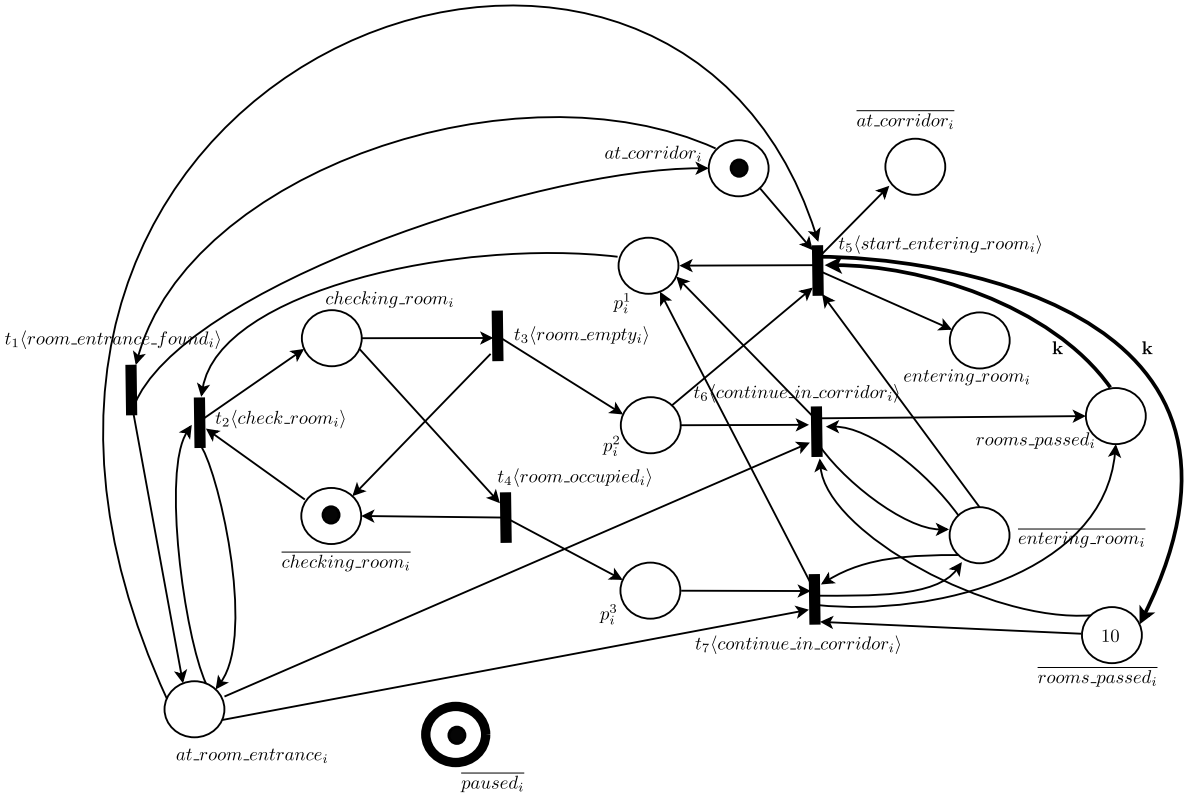


Figure A.1: PN model for the surveillance behaviour while in the corridor. Note that t_5 is in fact representing 10 transitions, one for each $k = 1, ..., 10$ (to represent this we also but the corresponding arcs with weight k in bold, to depict the representation of 10 arcs). Each transition t_5^k is enabled if and only if place $rooms\_passed_i$ has exactly k tokens. This structure can be easily implemented in a PN. The place $\overline{paused_i}$ has reflexive arcs for all the transitions, i.e., they can only be active when there is a token in $\overline{paused_i}$. To improve readability, we depict this by putting the place in bold.

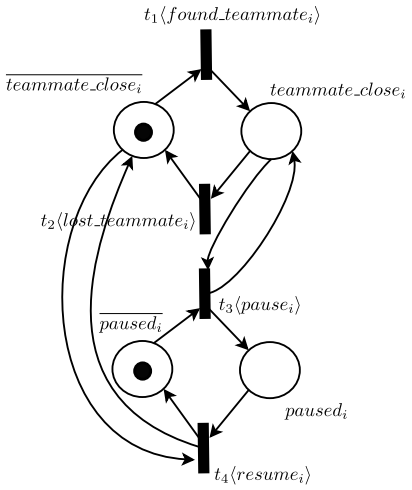


Figure A.2: PN model for the teammate avoidance behaviour.

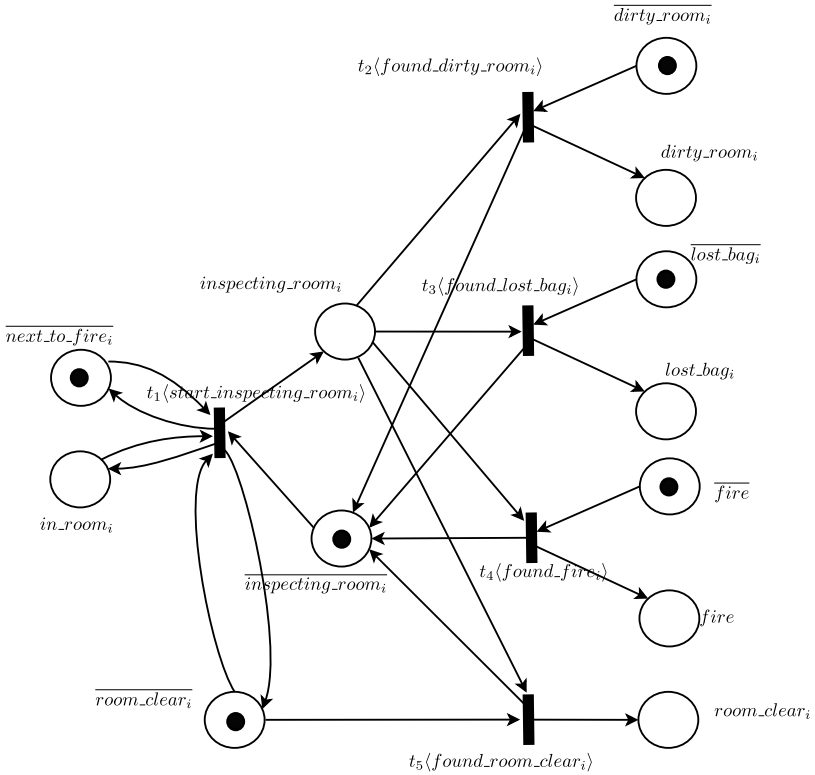


Figure A.3: PN model for the room inspection behaviour.

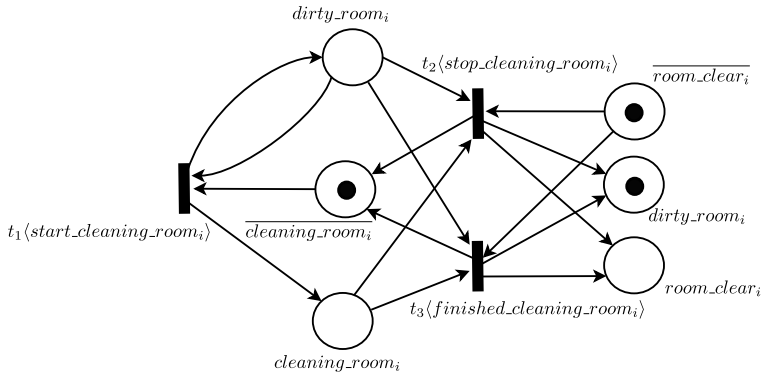


Figure A.4: PN model for the room cleaning behaviour.

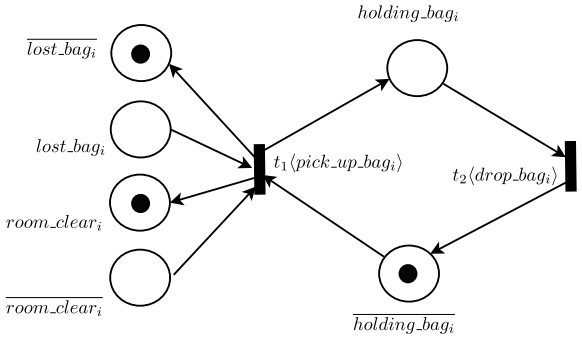


Figure A.5: PN model for the pick up and drop bag behaviour.

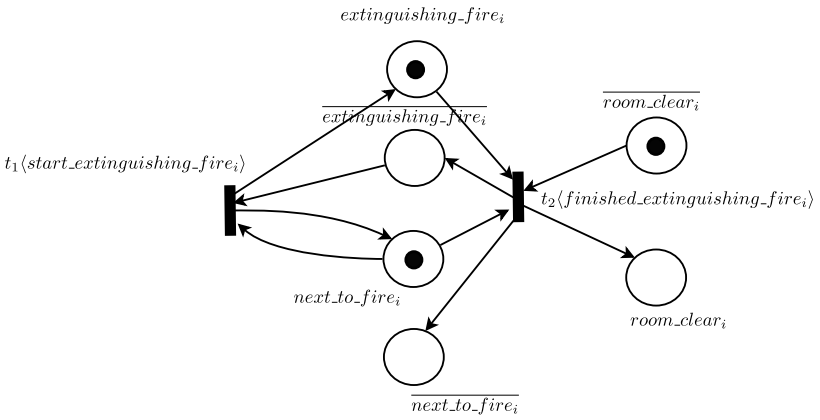


Figure A.6: PN model for the fire extinguishing behaviour.

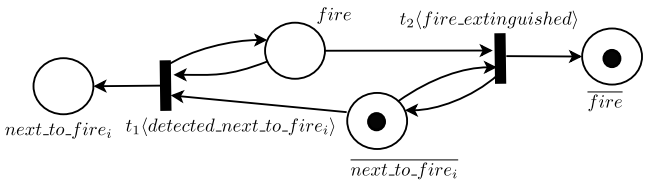


Figure A.7: PN model for the detection of being next to a fire. Note that the places representing $fire$ and $\overline{fire}$ are not indexed with *i*, i.e., they are present in the models for all robots and will be merged when doing the parallel composition to create the team model.

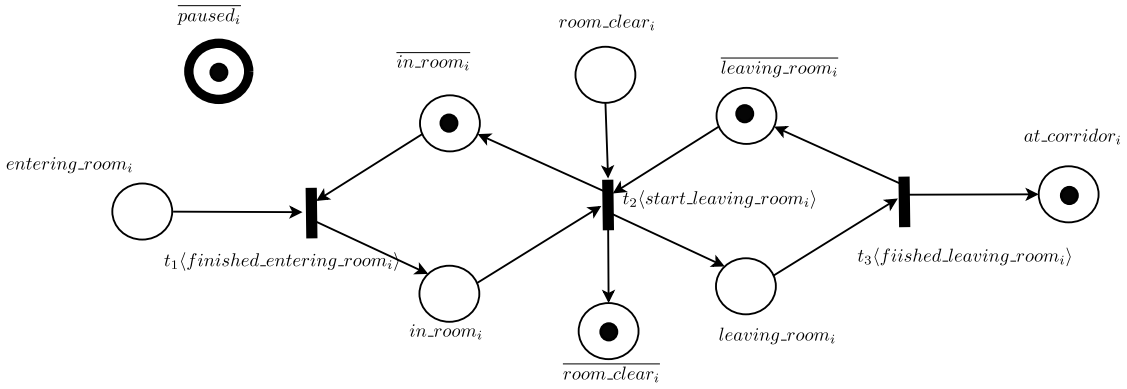


Figure A.8: PN model for moving behaviour when inside a room.

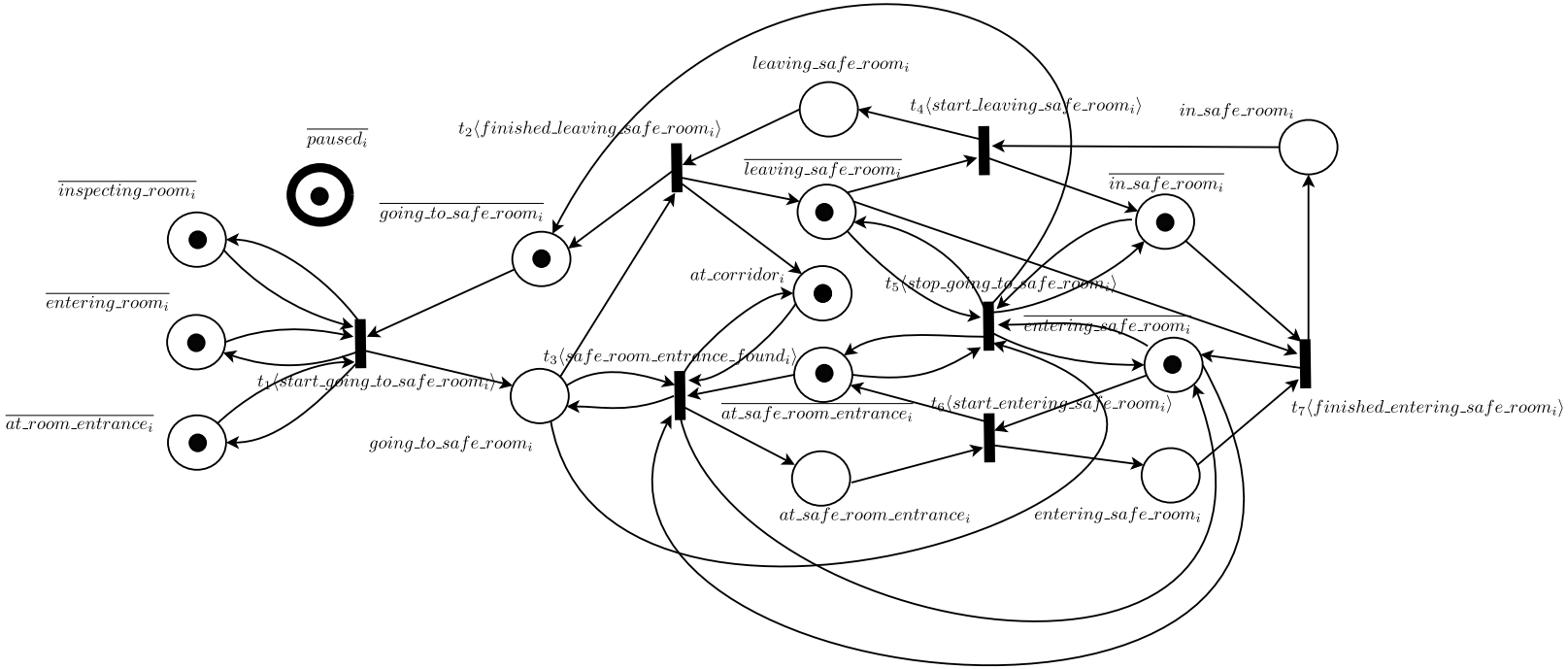


Figure A.9: PN model for the moving towards safe room behaviour.

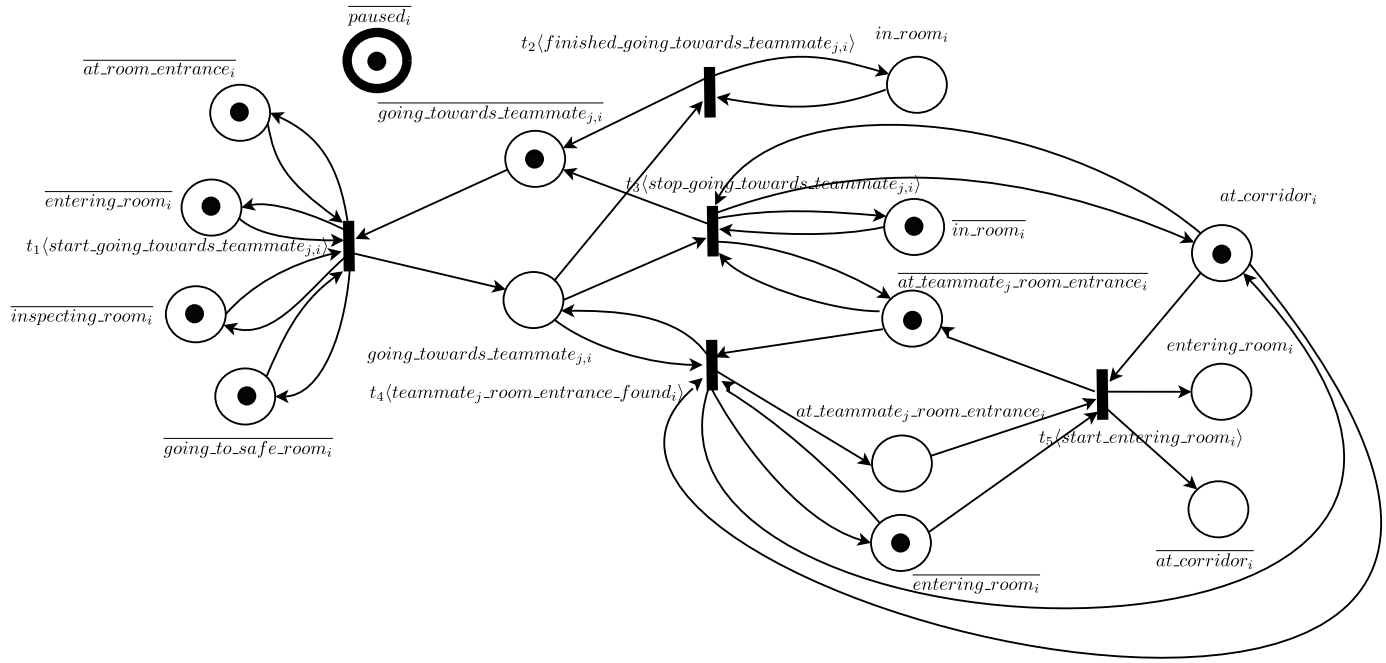


Figure A.10: PN model for the moving towards teammate j. Note that for each robot i, we have $n-1$ of these models, where n is the total number of robots on the team.

Uncontrollable events for robot *i*:

$room\_entrance\_found_i$	$room\_empty_i$	$room\_occupied_i$
$finished\_cleaning\_room_i$	$detected\_next\_to\_fire_i$	$fire\_extinguished$
$found\_dirty\_room_i$	$found\_lost\_bag_i$	$found\_fire_i$
$found\_room\_clear_i$	$finished\_extinguishing\_fire_i$	$finished\_entering\_room_i$
$finished\_leaving\_room_i$	$found\_teammate_i$	$lost\_teammate_i$
$finished\_leaving\_safe\_room_i$	$safe\_room\_entrance\_found_i$	$finished\_entering\_safe\_room_i$
$finished\_going\_towards\_teammate_{j,i}$	$teammate_j\_room\_entrance\_found_i$	

Bibliography

[Abdulla et al., 2004] Abdulla, P. A., Iyer, S., and Nylén, A. (2004). SAT-solving the coverability problem for Petri nets. *Formal Methods in System Design*, 24(1):25–43.

[Alpern and Schneider, 1987] Alpern, B. and Schneider, F. (1987). Recognizing safety and liveness. *Distributed computing*, 2(3):117–126.

[Asín et al., 2011] Asín, R., Nieuwenhuis, R., Oliveras, A., and Rodríguez-Carbonell, E. (2011). Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221.

[Aydin Gol et al., 2012] Aydin Gol, E., Lazar, M., and Belta, C. (2012). Language-guided controller synthesis for discrete-time linear systems. In *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, pages 95–104, Beijing, China. ACM.

[Barber and Salichs, 2001] Barber, R. and Salichs, M. (2001). A new human based architecture for intelligent autonomous robots. In *Proceedings of the The Fourth IFAC Symposium on Intelligent Autonomous Vehicles*, pages 85–90, Sapporo, Japan.

[Basile et al., 2006] Basile, F., Chiacchio, P., and Giua, A. (2006). Suboptimal supervisory control of petri nets in presence of uncontrollable transitions via monitor places. *Automatica*, 42(6):995–1004.

[Belta et al., 2007] Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., and Pappas, G. (2007). Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine,*, 14(1):61–70.

[Cassandras and Lafortune, 2006] Cassandras, C. G. and Lafortune, S. (2006). *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Chen et al., 2012a] Chen, C.-H., Kuo, C.-M., Chen, C.-Y., and Dai, J.-H. (2012a). The design and synthesis using hierarchical robotic discrete-event modeling. *Journal of Vibration and Control*.

[Chen et al., 2012b] Chen, Y., Tumová, J., and Belta, C. (2012b). LTL robot motion control based on automata learning of environmental dynamics. In *Procedings of ICRA '12: The 2012 IEEE International Conference on Robotics and Automation*, pages 5177–5182, St. Paul, MN, USA. IEEE.

[Chen et al., 2000] Chen, Y.-L., Lafortune, S., and Lin, F. (2000). Design of nonblocking modular supervisors using event priority functions. *IEEE Transactions on Automatic Control*, 45(3):432–452.

[Cheng et al., 1993] Cheng, A., Esparza, J., and Palsberg, J. (1993). Complexity results for 1-safe nets. In *Foundations of software technology and theoretical computer science*, pages 326–337. Springer.

[Cimatti et al., 2002] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). Nusmv 2: An opensource tool for symbolic model checking. In *Proceedings of CAV '02: The 14th International Conference on Computer Aided Verification*, pages 241–268, Copenhagen, Denmark. Springer.

[Cimatti et al., 2003] Cimatti, A., Pistore, M., Roveri, M., and Traverso, P. (2003). Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1–2):35 – 84.

[Cizelj and Belta, 2012] Cizelj, I. and Belta, C. (2012). Control of noisy differential-drive vehicles from time-bounded temporal logic specifications. *arXiv preprint arXiv:1209.1139*.

[Costelha and Lima, 2007] Costelha, H. and Lima, P. (2007). Modeling, analysis and execution of robotic tasks using Petri nets. In *Proceedings of IROS 2007 - IEEE International Conference on Intelligent Robots and Systems*, pages 1449–1454, San Diego, CA, USA.

[Costelha and Lima, 2008] Costelha, H. and Lima, P. (2008). Modelling, analysis and execution of multi-robot tasks using Petri nets. In *Proceedings of AAMAS '08: The 7th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1449–1454, Estoril, Portugal.

[Costelha and Lima, 2012] Costelha, H. and Lima, P. U. (2012). Robot task plan representation by petri nets: modelling, identification, analysis and execution. *Autonomous Robots*, pages 1–24.

[David and Alla, 2010] David, R. and Alla, H. (2010). *Discrete, continuous, and hybrid Petri nets*. Springer.

[De Giacomo and Vardi, 2000] De Giacomo, G. and Vardi, M. (2000). Automata-theoretic approach to planning for temporally extended goals. *Recent Advances in AI Planning*, pages 226–238.

[Diaz, 2010] Diaz, M. (2010). *Petri Nets: Fundamental models, verification and applications*. Wiley-ISTE.

[Emerson, 1990] Emerson, E. A. (1990). *Temporal and modal logic. In Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 995–1072. MIT Press, Cambridge, MA, USA.

[Esparza, 1994] Esparza, J. (1994). Model checking using net unfoldings. *Science of Computer Programming*, 23(2–3):151 – 195.

[Esparza and Heljanko, 2001] Esparza, J. and Heljanko, K. (2001). Implementing ltl model checking with net unfoldings. *Lecture Notes on Computer Science – Model Checking Software*, 2057:37–56.

[Fainekos et al., 2005] Fainekos, G., Kress-Gazit, H., and Pappas, G. (2005). Temporal logic motion planning for mobile robots. In *Proceedings of ICRA '05: The 2005 IEEE International Conference on Robotics and Automation*, pages 2020–2025, Barcelona, Spain. IEEE.

[Fainekos et al., 2006] Fainekos, G., Loizou, S., and Pappas, G. (2006). Translating temporal logic to controller specifications. In *Proceedings of CDC '06: The 45th IEEE Conference on Decision and Control*, pages 899–904, San Diego, CA, USA. IEEE.

[Gastin and Oddoux, 2001] Gastin, P. and Oddoux, D. (2001). Fast LTL to Büchi automata translation. In *Proceedings of CAV '01: The 13th International Conference on Computer Aided Verification*, pages 53–65, London, UK.

[Girault and Valk, 2002] Girault, C. and Valk, R. (2002). *Petri nets for systems engineering: a guide to modeling, verification, and applications*. Springer.

[Giua, 1992] Giua, A. (1992). *Petri nets as discrete event models for supervisory control*. PhD thesis, Rensselaer Polytechnic Institute.

[Giua, 2013] Giua, A. (2013). Supervisory control of Petri nets with language specifications. *Control of Discrete-Event Systems – Lecture Notes in Control and Information Sciences*, 433:235–255.

[Giua and DiCesare, 1991] Giua, A. and DiCesare, F. (1991). Supervisory design using Petri nets. In *Proceedings of the 30th IEEE Conference on Decision and Control*, pages 92–97, Brighton , UK.

[Giua and DiCesare, 1994a] Giua, A. and DiCesare, F. (1994a). Blocking and controllability of Petri nets in supervisory control. *IEEE Transactions on Automatic Control*, 39(4):818–823.

[Giua and DiCesare, 1994b] Giua, A. and DiCesare, F. (1994b). Petri net structural analysis for supervisory control. *IEEE Transactions on Robotics and Automation*, 10(2):185–195.

[Giua and DiCesare, 1995] Giua, A. and DiCesare, F. (1995). Decidability and closure properties of weak Petri net languages in supervisory control. *IEEE Transactions on Automatic Control*, 40(5):906–910.

[Giua et al., 1992] Giua, A., DiCesare, F., and Silva, M. (1992). Generalized mutual exclusion constraints on nets with uncontrollable transitions. In *Procedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 974–979 vol. 2, Chicago, IL, USA.

[Glynn, 1989] Glynn, P. W. (1989). A GSMP formalism for discrete event systems. *Proceedings of the IEEE*, 77(1):14–23.

[Gromyko et al., 2006] Gromyko, A., Pistore, M., and Traverso, P. (2006). A tool for controller synthesis via symbolic model checking. In *Proceedings of the 8th International Workshop on Discrete Event Systems*, pages 475–476, Ann Arbor, MI, USA. IEEE.

[Hill and Tilbury, 2006] Hill, R. C. and Tilbury, D. M. (2006). Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction. In *8th International Workshop on Discrete Event Systems*, pages 399–406. IEEE.

[Holloway et al., 1997] Holloway, L. E., Krogh, B. H., and Giua, A. (1997). A survey of Petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems*, 7(2):151–190.

[Hopcroft et al., 2006] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA.

[Ichikawa and Hiraishi, 1988] Ichikawa, A. and Hiraishi, K. (1988). Analysis and control of discrete event systems represented by Petri nets. *Discrete Event Systems: Models and Applications*, pages 115–134.

[Iordache and Antsaklis, 2003] Iordache, M. and Antsaklis, P. (2003). Design of T-liveness enforcing supervisors in Petri nets. *IEEE Transactions on Automatic Control*, 48(11):1962–1974.

[Iordache and Antsaklis, 2006a] Iordache, M. and Antsaklis, P. (2006a). *Supervisory Control of Concurrent Systems: A Petri Net Structural Approach*. Birkhäuser Boston.

[Iordache et al., 2002] Iordache, M., Moody, J., and Antsaklis, P. (2002). Synthesis of deadlock prevention supervisors using Petri nets. *IEEE Transactions on Robotics and Automation*, 18(1):59–68.

[Iordache and Antsaklis, 2002] Iordache, M. V. and Antsaklis, P. J. (2002). Synthesis of supervisors enforcing general linear vector constraints in Petri nets. In *Proceedings of the 2002 American Control Conference*, pages 154–159, Anchorage, Alaska, USA.

[Iordache and Antsaklis, 2006b] Iordache, M. V. and Antsaklis, P. J. (2006b). Supervision based on place invariants: A survey. *Discrete Event Dynamic Systems*, 16(4):451–492.

[Jantzen, 1987] Jantzen, M. (1987). Language theory of Petri nets. *Petri Nets: Central Models and Their Properties*, pages 397–412.

[Jiang and Kumar, 2006a] Jiang, S. and Kumar, R. (2006a). Diagnosis of repeated failures for discrete event systems with linear-time temporal-logic specifications. *IEEE Transactions on Automation Science and Engineering*, 3(1):47–59.

[Jiang and Kumar, 2006b] Jiang, S. and Kumar, R. (2006b). Supervisory control of discrete event systems with CTL* temporal logic specifications. *SIAM Journal on Control and Optimization*, 44(6):2079–2103.

[Johnson et al., 2012] Johnson, B., Havlak, F., Campbell, M., and Kress-Gazit, H. (2012). Execution and analysis of high-level tasks with dynamic obstacle anticipation. In *Procedings of ICRA '12: The 2012 IEEE International Conference on Robotics and Automation*, pages 330–337, St. Paul, MN, USA. IEEE.

[Kato and Billinghurst, 1999] Kato, H. and Billinghurst, M. (1999). Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proceedings of IWAR '99: The 2nd IEEE and ACM International Workshop on Augmented Reality*, pages 85–97, Washington, DC, USA.

[Katoen, 2012] Katoen, J.-P. (2012). GSPNs revisited: Simple semantics and new analysis algorithms. In *Proceedings of ACSD '12: The 12th International Conference on Application of Concurrency to System Design*, pages 6–11, Hamburg, Germany.

[Kloetzer and Belta, 2006] Kloetzer, M. and Belta, C. (2006). LTL planning for groups of robots. In *ICNSC '06: Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control*, pages 393 – 398, Ft. Lauderdale, FL, USA.

[Kloetzer and Belta, 2007] Kloetzer, M. and Belta, C. (2007). Control of multi-robot teams based on LTL specifications. In *Proceedings of IFAC '07: Conference on Management and Control of Production and Logistics*, pages 103–108, Sibiu, Romania.

[Kloetzer and Belta, 2008a] Kloetzer, M. and Belta, C. (2008a). Distributed implementations of global temporal logic motion specifications. In *ICRA '08: Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, pages 393 – 398, Pasadena, CA, USA.

[Kloetzer and Belta, 2008b] Kloetzer, M. and Belta, C. (2008b). A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297.

[Kosecka and Bogoni, 1994] Kosecka, J. and Bogoni, L. (1994). Application of discrete events systems for modeling and controlling robotic agents. In *Proceedings of ICRA '94: The IEEE International Conference on Robotics and Automation*, pages 2557–2562. IEEE.

[Kress-Gazit et al., 2008] Kress-Gazit, H., Conner, D. C., Choset, H., Rizzi, A. A., and Pappas, G. J. (2008). Courteous cars: Decentralized multiagent traffic coordination. *IEEE Robotics and Automation Magazine*, 15(1):30–38.

[Kress-Gazit et al., 2007a] Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J. (2007a). From structured english to robot motion. In *Proceedings of IROS '07: IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2717 – 2722, San Diego, California, USA.

[Kress-Gazit et al., 2007b] Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J. (2007b). Where's waldo? Sensor-based temporal logic motion planning. In *Proceedings of ICRA '07: The 2007 IEEE International Conference on Robotics and Automation*, pages 3116–3121, Rome, Italy.

[Kress-Gazit et al., 2009] Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J. (2009). Temporal logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381.

[Krogh, 1987] Krogh, B. (1987). Controlled Petri nets and maximally permissive feedback logic. In *Proceedings of 25th Annual Allerton Conference*, Urbana, Illinois.

[Kumar and Garg, 1995] Kumar, R. and Garg, V. (1995). *Modeling and control of logical discrete event systems*. Boston, MA: Kluwer Academic Publishers.

[Kumar and Holloway, 1996] Kumar, R. and Holloway, L. (1996). Supervisory control of deterministic Petri nets with regular specification languages. *IEEE Transactions on Automatic Control*, 41(2):245–249.

[Kupferman and Vardi, 2001] Kupferman, O. and Vardi, M. (2001). Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314.

[Lacerda and Lima, 2009] Lacerda, B. and Lima, P. (2009). LTL plan specification for robotic tasks modelled as finite state automata. In *Proceedings of Workshop ADAPT - Agent Design: Advancing from Practice to Theory, Worksshop at AAMAS '09: The 8th Int. Conf. on Autonomous Agents and Multiagent Systems*, Budapest, Hungary.

[Lacerda and Lima, 2008] Lacerda, B. and Lima, P. U. (2008). Linear-time temporal logic control of discrete event models of cooperative robots. *Journal of Physical Agents*, 2(1):53–61.

[Lacerda and Lima, 2011a] Lacerda, B. and Lima, P. U. (2011a). Designing Petri net supervisors for from LTL specifications. In *Proceedings of RSS VII – The 2011 Robotics: Science and Systems Conference*, Los Angeles, CA, USA.

[Lacerda and Lima, 2011b] Lacerda, B. and Lima, P. U. (2011b). Designing Petri net supervisors for multi-agent systems from LTL specifications (extended abstract). In *Proceedings of AAMAS '11: The 10th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1253–1254, Taipei, Taiwan.

[Lacerda and Lima, 2011c] Lacerda, B. and Lima, P. U. (2011c). LTL-based decentralized supervisory control of multi-robot tasks modelled as Petri nets. In *Proceedings of IROS '11: The IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3081–3086. IEEE.

[Lacerda et al., 2011] Lacerda, B., Lima, P. U., Gorostiza, J., and Salichs, M. A. (2011). Petri net based supervisory control of a social robot with LTL specifications. In *Proceedings of ROBOTICA '11: the 11th International Conference on Mobile Robots and Competitions*, pages 46–51, Lisbon, Portugal.

[Lahijanian et al., 2009] Lahijanian, M., Andersson, S., and Belta, C. (2009). A probabilistic approach for control of a stochastic system from LTL specifications. In *Proceedings of CDC '09: The 48th IEEE Conference on Decision and Control*, pages 2236–2241, Shanghai, China. IEEE.

[Lee et al., 2005] Lee, J.-S., Zhou, M.-C., and Hsu, P.-L. (2005). An application of Petri nets to supervisory control for human-computer interactive systems. *IEEE Transactions on Industrial Electronics*, 52(5):1220 – 1226.

[Li et al., 2012] Li, Z., Wu, N., and Zhou, M. (2012). Deadlock control of automated manufacturing systems based on Petri nets,Äî-a literature review. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(4):437–462.

[Li and Zhou, 2009] Li, Z. and Zhou, M. (2009). *Deadlock resolution in automated manufacturing systems: a novel Petri net approach*, volume 1430. Springer.

[Li et al., 2008] Li, Z. W., Zhou, M. C., and Wu, N. Q. (2008). A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):173–188.

[Loizou and Kyriakopoulos, 2004] Loizou, S. and Kyriakopoulos, K. (2004). Automatic synthesis of multi-agent motion tasks based on LTL specifications. In *Proceedings of CDC '04: The 43rd IEEE Conference on Decision and Control*, pages 153–158, Paradise Island, Bahamas. IEEE.

[McMillan, 1992] McMillan, K. (1992). Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proceddings of CAV '92: The 4th International Workshop on Computer Aided Verification*, pages 164–177, Montreal, Canada. Springer.

[Melzer and Römer, 1997] Melzer, S. and Römer, S. (1997). Deadlock checking using net unfoldings. In *Proceedings of CAV '97: The 9th International Workshop on Computer Aided Verification*, pages 352–363, Haifa, Israel. Springer.

[Milutinovic and Lima, 2002] Milutinovic, D. and Lima, P. (2002). Petri net models of robotic tasks. In *ICRA '02: Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 4059–4064, Washington, D.C.

[Mondada et al., 2009] Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, pages 59–65, Castelo Branco, Portugal.

[Moody and Antsaklis, 1998] Moody, J. and Antsaklis, P. (1998). *Supervisory control of discrete event systems using Petri nets*. Springer.

[Moody and Antsaklis, 1999] Moody, J. O. and Antsaklis, P. J. (1999). Petri net supervisors for DES with uncontrollable and unobservable transitions. *IEEE Transactions on Automatic Control*, 45:462–476.

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

[Pistore and Traverso, 2001] Pistore, M. and Traverso, P. (2001). Planning as model checking extended goals in non-deterministic domains. In *Proceedings of IJCAI '01: The 17th Int. Joint Conf. On Artificial Intelligence*, pages 479–484, Seattle, WA, USA.

[Ramadge and Wonham, 1987] Ramadge, P. and Wonham, W. (1987). Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1):206–230.

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

[Raman and Kress-Gazit, 2012a] Raman, V. and Kress-Gazit, H. (2012a). Automated feedback for unachievable high-level robot behaviors. In *Procedings of ICRA '12: The 2012 IEEE International Conference on Robotics and Automation*, pages 5156–5162, St. Paul, MN, USA. IEEE.

[Raman and Kress-Gazit, 2012b] Raman, V. and Kress-Gazit, H. (2012b). Explaining impossible high-level robot behaviors. *IEEE Transactions on Robotics*. to be published.

[Reisig and Rozenberg, 1998a] Reisig, W. and Rozenberg, G. (1998a). *Lectures on Petri Nets Part I: Basic Models: Advances in Petri Nets*. Springer.

[Reisig and Rozenberg, 1998b] Reisig, W. and Rozenberg, G. (1998b). *Lectures on Petri Nets Part II: Applications*. Springer.

[Ricker et al., 1996] Ricker, S., Sarkar, N., and Rudiet, K. (1996). A discrete-event systems approach to modeling dextrous manipulation. *Robotica*, 14(05):515–525.

[Ru et al., 2012] Ru, Y., Cabasino, M. P., Giua, A., and Hadjicostis, C. N. (2012). Supervisor synthesis for discrete event systems under partial observation and arbitrary forbidden state specifications. *Discrete Event Dynamic Systems*, pages 1–33.

[Russell et al., 1995] Russell, S., Norvig, P., Canny, J., Malik, J., and Edwards, D. (1995). *Artificial intelligence: a modern approach*, volume 2. Prentice hall Englewood Cliffs, NJ.

[Salichs et al., 2006] Salichs, M., Barber, R., Khamis, A., Malfaz, M., Gorostiza, J., Pacheco, R., Rivas, R., Corrales, A., Delgado, E., and Garcia, D. (2006). Maggie: A robotic platform for human-robot social interaction. In *RAM '06: 2006 IEEE Conference on Robotics, Automation and Mechatronics*, pages 1–7, Bangkok, Thailand.

[Sarid et al., 2012] Sarid, S., Xu, B., and Kress-Gazit, H. (2012). Guaranteeing high-level behaviors while exploring partially known maps. *Proc. of Robotics: Science and Systems, Sydney, Australia*.

[Schrijver, 1998] Schrijver, A. (1998). *Theory of linear and integer programming*. Wiley.

[Schröter and Khomenko, 2004] Schröter, C. and Khomenko, V. (2004). Parallel LTL-X model checking of high-level Petri nets based on unfoldings. In *Proceedings of CAV '04: The 16th International Conference on Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 374–377. Springer Berlin / Heidelberg.

[Seatzu et al., 2012] Seatzu, C., Silva, M., and van Schuppen (Editors), J. H. (2012). *Control of Discrete-Event Systems Lecture Notes in Control and Information Sciences Vol. 433*. Springer.

[Sheng and Yang, 2005] Sheng, W. and Yang, Q. (2005). Peer-to-peer multi-robot coordination algorithms: Petri net based analysis and design. In *Proceedings of 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 1407–1412, Monterey, CA, USA.

[Silva et al., 1998] Silva, M., Terue, E., and Colom, J. (1998). Linear algebraic and linear programming techniques for the analysis of place/transition net systems. *Lecture Notes in Computer Science – Lectures on Petri Nets I: Basic Models*, pages 309–373.

[Sistla, 1994] Sistla, A. (1994). Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511.

[Smith et al., 2011] Smith, S., Tůmová, J., Belta, C., and Rus, D. (2011). Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research*, 30(14):1695–1708.

[Sreenivas, 1997] Sreenivas, R. S. (1997). On the existence of supervisory policies that enforce liveness in discrete-event dynamic systems modeled by controlled Petri nets. *IEEE Transactions on Automatic Control*, 42(7):928–945.

[Sreenivas, 2012] Sreenivas, R. S. (2012). On the existence of supervisory policies that enforce liveness in partially controlled free-choice Petri nets. *IEEE Transactions on Automatic Control*, 57(2):435–449.

[Tabuada and Pappas, 2003] Tabuada, P. and Pappas, G. J. (2003). Model checking LTL over controllable linear systems is decidable. In *Proceedings of HSCC'03: The 6th International Conference on Hybrid systems: Computation and Control*, pages 498–513, Prague, Czech Republic.

[Tabuada and Pappas, 2006] Tabuada, P. and Pappas, G. J. (2006). Linear time logic control of discrete-time linear systems. *IEEE Transactions on Automatic Control*, 51:1862–1877.

[Thistle, 1996] Thistle, J. (1996). Supervisory control of discrete event systems. *Mathematical and Computer Modelling*, 23(11):25–53.

[Thistle and Wonham, 1994] Thistle, J. G. and Wonham, W. M. (1994). Supervision of infinite behavior of discrete-event systems. *SIAM Journal on Control and Optimization*, 32(4):1098–1113.

[Ulusoy et al., 2012a] Ulusoy, A., Smith, S., and Belta, C. (2012a). Optimal multi-robot path planning with LTL constraints: Guaranteeing correctness through synchronization. *arXiv preprint arXiv:1207.2415*.

[Ulusoy et al., 2012b] Ulusoy, A., Smith, S., Ding, X., and Belta, C. (2012b). Robust multi-robot optimal path planning with temporal logic constraints. In *Procedings of ICRA '12: The 2012 IEEE International Conference on Robotics and Automation*, pages 4693–4698, St. Paul, MN, USA. IEEE.

[Ulusoy et al., 2012c] Ulusoy, A., Wongpiromsarn, T., and Belta, C. (2012c). Incremental control synthesis in probabilistic environments with temporal logic constraints. *arXiv preprint arXiv:1209.0136*.

[van der Aalst, 1998] van der Aalst, W. (1998). The application of Petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66.

[van der Aalst and Stahl, 2011] van der Aalst, W. and Stahl, C. (2011). *Modeling Business Processes: A Petri Net-Oriented Approach*. MIT Press.

[Vardi, 1996] Vardi, M. (1996). An automata-theoretic approach to linear temporal logic. *Logics for concurrency*, pages 238–266.

[Viswanadham and Narahari, 1992] Viswanadham, N. and Narahari, Y. (1992). *Performance modeling of automated manufacturing systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Wang et al., 1991] Wang, F., Kyriakopoulos, K., Tsolkas, A., and Saridis, G. (1991). A Petri net coordination model for an intelligent mobile robot. *IEEE Transactions on Systems, Man and Cybernetics*, 21(4):777–789.

[Weidlich and van der Werf, 2012] Weidlich, M. and van der Werf, J. (2012). On profiles and footprints–relational semantics for Petri nets. In *Proceedings of Petri nets 2012: The 33rd International Conference on Application and Theory of Petri Nets and Concurrency*, pages 148–167, Hamburg, Germany.

[Wolff et al., 2012] Wolff, E., Topcu, U., and Murray, R. (2012). Optimal control with weighted average costs and temporal logic specifications. In *Proceedings of RSS VIII – The 2012 Robotics: Science and Systems Conference*, Sydney, NSW, Australia.

[Wolper, 1983] Wolper, P. (1983). Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99.

[Wolper, 2001] Wolper, P. (2001). Constructing automata from temporal logic formulas: A tutorial. In *Lectures on Formal Methods and Performance Analysis, vol. 2090 of Lecture Notes in Computer Science*, pages 261–277. Springer.

[Wonham and Ramadge, 1988] Wonham, W. M. and Ramadge, P. J. (1988). Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems (MCSS)*, 1(1):13–30.

[Wu et al., 2002] Wu, W., Su, H., and Chu, J. (2002). Supervisory control of discrete event systems using enabling arc Petri nets. In *Proceedings ICRA '02: The 2002 IEEE International Conference on Robotics and Automation*, pages 1913–1918, Washington, DC, USA.

[Ziparo and Iocchi, 2006] Ziparo, V. and Iocchi, L. (2006). Petri net plans. In *Proceedings of MOCA '06 - Fourth International Workshop on Modelling of Objects, Components and Agents*, pages 267–290, Turku, Finland.

[Ziparo et al., 2011] Ziparo, V. A., Iocchi, L., Lima, P. U., Nardi, D., and Palamara, P. F. (2011). Petri net plans: A framework for collaboration and coordination in multi-robot systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 23(3):344–383.

[Ziparo et al., 2008] Ziparo, V. A., Iocchi, L., Nardi, D., Palamara, P. F., and Costelha, H. (2008). Petri net plans: a formal model for representation and execution of multi-robot plans. In *Proceedings of AAMAS '08: The 7th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 79–86, Estoril, Portugal.