Petri Net Based Multi-Robot Task Coordination from Temporal Logic Specifications

Bruno Lacerda[a,*], Pedro U. Lima[b]

[a] *Oxford Robotics Institute, University of Oxford, UK*
[b] *Institute for Systems and Robotics, Instituto Superior Técnico, Portugal*

ARTICLE INFO

Keywords:
Multi-robot coordination
Linear temporal logic
Supervisory control
Petri nets

ABSTRACT

We propose a methodology for enforcing a set of coordination rules onto a multi-robot system, based on the use of Petri nets to model the team of robots, safe linear temporal logic to specify a set of coordination rules to be enforced, and supervisory control theory to synthesise a supervisor that enforces the coordination rules. We introduce a composition algorithm that allows us to build a Petri net that represents the largest restriction of the team behaviour that still satisfies the specification. Such a Petri net can be interpreted as a candidate for a supervisor, for which one needs to verify admissibility. We present a general verification procedure for this problem. We also present a syntactic restriction to safe linear temporal logic that guarantees admissibility of the composition a priori. We finish by providing an illustrative example, where we show how the use of temporal logic allows the designer to write the specifications intuitively, and the use of Petri nets allows us to tackle the large state spaces and high concurrency associated with multi-robot systems.

1. Introduction

A wide range of applications, such as logistics, environmental monitoring, and smart transportation systems require fleets of autonomous mobile robots. For such systems, formal guarantees concerning aspects such as safety, predictability, performance, robustness and reliability are essential to support the acceptability of the systems. However, in the field of robotics innovation is often associated with non-formal approaches. Novel ideas and concepts are typically introduced for particular applications, through well-engineered systems, but lack the ability to provide formal guarantees. A particularly important class of guarantee is *safety*, which allow for specifying that something "bad" never happens. Safety properties include requirements that must be satisfied in all states (e.g., robots should avoid collisions) or can dynamically change depending on observations (e.g., robots should change from a surveillance to a evacuation behaviour if a fire is detected).

In this work, we introduce a framework, based on a combination of concepts from discrete event systems and formal verification, that allows a designer to enforce a set of safety requirements on a multi-robot team. The framework proposes the use of Petri nets (PNs) to model the uncontrolled behaviour of the robot team, and safe linear temporal logic (LTL) for the specification of the coordination requirements. Then, we build upon concepts from supervisory control (SC) to synthesise a PN representation of a supervisor that restricts the behaviours of the team such that the coordination requirements are met. With this approach, we exploit the ability of PNs to compactly model systems with high concurrency and the powerful and intuitive ability to specify behaviours naturally provided by LTL.

Figure 1 depicts a diagram illustrating the methodology. To summarise, we start with a PN model of the team and a syntactically safe LTL formula written over the set of events of the system plus linear constraints on the markings of the PN. We translate the safe LTL formula into a deterministic finite automaton (DFA) that represents it. The DFA is then appropriately composed with the PN. This composition provides a structure that represents the smallest restriction – in the sense that any restriction which is more permissive in the 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, one needs to guarantee that the composition is admissible. Intuitively, admissibility means that the supervisor will not attempt to disable events it cannot control. We provide two approaches to ensure admissibility. The first, more general approach is based on the verification procedure described in Subsection 5.1. With this approach, if the composition is found to be admissible, then it can be used to supervise the PN. However, if it is not admissible, and since PNs are not closed under the supremal controllable sublanguage operator [13], the designer has to re-write the specification and re-check the new composition, which in practical terms is not satisfactory. Given this fact, along with the significant complexity of checking for admissibility, in Subsection 5.2 we present a syntactic restriction that ensures composition admissibility by design. Thus, if the specification is written according to this restriction, the composition is guaranteed to be an appropriate supervisor. As we will see, in practical terms using the restriction provided in Subsection 5.2 is a better approach, as it allows for the supervision of larger systems, and does not require formula re-writing. However, we also present the general algorithm for admissibility check in Section 5.1 due to its theoretical interest.

Figure 1 also includes the paper organisation, referring each of the required operations to the section where it is de-

bruno@robots.ox.ac.uk (B. Lacerda); pedro.lima@tecnico.ulisboa.pt (P.U. Lima)
ORCID(s): 0000-0003-0862-331X (B. Lacerda); 0000-0002-8962-8050 (P.U. Lima)

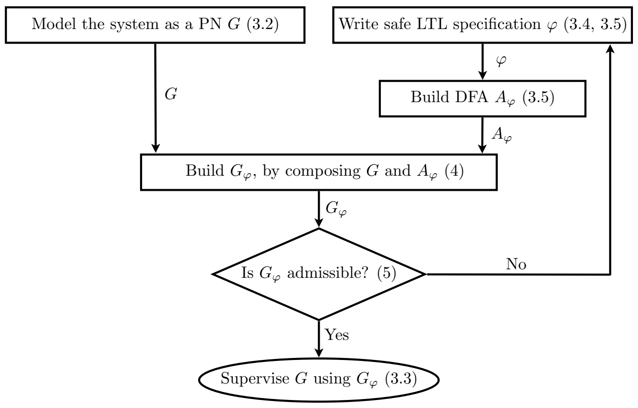
Model the system as a PN G (3.2)
Write safe LTL specification φ (3.4, 3.5)
φ
Build DFA A_φ (3.5)
G
A_φ
Build G_φ, by composing G and A_φ (4)
G_φ
Is G_φ admissible? (5)
No
Yes
Supervise G using G_φ (3.3)

Figure 1: The proposed methodology, along with pointers to the sections where each operation is described.

scribed. Our contributions are mainly on the steps described in Sections 4 and 5. Broadly speaking, these are:

1. The definition of a semantics to evaluate state/event 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;
2. A novel composition algorithm between a PN and a DFA representing the safe state/event LTL formula;
3. A general procedure to verify supervisor admissibility for the result of our composition algorithm, thus proving decidability of the problem;
4. A syntactic restriction to the specification language that guarantees admissibility by design, and can be used in practical terms.

In Section 6, we present an illustrative example that shows how our approach allows for the specification of intricate behaviours and the control of systems with large state spaces. While the procedure we present can be computationally expensive, all the heavy 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. This is crucial, given that the supervisor is used in a feedback-loop with the system, hence it needs to quickly react to changes in the system state.

2. Related Work

PNs have been used as a modelling formalism in several sub-fields of robotics. For example, [53] use PNs to model robot plans which support robust execution strategies and [6] use generalised stochastic PNs (GSPNs) to model and analyse the closed-loop execution of robot behaviour in its environment. However, in these works behaviours are not synthesised from the PN, as we do here. PN models have also been proposed for human-robot collaboration [5].

Given their clear semantics and ability to compactly model systems with high levels of concurrency, the application of PNs to model the behaviour of multi-robot systems has attracted particular interest. The work in [26] maps a homogeneous robot team to tokens in a PN to provide a compact representation of interchangeable robots in order to generate multi-robot paths that reach certain target locations in the environment. This is extended in [38] to allow for synthesis of multi-robot plans that satisfy a boolean team specification. Finally, [39] uses GSPNs to coordinate a team of robots under navigation duration uncertainty such that a reward representing system performance is maximised whilst maintaining a global constraint on team behaviour. These works exploit the mapping of robots to tokens to maintain a compact representation of the environment. Whilst this is not a requirement of the work we present here, our application example uses the same idea to keep the model compact.

In recent years, the use of formal methods for robotics, in particular temporal logics for robot behaviour specification and correct-by-construction synthesis have been advocated in several works and tackling a range of different problems, such as motion planning [27, 9, 8], partial satisfiability [35, 30] or multi-objective planning [34, 36]. Furthermore, many of these methods have been used as the basis for extensions that deal with multi-robot systems, both with local specifications for each robot [19, 42] or global, team-level, specification for which parts of the specification must be assigned to each robot [51, 47, 10]. All these works are based on a model that explicitly enumerates the possible states of the system. In our work, by using structural analysis of PNs, we strive to mitigate the scalability issues associated with state space enumeration. Furthermore, we show how our approach can be applied to task assignment for a team of robots. This problem has been addressed by several authors, and tackled using different techniques such as auctioning [7, 43], hybrid systems [17, 52], and fuzzy logic [50]. These solutions, however, are specific to the problem of task assignment, whilst the approach we formalise here is more general and can be applied to different problems, with task assignment being an instance of a problem our approach is particularly well suited for.

Supervisory control (SC) of discrete event systems (DES) modelled as DFA models and language specifications was introduced in the seminal work [45], being currently known in the community as the Ramadge-Wonham (R-W) framework. This work introduced the idea, also used here, of obtaining a supervisor by composing the system model with the specification model, and then analysing such composition in order to check for controllability. A fairly complete introduction to DES and SC is avaiable in [4]. The R-W framework was then extended to PN models of DES in [14, 12, 15]. This work shows some undesirable properties of PN languages that hinder their usability as models on the R-W framework. For example, general PN languages are not closed under the *supremal controllable sublanguage operator* [13], which makes it impossible to define general methods to find PN representations of the *least restrictive* admissible supervisor (i.e., the supervisor that is admissible, satisfies the specification, and disables the minimal amount of transitions to do so). In spite of this limitation, one can still check if a given supervisor candidate is admissible. See [11] for a thorough compilation of these results. The work presented here extends the specification language from a solely event-based language to a state/event-based language. This introduces a new challenge in terms of how to compose the plant model with the supervisor model. Furthermore, we introduce safe LTL as a specification language. We discuss these points further in Subsection 4.4. Alternative to using language specifications is the so called *supervision based on place invariants* (SBPI) or *generalised mutual exclusion constraints* (GMEC) [16, 21, 22, 40]. 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 is based on a linear algebraic approach that deals with several relevant notions for SC, such as admissibility [3], observability [41], deadlock avoidance [23] and liveness [20]. We will further discuss the relation between our approach and GMEC in Subsection 4.4.

The use of temporal logic for SC has mostly focused on modelling both system and specification as a temporal logic formula [24, 25] or in state-based models such as DFA [18, 48, 31]. The work presented here builds on [32], where LTL-based approaches for SC was defined for PNs. Here, we extend the semantics of LTL in order to reason directly over markings. This not only enriches the specification language, but also reduces the size of the supervisors, as we will show in Section 6.

3. Preliminaries

3.1. Notation

Let A be a set. We define the set A^* as the set of all finite sequences that can be built from A (including the empty string ϵ), and A^ω as the set of all infinite sequences that can be built from A. For $s \in A^*$ and $\sigma \in A^\omega$, we define $s.\sigma \in A^\omega$ as the concatenation of s and σ. If A is finite, we denote the number of elements of A as $|A|$. The set of relations between positive integers is denoted $\Lambda = \{<, \leq, =, \neq, \geq, >\}$. Finally, let M, $M' \in \mathbb{N}^n$. We write $M \leq M'$ if $M(k) \leq M'(k)$ for all $k \in \{1, ..., n\}$.

3.2. Petri Nets

We will model our system as a Petri net (PN) with event labels associated with the transitions. The alphabet for our temporal logic specifications will be the union of the event set and linear constraints on the markings of the system.

Definition 1 (Petri Net)**.** A PN is a bipartite graph defined as $G = \langle P, T, W^+, W^-, M_0, E, \ell \rangle$. The two types of nodes are given by sets P (places) and T (transitions). Matrices $W^-, W^+ \in \mathbb{N}^{|P|\times|T|}$ represent, respectively, the arc weights from places to transitions, and the arc weights from transitions to places. The vector $M_0 \in \mathbb{N}^{|P|}$ is the initial marking, set E represents the events associated to the firing of transitions, with labelling function $\ell : T \to E$ assigning to each transition a label from E.

A marking is a distribution of *tokens* among places and represents a state of the system. Intuitively, places represent different components of the system, e.g., number of robots in a location or number of jobs in a queue. Transitions re labelled with an associated event, and represent the dynamics of the states of the system, i.e., changes in the marking of the PN. Changes of state can be due to the triggering of actions sent to the robots or the observation of a state update. These map to controllable and uncontrollable events as we will see later.

Definition 2 (Input and Output Weights)**.** Let $t \in T$. We define the vectors ${}^\bullet t \in \mathbb{N}^{|P|}$ (input weights of t) and $t^\bullet \in \mathbb{N}^{|P|}$ (output weights of t) as ${}^\bullet t(p) = W^-(p, t)$ and $t^\bullet(p) = W^+(p, t)$.

We say that p is an input place, or an output place, of t if ${}^\bullet t(p) > 0$, or $t^\bullet(p) > 0$, respectively.

Definition 3 (Enabled Transition)**.** Let $M \in \mathbb{N}^{|P|}$ and $t \in T$. We say that t is enabled if ${}^\bullet t \leq M$.

An enabled transition can fire, updating the marking of the PN according to the input and output weight vectors.

Definition 4 (Transition Firing)**.** Let t be an enabled transition in marking M. The firing of t evolves the marking to $M' = M - {}^\bullet t + t^\bullet$. This is denoted $M \xrightarrow{t} M'$.

The notion of firing of a transition can be extended to sequences of transitions.

Definition 5 (Firing Sequences)**.** Let $\tau = t_1...t_n \in T^*$. We write $M \xrightarrow{\tau} M'$ if exists $M_1, ..., M_{n-1} \in \mathbb{N}^{|P|}$ such that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} ... \xrightarrow{t_{n-1}} M_{n-1} \xrightarrow{t_n} M'$.

We can now define the set of reachable markings as the set of reachable markings in G, starting in M_0 and following the firing rule.

Definition 6 (Reachable Markings)**.** The set of reachable markings for PN G is defined as:

$$R(G) = \{M \in \mathbb{N}^{|P|} \mid \text{exists } \tau \in T^* \text{such that } M_0 \xrightarrow{\tau} M\} \tag{1}$$

Furthermore, we represent the behaviour of the PN as sets of *generated languages*.

Definition 7 (Generated Languages)**.** We define the following two languages generated by a labelled PN G:

$$\mathcal{L}^E_{fin}(G) = \{\ell(t_1)...\ell(t_n) \in E^* \mid \text{exists } M_1, ..., M_n \text{ such that } M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} ... \xrightarrow{t_n} M_n\} \tag{2}$$

$$\mathcal{L}(G) = \{(\ell(t_1), M_1)(\ell(t_2), M_2)... \in (E \times R(G))^\omega \mid M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} ...\} \tag{3}$$

As we will see later, the elements of $\mathcal{L}^E_{fin}(G)$ will be used in the supervisory control loop, and the elements of $\mathcal{L}(G)$ will be used to evaluate the LTL specifications. Also, note that M_1 must be reachable from the initial marking M_0 through the firing of t_1, but M_0 is not included in $\mathcal{L}(G)$. This is because we assume, without loss of generality, that G has place *init* $\in P_g$ and an *initialise_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 *initialise_system* transition, which consumes the token of the *init* place and distributes tokens to the other places of G, according to the real initial state of the system. This ensures that the initial marking of the system will be taken into account by our composition algorithm. We require the PNs in this work to be deterministic, to ensure that the supervisor can keep track of the current marking of the PN model of the system during execution while only observing the generated sequence of events.

Definition 8 (Deterministic PN)**.** Let G be a PN. We say that G is deterministic if in all of its reachable markings $M \in R(G)$, if $M \xrightarrow{t} M'$, $M \xrightarrow{t'} M''$ and $M' \neq M''$, then $\ell(t) \neq \ell(t')$.

Finally, we assume that P is partitioned into $P_\omega \cup P_b$, where P_ω is the set of unbounded places and P_b is the set of bounded places.

Definition 9 (Bounded Place)**.** Let $p \in P$. We say that p is bounded, with bound $\kappa(p)$, if for all $M \in R(G)$, $M(p) \leq \kappa(p)$.

We also assume that, for each $p \in P_b$, its complement place $\overline{p}$ is also in P_b.

Definition 10 (Complement Place)**.** Let $p \in P_b$ be a bounded place. We define its complement place as the place $\overline{p}$ such that $M(p) + M(\overline{p}) = \kappa(p)$ for all $M \in R(G)$. Note that $\kappa(\overline{p}) = \kappa(p)$ and $\overline{\overline{p}} = p$.

Note that if we know the bound for a place, we can add its complement place to the PN without changing its behaviour [46]. Thus, we do not lose generality with our assumption.

Example 1. We introduce the multi-robot coordination example that will be built throughout the paper. Assume a team of n_R robots that are in charge of performing a set $J = \{j_1, ..., j_{n_J}\}$ of different types of jobs. We assume that there is a maximum number n_{j_i} of jobs of type j_i that are allowed to be processed by the system at any given time. Furthermore, robots might fail while executing tasks. When this happens, the robot gets into a broken state, and the task goes to a queue of incomplete tasks. The PN is Figure 2 represents this system, for task j_i.

Also, when a robot breaks, other robots can repair it. When doing so, robots might be successful (leading to both robots being able to continue executing tasks), fail (leading to both robots being considered broken), or discover that the

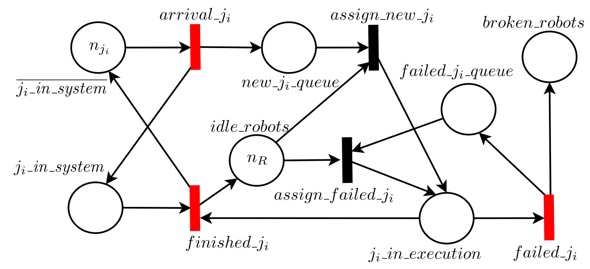

Figure 2: PN model for job allocation. Complement places that do not restrict the language of the model as omitted for readability. Transitions in red are uncontrollable.

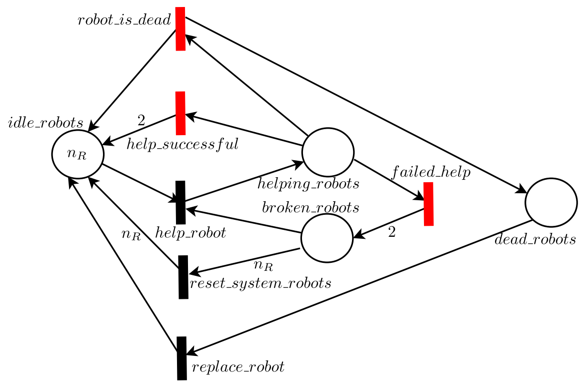

Figure 3: PN model for robot repair. Complement places that do not restrict the language of the model as omitted for readability. Transitions in red are uncontrollable.

broken robot is not repairable (leading to a state where the broken robot needs to be replaced). This repair module of the system is represented by the PN in Figure 3. The overall model is obtained by merging places with the same name in Figs. 2 and 3.

For simplicity, we do not depict all the complement places. Furthermore, we define $E = T$, and $\ell(t) = t$, i.e., we will refer the transitions by the event they represent, since all events are unique. Also, for clarity, we depict the models without the *init* place and *initialise_system* transition. The *initialise_system* transition would put n_R tokens in place *idle_robots*, and n_{j_i} tokens in $\overline{j_i\_in\_system}$, for each $j_1, ..., j_{n_J}$.

3.3. Supervisory Control

The purpose of supervisory control (SC), as introduced in [45] is, given a model of the uncontrolled behaviour of a system, to restrict its behaviour to an admissible language $\mathcal{L}_a \subseteq \mathcal{L}(G)$, through a feedback loop. In this subsection, we provide a brief overview on SC with language specifications for PNs (see [11, 33] for further details).

One starts by partitioning the event set E into controllable and uncontrollable transitions.

Definition 11 (Controllable and Uncontrollable Events)**.** We split the set of events E into two disjoint subsets $E = E_c \cup E_{uc}$. We call E_c the set of controllable events and E_{uc} the set of uncontrollable events.

E_c is the set of events that can be prevented from happening by the supervisor and E_{uc} is the events that cannot be prevented from happening. This partition is due to the fact that, in general, there are events that make a system change its state that are not of the "responsibility" of the system itself (e.g., failures in execution). The set of uncontrollable events induces a set of uncontrollable transitions for G.

Definition 12 (Uncontrollable Transitions)**.** We define the set of uncontrollable transitions as $T_{uc} = \{t \in T \mid \ell(t) \in E_{uc}\}$.

Example 2. In the PN models depicted in Figs. 2 and 3, controllable transitions, corresponding to actions that can be taken on the system are depicted in black, and uncontrollable transitions, corresponding to events that represent the arrival of new jobs or a given action being successful, or unsuccessful, are depicted in red.

Definition 13 (Supervisor)**.** A *supervisor* for G is a function $S : \mathcal{L}^E_{fin}(G) \rightarrow 2^E$ that, given $s \in \mathcal{L}^E_{fin}(G)$, outputs the set of enabled events, i.e., the set of events that G can execute next.

A supervisor restricts the language generated by G such that some behaviour specification for the system is fulfilled.

Definition 14 (Controlled Language)**.** The language generated by G when controlled by S is then given by:

$$\begin{aligned}\mathcal{L}(S/G) = \{&(e_1, M_1)(e_2, M_2)... \in \mathcal{L}(G) \mid \\ &e_{i+1} \in S(e_1...e_i) \text{ for all } i \in \mathbb{N}\}\end{aligned} \tag{4}$$

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 enabled in its current state and which is enabled by S.

For analysis and implementation purposes, it is important to represent the supervisor in a convenient way. In this work, we will use a PN G_φ to represent the supervisor. In order for G_φ to be implementable in the real system, we need to guarantee that it is *admissible*. Intuitively, admissibility means that the supervisor never disables enabled uncontrollable events. We will discuss admissibility in Section 5. If the composition is admissible, then it can be used to control the system, i.e., we can directly represent the supervisor as a PN, using G_φ. The feedback loop is then implemented as follows: at each step, G executes an event e, according to the enabled events in its current state and the current enabled events by G_φ, evolving to a new marking. This event is then sent to G_φ, which passively executes e, also evolving to a new marking. The set of enabled events after the execution of e is the set of enabled events of G_φ in the new marking. Representing supervisors with the same formalism used to model for system modelling also gives us analysis benefits. These benefits stem from the fact that, for this case, $\mathcal{L}(S/G) = \mathcal{L}(G_\varphi)$, i.e., G_φ also models the closed-loop behaviour of the system. Thus, we can use all the analysis techniques available for PNs on the controlled system.

3.4. Linear Temporal Logic

Linear temporal logic (LTL) is an extension of propositional logic developed as a means for formal reasoning about concurrent systems [44]. It provides a convenient and powerful way to formally specify a variety of qualitative properties of a system.

Definition 15 (Syntax)**.** The syntax of LTL is defined over a set of atomic propositions Π using the grammar below.

$$\varphi ::= true \mid \pi \mid \neg\varphi \mid \varphi\wedge\varphi \mid \mathtt{X}\,\varphi \mid \varphi\,\mathtt{U}\,\varphi, \text{ where } \pi \in \Pi \tag{5}$$

We evaluate LTL formulas over ω-strings $\sigma = \sigma_0\sigma_1\sigma_2... \in \mathcal{L}(G)$. Thus, we define the set of atomic propositions as $\Pi = E \cup (\mathbb{Z}^{|P_b|} \times \Lambda \times \mathbb{N})$.

Definition 16 (Atomic Propositions Satisfaction)**.** Let $\sigma_i = (e_i, M_i) \in E\times\mathbb{N}^{|P|}$ be an (event/marking) pair. $\sigma_i = (e_i, M_i)$ satisfies $\pi \in \Pi$ if (i) $\pi = e_i \in E$; or (ii) $\pi = (v, \bowtie, b)$ with $v \in \mathbb{Z}^{|P_b|}$, $\bowtie \in \Lambda$ and $b \in \mathbb{N}$, and $\sum_{p\in P_b} v(p)M_i(p) \bowtie b$.

Intuitively, $\pi = e$ means that the last event to occur was e, and $\pi = (v, \bowtie, b)$ is a linear constraint requiring $\sum_{p\in P_b} v(p)M(p) \bowtie b$ to hold for the current marking M. Note that we are allowing the writing of *state/event* specifications, i.e., we allow reasoning over both events *and* markings generated by the system.

Now that we know how to evaluate atomic propositions on $\sigma \in \mathcal{L}(G)$, we can define the LTL operators in the usual way (see, for example, [44] for a formal semantics): The $\mathtt{X}$ operator is read "next", meaning that the formula it precedes will be true in the next state. The $\mathtt{U}$ operator is read "until", meaning that its second argument will eventually become true in some state, and the first argument will be continuously true until that state is reached. We write $\sigma \Vdash \varphi$ to denote that σ satisfies φ.

Other useful LTL operators can be derived: (i) the "eventually" operator $\mathtt{F}\,\varphi$, which requires that φ is satisfied in some future state: $\mathtt{F}\,\varphi \equiv true\,\mathtt{U}\,\varphi$; (ii) the "always" operator $\mathtt{G}\,\varphi$, which requires that φ is satisfied in all states: $\mathtt{G}\,\varphi \equiv \neg\,\mathtt{F}\,\neg\varphi$; and (iii) the "weak until" operator $\varphi\,\mathtt{W}\,\psi$ which relaxes the until operator to include ω-strings for which ψ is never satisfied but φ is always satisfied: $\varphi\,\mathtt{W}\,\psi \equiv (\varphi\,\mathtt{U}\,\psi) \vee (\mathtt{G}\,\varphi)$.

Example 3. For the PN model depicted in Figs. 2 and 3, one might want to specify that if the number of working robots is greater or equal than the current number of jobs in the system, then we should not require dead robot replacement, as this requires human intervention and should only be done when the resources start becoming low. This can be written as the following LTL specification:

$$\begin{aligned}\varphi_1 = \mathtt{G}((&n_R - M(broken\_robots) - M(dead\_robots) \\ &\geq (\sum_{j_i\in J} M(j_i\_in\_system))) \Rightarrow (X(\neg replace\_robot)))\end{aligned} \tag{6}$$

Note that $n_R - M(broken\_robots) - M(dead\_robots) \geq (\sum_{j_i\in J} M(j_i\_in\_system))$ is a linear constraint on the system.

We write it using this form for readability. The formal specification is $(v, \leq, n_R)$, where $v(p) = 1$ if $p \in \{broken\_robots,\ dead\_robots,\ j_i\_in\_system \mid j_i \in J\}$, and $v(p) = 0$ otherwise.

3.5. Safe LTL and Deterministic Automata

In SC theory, we are generally interested in restricting the behaviour of a system so that it satisfies a given specification, i.e., we are interested in avoiding sequences of execution that are considered "bad". The class of LTL that specifies properties that state that something "bad" will never happen is a well-defined class, called *safe* LTL [28, 49]. These are formulas for which the non-satisfying infinite sequences always have a finite *bad prefix*.

Definition 17 (Bad Prefix)**.** Let φ be an LTL formula and $\sigma = \sigma_0\sigma_1... \in \mathcal{L}(G)$ such that $\sigma \not\Vdash \varphi$. We say σ has a bad prefix if there exists $n \in \mathbb{N}$ for which the truncated finite sequence $\sigma|_n = \sigma_0\sigma_1...\sigma_n$ is such that $\sigma|_n{\cdot}\sigma' \not\Vdash \varphi$ for all infinite sequences $\sigma' \in (2^{\Pi})^{\omega}$.

Definition 18 (Safe LTL)**.** We say that the LTL formula φ is safe if *all* the infinite sequences σ such that $\sigma \not\Vdash \varphi$ have a bad prefix.

Intuitively, a safe LTL formula is a formula that can always be falsified in a finite horizon. It is known that formulas in the positive normal form (i.e., where negation is only applied directly to atomic propositions) which only use the temporal operators X, G and W are safe. We will keep our specifications within this syntactic restriction.

Safe LTL formulas φ can be translated into deterministic finite automata (DFA)[1].

Proposition 1 ([28])**.** *Let φ be a safe LTL formula. There exists a DFA $A_\varphi = \langle Q, 2^{\Pi}, \delta, q_0, Q_f \rangle$ where (i) Q is a set of states; (ii) $\delta : Q \times 2^{\Pi} \to Q$ is a (partial) deterministic transition function; (iii) q_0 is an initial state; and (iv) Q_f is a set of accepting states; that accepts exactly the sequences that are* not *a bad prefix for φ.*

The fact that φ is safe also allows us to assume, without loss of generality, that (i) A_φ is *pruned*, i.e., there are no *sink* states from which all accepting states are unreachable; and (ii) $Q_f = Q$. Furthermore, in order to describe the transition labels (elements of 2^{Π}) in a more compact way, we use *DNF transition labels*, i.e., we label the transitions with propositional formulas over Π, in the disjunctive normal form (DNF). A propositional formula is said to be in the DNF if it is a disjunction of conjunctive clauses. A conjunctive clause is the conjunction of positive and negative literals. The labels of the transition are then the pairs $(e, M) \in 2^{\Pi}$ that satisfy the DNF formula. We write $(e, M) \Vdash \psi$ when ψ is a DNF formula satisfied by (e, M). Also, the satisfaction of conjunctive clauses ψ_E written only over E only depends on e. Thus, we write $e \Vdash \psi_E$ if e satisfies ψ_E.

[1]Note that, given the semantics over infinite sequences, in general one needs automata that accept infinite strings (such as Büchi automata) to represent LTL formulas. However, for the safety fragment of LTL DFA suffice, and make the presentation clearer.

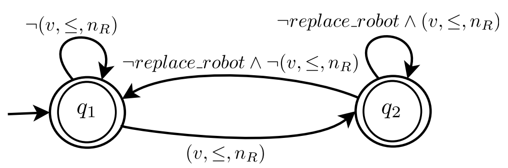
$\neg(v, \leq, n_R)$
$\neg replace\_robot \wedge (v, \leq, n_R)$
$\neg replace\_robot \wedge \neg(v, \leq, n_R)$
q_1
q_2
$(v, \leq, n_R)$

Figure 4: DFA for formula φ_1. States are labelled q_1 and q_2.

Example 4. The LTL formula in Ex. 3 is syntactically safe. The DFA A_{φ_1} that only accepts the sequences which are not a bad prefix for φ_1 is depicted in Figure 4.

All atomic propositions (e, M) such that $n_R - M(broken\_robots) - M(dead\_robots) \geq (\sum_{j_i \in J} M(j_i\_in\_system))$ are transition labels of the transition from q_1 to q_2, while only atomic propositions that satisfy the same condition over M *and* are such that $e \neq replace\_robot$ are transition labels of the self-loop from q_2 to q_2.

4. Composition of PN System Model with DFA

In this section, we present the composition algorithm between a PN system model G and the DFA obtained from a safe LTL specification φ. The result of this composition is a PN G_φ whose generated language is the generated language of G restricted to the ω-strings that satisfy φ. Formally, we will tackle the following problem:

Problem 1. Let $G = \langle P_b \cup P_\omega, T, W^-, W^+, M_0, E, \ell \rangle$ be a PN and φ be a safe state/event LTL formula written over $\Pi = E \cup (\mathbb{Z}^{P_b} \times \Lambda \times \mathbb{N})$. Build a PN G_φ such that:

$$\sigma \in \mathcal{L}(G_\varphi) \text{ if and only if } \sigma \in \mathcal{L}(G) \text{ and } \sigma \Vdash \varphi \tag{7}$$

4.1. From General Linear Constraints to SP-GEQ-Cs

We start by showing how one can translate arbitrary literals of the form $(v, \bowtie, b)$ to *single place greater-or-equal constraints* (SP-GEQ-Cs). An SP-GEQ-C is of the form $(v_s, \geq, b)$, where v_s is a vector with exactly one entry equal to 1, and all other entries equal to 0. We will represent all DNF labels in the DFA obtained for the safe LTL formulas using SP-GEQ-Cs, as this representation is required to define the notion of *minimal satisfying transition*, which will be central to the composition algorithm. We split the translation into 4 steps. These steps are more or less straightforward and use known properties of bounded PNs and of complement places so, for the sake of brevity, we omit the details. We refer the reader to [29] for a thorough description of the translation.

4.1.1. From Negative Literals to Positive Literals

Firstly, we convert any negative literal into a positive literal over $(\mathbb{Z}^{|P_B|} \times \Lambda \times \mathbb{N})$, by noting that the negation of a linear constraint can always be transformed in an equivalent "positive" linear constraint by changing the relation accordingly. For example, $\neg(v, \geq, b)$ can be re-written as $(v, <, b)$.

4.1.2. *From General Constraints to GEQ and LEQ Constraints*

One can now convert any positive literal $(v, \bowtie, b)$ into greater-or-equal (GEQ) or less-or-equal (LEQ) constraints, by taking into account that these are constraints over integers. For example, one can write $\sum_{p\in P_b} v(p)M(p) \geq b+1$ instead of $\sum_{p\in P_b} v(p)M(p) > b$.

4.1.3. *From GEQ and LEQ Constraints to Positive GEQ Constraints*

After preforming the two conversion steps described above, all the literals in the DFA transition labels are of the form $(v, \bowtie, b) \in \mathbb{Z}^{|P_B|} \times \{\leq, \geq\} \times \mathbb{N}$. We can now convert each constraint to a *positive GEQ constraint*, i.e., a constraint $(v', \geq, b') \in \mathbb{N}^{|P_B|} \times \{\geq\} \times \mathbb{N}$ equivalent to $(v, \bowtie, b)$. This can be done by re-writing the LEQ constraints and the negative weights in v, using the relations between places and their complement places, and using the corresponding place bounds. For example, if $\kappa(p) = 3$ a constraint that imposes $M(p) \leq 2$ can be re-written as $M(\overline{p}) \geq 1$.

4.1.4. *From Positive GEQ Constraints to SP-GEQ-Cs*

We can now assume that we are dealing with constraints $(v, \geq, b) \in \mathbb{N}^{|P_B|} \times \{\geq\} \times \mathbb{N}$. In order to re-write these constraints as SP-GEQ-Cs, we add, for each $(v, \geq, b)$, an extra place p_v in the PN. We call this place *counter place*, as it is used to "count" the sum of tokens in M weighted by v, i.e., for all $M \in R(G)$:

$$M(p_v) = \sum_{p\in P_b} v(p)M(p). \tag{8}$$

Definition 19 (Counter Place)**.** Let $G = \langle P_b \cup P_\omega, T, W^-, W^+, M_0, E, \ell\rangle$ be a PN and $(v, \geq, b) \in \mathbb{N}^{|P_B|} \times \{\geq\} \times \mathbb{N}$ a constraint. The addition of a counter place for v to G, yields the PN $G^v = \langle P \cup \{p_v\}, T, M_0^v, W^{v-}, W^{v+}, E, \ell\rangle$, where, for p_v:

- $M_0^v(p_v) = v^T M_0 = \sum_{p\in P_b} v(p)M_0(p)$;
- For $t \in T$, $W^{v-}(p_v, t) = \max\{-W^v(p_v, t), 0\}$ and $W^{v+}(p_v, t) = \max\{W^v(p_v, t), 0\}$, where:

  $$W^v(p_v, t) = \sum_{p\in P_b} v(p)(W^+(p, t) - W^-(p, t)) \tag{9}$$

  Furthermore $W^{v-}(p, t) = W^-(p, t)$ and $W^{v+}(p, t) = W^+(p, t)$ for all $p \in P$, $t \in T$.

Note that the addition of the counter place does not change the behaviour of the PN, and one can see G^v as an extension of G that keeps track of the sum of the number of tokens in certain places.

Thus, after the counter place p_v is built, $(v, \geq, b)$ can be re-written as an SP-GEQ-C $(v_{p_v}, \geq, b)$, where $v_{p_v}(p_v) = 1$ and $v_{p_v}(p) = 0$ for all other $p \in P$. For simplicity we will denote SP-GEQ-Cs as $(p_v, \geq, b)$ in the remainder of the paper.

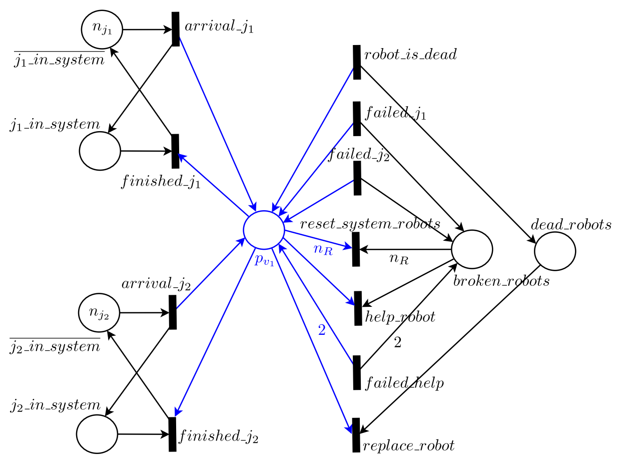

Figure 5: Illustration of the counter place construction.

Example 5. Consider the linear constraint $\neg(v_1, \leq, n_R)$, where v_1 is defined in Ex. 3, and the PN models of Ex. 1. We illustrate the full conversion procedure:

1. Convert the constraint into a positive literal $(v_1, >, n_R)$;
2. Convert the constraint into a GEQ constraint $(v_1, \geq, n_R - 1)$;
3. Since v_1 is already in $\mathbb{N}^{|P_B|}$ and the constraint is already a GEQ constraint, we do not need to modify it;
4. Build the counter place p_{v_1} for v_1. The construction is depicted in blue in Figure 5, where we assume two types of jobs, and only depict the relevant places and transitions of the model. The final SP-GEQ-C, equivalent to the initial constraint, is $(p_{v_1}, \geq, n_R - 1)$.

4.2. Adding Minimal Satisfying Transitions

We now describe the construction of minimal satisfying transitions for conjunctive clauses of the form $\psi = \psi_E \wedge \psi_P$, where $\psi_E = l_1 \wedge \ldots \wedge l_{k_E}$ is a conjunctive clause over E, and $\psi_P = (p_1, \geq, b_1) \wedge \ldots \wedge (p_{k_P}, \geq, b_{k_P})$ is a conjunctive clause, where all the literals are SP-GEQ-Cs.

Definition 20 (Minimal Satisfying Transition)**.** Let $\psi = \psi_E \wedge \psi_P$, where $\psi_E = l_1 \wedge \ldots \wedge l_{k_E}$. We define the set of places associated with ψ as $P_\psi = \bigcup_{i=1}^{k_P} p_i$ and the function $b : P_\psi \to \mathbb{N}$ that maps each p_i to the corresponding b_i, i.e., $b(p_i) = b_i$, $i = 1, \ldots, k_P$. 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:

$${}^\bullet t_{t,\psi}(p) = \begin{cases} {}^\bullet t(p) & \text{if } p \notin P_\psi \text{ or } b(p) \leq t^\bullet(p) \\ b(p) + {}^\bullet t(p) - t^\bullet(p) & \text{otherwise} \end{cases} \tag{10}$$

$$t^\bullet_{t,\psi}(p) = \begin{cases} t^\bullet(p) & \text{if } p \notin P_\psi \text{ or } b(p) \leq t^\bullet(p) \\ b(p) & \text{otherwise} \end{cases} \tag{11}$$

In the cases where $\ell(t) \not\Vdash \psi_E$, or exists $p \in P_\psi$ such that $\kappa(p) - {}^\bullet t(p) + t^\bullet(p) < b(p)$, we say that the minimal satisfying transition is undefined.

The minimal satisfying transition for t and ψ is enabled if and only if (i) t is enabled; (ii) $\ell(t)$ satisfies the conjunctive clause ψ_E; and (iii) the firing of t drives the PN to a marking that satisfies all the SP-GEQ-Cs in ψ_P. Furthermore, when the minimal satisfying transition is not defined, then either (i) $\ell(t)$ does not satisfy the conjunctive clause ψ_E; or (ii) all the firings of t will drive the PN to a marking M where at least one SP-GEQ-C $(p_i, \geq, b_i)$ is not satisfied, i.e., $M(p_i) < b_i$.

Proposition 2. *Let G be a PN, $t \in T$, and $\psi = \psi_E \wedge \psi_P$, where ψ_E is a conjunctive clause written over E and ψ_P is a conjunctive clause of SP-GEQ-Cs. The minimal satisfying 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 $M \xrightarrow{t} M'$, $\ell(t) \Vdash \psi_E$ 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 either $\ell(t) \not\Vdash \psi_E$, or 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., $(M, \ell(t)) \not\Vdash \psi$.*

Proof. We note that the part of the result concerning $\ell(t)$ and ψ_E follows directly from the definition. Thus, we just need to analyse ψ_P and the possible markings obtained immediately after the firing of t.

1. We start by noting that the input and output weights 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 $b(p) > t^\bullet(p)$.

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

   $$M(p) \geq {}^\bullet t_{t,\psi}(p) = b(p) + {}^\bullet t(p) - t^\bullet(p) > {}^\bullet t(p) \tag{12}$$

   (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:

   $$\begin{aligned} &M(p) - {}^\bullet t_{t,\psi}(p) + t^\bullet_{t,\psi}(p) = \\ &M(p) - b(p) - {}^\bullet t(p) + t^\bullet(p) + b(p) = \\ &M(p) - {}^\bullet t(p) + t^\bullet(p) \end{aligned} \tag{13}$$

   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 $p \in P_\psi$.

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

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

   $$\begin{aligned} &M'(p) = \\ &M(p) - {}^\bullet t(p) + t^\bullet(p) = \\ &M(p) - {}^\bullet t_{t,\psi}(p) + b(p) - t^\bullet(p) + t^\bullet(p) = \\ &M(p) - {}^\bullet t_{t,\psi}(p) + b(p) \end{aligned} \tag{14}$$

   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:

   $$\begin{aligned} M'(p) =& M(p) - {}^\bullet t(p) + t^\bullet(p) < \\ & M(p) + b(p) - \kappa(p) \leq b(p) \end{aligned} \tag{15}$$

Hence the proof is completed. □

Example 6. Let $\gamma = \neg replace\_robot \wedge (p_{v_1}, \geq, n_R - 1)$, where p_{v_1} is the counter place defined in Ex. 5. We assume $n_R > 1$, otherwise the SP-GEQ-C is trivially satisfied in all markings. We will analyse transitions $replace\_robot$ and $help\_robot$. For transition $replace\_robot$, $replace\_robot \not\Vdash \neg replace\_robot$, thus $t_{replace\_robot,\gamma}$ is not defined. For transition $help\_robot$, $help\_robot \Vdash \neg replace\_robot$. Furthermore, we have $help\_robot^\bullet(p_{v_1}) = 0 \ngeq b(p_{v_1}) = n_R - 1$, so we change the arc weights to ${}^\bullet t_{help\_robot,\gamma}(p_{v_1}) = b(p_{v_1}) + {}^\bullet help\_robot(p_{v_1}) - help\_robot^\bullet(p_{v_1}) = (n_R - 1) + 1 - 0 = n_R$ and $t^\bullet_{help\_robot,\gamma}(p_{v_1}) = b(p_{v_1}) = n_R - 1$. Figure 6 depicts the result of the minimal satisfying transition $t_{help\_robot,\gamma}$, as just described. We only depict the relevant places and transitions for its construction. The transitions with arc changes are depicted in blue. For this example, the places and arcs depicted in green are to be ignored, as they are related to the composition algorithm that will be described next.

4.3. Composition Algorithm

To solve Problem 1, we define a composition function that, given the PN G and the DFA A_φ, builds a PN G_φ that simulates the run of the PN and the DFA in parallel, such that G only fires a transition t if the firing of t satisfies one of the enabled DNF transition labels in the current state of A_φ. The construction of this PN follows Algorithm 1. The algorithm is based on the notion of minimal satisfying transition, so we assume that all the DNF formulas labelling the DFA transitions have either literals over E or SP-GEQ-Cs,

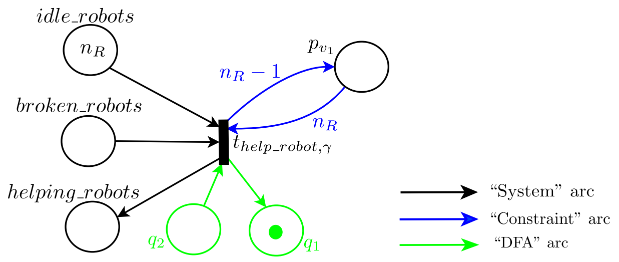

Figure 6: Synchronising transition *help_robot* with the self-loop of state q_2 of the DFA depicted in Figure 4.

Algorithm 1 DFA/PN Composition

```
Input: PN $G = \langle P, T, W^+, W^-, M_0, E, \ell \rangle$, and syntactically safe LTL formula $\varphi$, written over $\Pi$
Output: PN $G_\varphi = \langle P_\varphi, T_\varphi, W_\varphi^+, W_\varphi^-, M_{\varphi,0}, E, \ell_\varphi \rangle$
 1: $A_\varphi = \langle Q, 2^\Pi, \delta, q_0, Q_f \rangle$
 2: $P_\varphi \leftarrow P$; $M_{\varphi,0} \leftarrow M_0$
 3: $queue.push(q_0)$
 4: add place $p$ with label $q_0$ to $P_\varphi$; $M_{\varphi,0}(p) \leftarrow 1$
 5: while $queue \neq \emptyset$ do
 6:     $q \leftarrow queue.pop()$
 7:     for all $t \in T$ do
 8:         for all $(\psi, q')$ such that $\delta(q, \psi) = q'$ do
 9:             for all conjunctive clauses $\gamma$ in $\psi$ do
10:                 if $t_{t,\gamma}$ is defined then
11:                     if $\nexists p \in P_\varphi$ with label $q'$ then
12:                         $queue.push(q')$
13:                         add place $p'$ with label $q'$ to $P_\varphi$; $M_{\varphi,0}(p') \leftarrow 0$
14:                     end if
15:                     add transition $t_{t,\gamma}$ to $T_\varphi$; $\ell_\varphi(t_{t,\gamma}) \leftarrow \ell(t)$
16:                     ${}^\bullet t_{t,\gamma}(p) \leftarrow 1$, where $p \in P_\varphi$ is labelled $q$
17:                     $t_{t,\gamma}^\bullet(p') \leftarrow 1$, where $p' \in P_\varphi$ is labelled $q'$
18:                 end if
19:             end for
20:         end for
21:     end for
22: end while
```

and that the corresponding counter places have already been added to G.

We start by analysing the initial state q_0 of the DFA, and only analyse states that are attained during the execution of the algorithm, i.e., states added to *queue* (lines 11–14). These states are added as places to G_φ. At each marking of G_φ, the "DFA place" representing the current state of the DFA has one token and all other "DFA places" have zero tokens. This can be seen as representing the DFA as a PN structure, and allows us to represent the result of the composition as a PN.

When analysing a state q, taken from *queue*, we iterate over all DNF formulas ψ labelling transitions from q and, for each transition of the input PN, build the minimal satisfying transition for each conjunctive clause γ in ψ. For each of these conjunctive clauses, if the minimal satisfying transition $t_{t,\gamma}$ is defined, we add it to G_φ. Furthermore, to ensure that we keep the DFA evolution encoded in G_φ, we add the place representing q as an input place, and the place representing $\delta(q, \psi)$ as an output place to $t_{t,\gamma}$ (lines 7–21). Note that if the minimal satisfying transition is not defined, it means that the original transition cannot be executed without falsifying φ, hence we do not add anything to G_φ.

To summarise, for each transition $t_{t,\gamma}$ added to the output PN, there are 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. *DFA arcs*, i.e., arcs that consume a token from the place representing the current state of the DFA and put one token in the place representing the next state of the DFA. These arcs take into account the evolution of the DFA;
3. *Constraint arcs*, obtained from the minimal satisfying transition construction. These arcs do not change the token flow (with respect to the original PN) in the places they are related to. Furthermore, they guarantee that any marking obtained immediately after the firing of $t_{t,\gamma}$ satisfies γ (according to the properties of minimal satisfying transitions proved in Proposition 3).

Hence, the result of the composition simulates the running in parallel of the PN and the DFA, where we add transitions that represent all the possible firings of transitions in the PN that can satisfy a DNF label in the DFA, evolving both structures accordingly. Given that we synchronise all transitions in both models (as in the parallel composition of PNs where transition synchronisation is done using the events of each model, see for example [11]), and the notion of minimal satisfying transition is ocrrect as proved in Prop. 3, we can ensure the correctness of the composition algorithm.

Proposition 3. *Algorithm 1 solves Problem 1.*

Proof. The counter place represents a linear constraint $(v, \bowtie, b)$ while maintaining the language of the PN. Thus, after building counter places for all linear constraints in φ, one just needs to ensure that the corresponding SP-GEQ-Cs over the counter places are kept valid according to the transitions of A_φ, while not restricting the firing of transitions more than needed to guarantee that validity. The minimal satisfying transitions exactly represent the minimal restriction over a transition such that its firing is guaranteed to evolve the PN to a marking such that the SP-GEQ-C is satisfied, as proved in Proposition 2. Algorithm 1 generalises the notion of event-based parallel composition of PNs as described in [11], synchronising all transitions between the PN and the DFA, using the minimal satisfying transition notion, and keeping track of the marking evolution of G and the state evolution of A_φ. Thus, the result follows. □

Example 7. Algorithm 1 needs to synchronise all transitions of G with all transitions from DFA states that are queued (lines 6 and 12). Consider the PN model G in Figs. 2 and 3, and the DFA A_φ in Figure 4. Assume that all transitions from state q_1 of A_φ have been processed, and we are currently processing transition *help_robot* of G, and the transition in A_φ from q_2 to q_1, labelled by $\neg replace\_robot \wedge \neg(v_1, \leq, n_R)$. As seen in Ex. 5, this label is equivalent to $\neg replace\_robot \wedge (p_{v_1}, \geq, n_R - 1)$, where p_{v_1} is the associated counter place. The construction of the minimal satisfying transition for *help_robot* and $\neg replace\_robot \wedge (p_{v_1}, \geq, n_R - 1)$ was illustrated in Ex. 6. Thus, we just need to add arcs representing the evolution of the DFA from q_2 to q_1, as depicted in green in Figure 6.

4.4. Additional Remarks

4.4.1. Complexity of the Composition

It is known that for an LTL formula φ, the number of states of the deterministic automaton A_φ is, in the worst-case, doubly exponential on the size of φ, i.e., $|Q| = 2^{2^{|\varphi|}}$ [28]. Furthermore, the number of transitions is at most $|Q|2^{|\Pi_\varphi|}$, where $\Pi_\varphi \subseteq \Pi$ is the set of atomic proposition that occur in φ. This is because the maximum number of transitions from a state is the size of the DFA's alphabet. Thus, in the worst-case, $|P_\varphi| = |P| + 2^{2^{|\varphi|}}$ and $|T_\varphi| = |T||Q|2^{|\Pi_\varphi|} = |T|2^{(2^{|\varphi|}+|\Pi_\varphi|)}$. Thus, the composition has complexity $O(|P| + |T|2^{(2^{|\varphi|}+|\Pi_\varphi|)})$. We note that the exponential blow-up on the size of the formula is unavoidable as it stems from the translation of the formula to the DFA. However, in general the size of the LTL specifications can be kept small, with our semantics based on linear constraints being an extra factor for such an argument, as we discuss next. Furthermore, the worst-case scenario for the automaton size is rarely attained, with state-of-the-art LTL to automaton translators generating automata that are much smaller than the worst-case for most LTL formulas.

4.4.2. Dead Transitions

The use of a structural approach allows us to avoid the state space explosion. However, our composition requires synchronising *all* transitions in the PN with *all* transitions in the DFA, to ensure that the DFA component of the composition evolves according to all the events fired in the original PN. Several of these transitions might not be enabled by any of the reachable markings in the PN. However, this cannot be analysed at composition time because it would require building the state space. In general, the computational savings of not generating the state space of the PN outweigh the impact of the generation of these *dead* transitions. Their presence does not affect the overall qualitative behaviour of the system, i.e., the generated controlled language is still the same. Hence, their main impact is on execution and analysis terms, as there is a large subset of transitions which, if not pruned, impact the process of finding the enabled transitions in a certain marking. In order to perform the pruning of dead transitions, one can use several techniques. Examples are building the coverability graph of the composition, and remove transitions that do not appear there, or using an incomplete structural approach, based on the notion of *firing count vector*, as described in [32]. In future work, we will investigate more involved approaches for removing dead transition, e.g., [1].

4.4.3. Complexity Benefit of Linear Constraints

In other works relating LTL and PNs, the proposed semantics of LTL is more restrictive than the one we present here. In general, LTL formulas are evaluated over PN models by associating a set of atomic propositions D to a subset of 1-bounded places (i.e., places p such that $\kappa(p) = 1$). Then, d is satisfied in a marking M iff $M(p_d) = 1$, where p_d is the 1-bounded place associated to d. Our approach is not only more general but also can yield smaller supervisors, by allowing the representation of some propositional connectives by linear constraints. For example, let $D' = \{d_1, ..., d_n\} \subseteq D$ and $v_{D'}$ a vector such that $v_{D'}(p) = 1$ if $p = p_{d_i}$ for some $d_i \in D'$ and 0 otherwise. A formula $\bigvee_{d_i \in D'} d_i$, can be represented by $(v_{D'}, \geq, 1)$; a formula $\bigwedge_{d_i \in D'} d_i$, can be represented by $(v_{D'}, \geq, n)$; more generally, consider the so-called *cardinality constraints*, i.e., constraints of the form "at least m of the n atomic propositions in D' need to be satisfied" or "at most m of the n atomic proposition in D' can be satisfied". To the best of our knowledge, the best lower-bound for the length of a propositional logic formula encoding such a constraint is in $O(n \log^2 m)$ [2]. We can represent "at least" constraints by $(v_{D'}, \geq, m)$ and "at most" constraints by $(v_{D'}, \leq, m)$, i.e., by a single atomic proposition.

4.4.4. Generalised Mutual Exclusion Constraints

The inspiration to use linear constraints came from the so called generalised mutual exclusion constraints (GMEC) approach [16, 21, 40, 22]. The GMEC approach is a form of *state-based* supervision, 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 ours. In fact, the construction of the counter place we presented here has similarities with the basic construction of monitors. Loosely speaking, a "GMEC specification" $(v, \bowtie, b)$ is equivalent to the safe LTL specification $G(v, \bowtie, b)$. Thus, our approach can be seen as an alternative to GMECs, that deals with specifications of the same flavour but, instead of only allowing for "static" linear restrictions that must always hold, allows for the specification of "dynamic" safe LTL relations between GMECs that can change due to the sequence of markings and/or events already executed. The GMEC approach has the advantage of being able to deal with unbounded places. This is because GMEC *always* keeps the number of tokens in the places above or below a given constant, being only applicable when the initial marking already satisfies the linear constraint. In our case, we must check the number of tokens in certain places, because we always need to know what is the marking after the firing of a transition in order to make the DFA evolve to the correct state. Since being able to check whether an unbounded place has 0 tokens increases the modelling power of the PN to the equivalent of a Turing machines (which is not desirable as most problems become undecidable), we cannot hope to reason directly about unbounded places.

4.4.5. Event Based DFA and PN Language Specifications

Our approach builds over the more traditional event based language specification approaches, by allowing for state/event specifications, where the state specifications are "GMEC-like". Hence, our composition algorithm can be seen as bringing together ideas from two well established techniques for SC of PNs: GMECs and language specifications. We also note that, while our algorithm receives a safe state/event LTL specifications as input, it can be easily generalised to specifications given directly as a DFA or a PN, as it follows the idea of concurrent composition for PNs, as presented for example in [11].

5. Supervisor Admissibility

5.1. Checking Admissibility

In this subsection we provide a general procedure to check admissibility of the composition G_φ obtained from Algorithm 1. This procedure is based on the notion of *partially covering markings* [11, 33].

Definition 21 (Partially Covering Marking)**.** Given a set of places P, a marking $M \in \mathbb{N}^{|P|}$, and $P^= \subseteq P$, the set of partially covering markings is defined as:

$$\begin{aligned}&\mathcal{S}(M, P^=) = \\ &\{M' \in \mathbb{N}^{|P|} \mid M'(p) = M(p) \text{ for all } p \in P^= \text{ and} \\ &M'(p) \geq M(p) \text{ for all } p \in P \setminus P^=\}\end{aligned} \tag{16}$$

Partially covering markings represent the set of markings which are equal to M for places in $P^=$ and greater or equal than M for all other places. In spite of being a set of infinite cardinality, it is shown in [33] that checking the reachability of an element in this set is decidable.

Definition 22 (Modified Input Places)**.** Given $t \in T_{uc}$, let $T^t_\varphi = \{t_1, ..., t_r\}$ be the set of transitions of the form $t_{t,\psi}$ created by Algorithm 1 for $t \in T$. We define the set of combinations (Cartesian product) of input places that were modified for T^t_φ as:

$$P^t_\varphi = \prod_{i=1}^{r} \{p_\varphi \in P_\varphi \mid {}^\bullet t_i(p_\varphi) - {}^\bullet t(p_\varphi) > 0\} \tag{17}$$

For places $p_Q \in P_\varphi \setminus P$, i.e., places representing DFA states, we consider ${}^\bullet t(p_Q) = 0$.

We can now define the set of tokens for input places of t for which t would be enabled in G but none of the its corresponding transitions created in G_φ would be enabled.

Definition 23 (Inadmissible Token Distribution)**.** Let $\mathbf{p} = (p_1, ..., p_r) \in P^t_\varphi$. We define the set of inadmissible k's for $t \in T_{uc}$ as:

$$\begin{aligned}K^{t,\mathbf{p}} = \{&(k_1, ..., k_r) \in \mathbb{N}^r \mid \text{for all } i, j = 1, ..., r, \\ &{}^\bullet t(p_i) \leq k_i < {}^\bullet t_i(p_i) \text{ and if } p_i = p_j \text{ then } k_i = k_j\}\end{aligned} \tag{18}$$

Example 8. Figure 7 depicts (a) a system model G; and (b) a composition G_φ for a safe LTL formula φ. Note that places q_0 and q_1 represent DFA places. We exemplify how to construct some instances of the sets defined above: $T^{t_1}_\varphi = \{t_{t_1,\psi_1}, t_{t_1,\psi_2}\}$; $P^{t_1}_\varphi = \{p_1, q_1\} \times \{p_2, q_0\} = \{(p_1,p_2), (p_1,q_0), (q_1,p_2), (q_1,q_0)\}$; and $K^{t_1,(p_1,p_2)} = \{(1,0), (2,0), (1,1), (2,1)\}$.

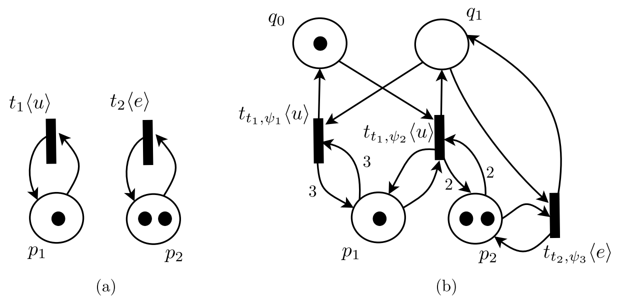
$t_1\langle u\rangle$
$t_2\langle e\rangle$
p_1
p_2
(a)
q_0
q_1
$t_{t_1,\psi_1}\langle u\rangle$
$t_{t_1,\psi_2}\langle u\rangle$
3
3
2
2
p_1
p_2
$t_{t_2,\psi_3}\langle e\rangle$
(b)

Figure 7: (a) A PN system model G. (b) A PN/DFA composition G_φ.

Note that, for the job allocation model in Figs. 2 and 3, and the self-loop of q_2 of the DFA A_{φ_1} in Figure 4, we have calculated one element of $T^{help\_robot}_{\varphi_1}$. However, this set has 4 elements, one for each transition of the DFA. Thus, exemplifying this construction for $T^{help\_robot}_{\varphi_1}$ is not possible, due to spacing constraints. This illustrates the first problem with this approach: one needs to take into account all possible combinations of k-tuples over a Cartesian product, which can become unmanageable very quickly.

The proposition below is a straightforward adaptation of the result in [33] to our composition algorithm.

Proposition 4. *Checking if G_φ is admissible for G can be reduced to checking reachability of a finite union of partially covering markings in G_φ.*

Proof. Let $t \in T_{uc}$, $\mathbf{p} = (p_1, ..., p_r) \in P^t_\varphi$, $P_\mathbf{p} = \{p_1\} \cup ... \cup \{p_r\}$ and $\mathbf{k} = (k_1, ..., k_r) \in K^{t,\mathbf{p}}$, and consider the marking following markings, defined over P_φ:

$$M_{t,\mathbf{p},\mathbf{k}}(p) = \begin{cases} k_i & \text{if } p = p_i \in P_\mathbf{p} \\ {}^\bullet t(p) & \text{if } p \in P \text{ and } p \notin P_\mathbf{p} \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

$$M_t(p) = \begin{cases} {}^\bullet t(p) & \text{if } p \in P \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

All possible uncontrollable markings for G_φ can be represented by the following:

$$\bigcup_{t \in T_{uc}} \bigcup_{\mathbf{p} \in P^t_\varphi} \bigcup_{\mathbf{k} \in K^{t,\mathbf{k}}} \mathcal{S}\left(M_{t,\mathbf{p},\mathbf{k}}, P_\mathbf{p}\right) \cup \bigcup_{t \in T_{uc} \mid T^t_\varphi = \emptyset} \mathcal{S}(M_t, \emptyset) \tag{21}$$

□

This result has the importance of showing that admissibility checking is decidable for G_φ. However, in practical terms, applying it is not efficient, as checking for reachability of a partially covering marking relies on building a PN G'_φ. The state space growth associated with the construction of G'_φ makes the procedure infeasible for PNs with state spaces larger than a few hundred states. In our experiences, we could only check a PN with less that 50 reachable markings, and G'_φ had more than 100,000 reachable markings.

5.2. Specification Language Restriction

Given the inefficiency of checking for admissibility, we present an extra syntactic restriction to safe LTL that guarantees supervisor admissibility a priori.

Definition 24 (Controllable SP-GEQ-Cs)**.** The set of controllable SP-GEQ-Cs for G is defined as:

$$\begin{aligned} spgecq_c = \{(p, \geq, b) \mid & \text{ for all } t \in T_{uc}, \\ & \text{if } t^\bullet(p) - {}^\bullet t(p) < 0 \text{ then } t^\bullet(p) \geq b\} \end{aligned} \tag{22}$$

Maintaining $(p, \geq, b) \in spgecq_c$ true between consecutive markings does not entail disabling uncontrollable transitions. Note that, given that our marking constraints are of the same form as GMECs, one can use known approaches for transforming uncontrollable SP-GEQ-Cs into controllable SP-GEQ-Cs, such as the approach in [41].

Example 9. In the PN of Figs. 2 and 3, $(idle\_robots, \geq, b)$ is controllable for all $b \in \{0, ..., k_R\}$, because ${}^\bullet t(idle\_robots) = 0$ for all $t \in T_{uc}$. Conversely, $(j_i\_in\_system, \geq, b)$ is uncontrollable for all $b \in \mathbb{N}$, because ${}^\bullet finished\_j_i(idle\_robots) = 1$, and ${}^\bullet finished\_j_i(idle\_robots) - finished\_j_i{}^\bullet(idle\_robots) = -1 < 0$, and $finished\_j_i$ is an uncontrollable transition.

Definition 25 (Syntactic Restriction)**.** We define the class of guaranteed admissible safe LTL formulas as a conjunction of formulas of the following types:

1. $G\psi$, where ψ is a propositional formula in the DNF where only SP-GEQ-Cs in $spgecq_c$ that are satisfied in the initial state, or literals $\neg e$ with $e \in E_c$ can occur.
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 SP-GEQ-Cs in $spgecq_c$ that also appear in γ, or literals $\neg e$ with $e \in E_c$ can occur.
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 SP-GEQ-Cs in $spgecq_c$ that also appear in γ, or literals $\neg e$ with $e \in E_c$ can occur.

Formulas of type 1 specify propositional logic relations between state descriptions and controllable events that should always be satisfied in all markings of the supervised PN.

Formulas of type 2 specify thatψ must be satisfied immediately after condition γ is met. Since γ is a condition, it can have occurrences of all atomic propositions. To ensure admissibility, only SP-GEQ-Cs in $spgecq_c$ that occur in γ, or the negation of controllable events can occur in ψ.

Formulas of type 3 specify that immediately after condition γ is satisfied, ψ must keep being satisfied until condition γ' is met. The restrictions for the SP-GEQ-Cs in ψ are the same as for formulas of type 2. Note that we can use formulas of type 3 to encode that an event $e \in E_c$ must occur as soon as possible:

$$\mathtt{G}(\gamma \Rightarrow X((\bigwedge_{e' \in E_c \setminus \{e\}} \neg e')W e)) \tag{23}$$

The restriction to formulas of the types above ensures admissibility of the supervisor because:

- We never enforce the occurrence of an event or set of events, we only enforce that a given subset of E_c must not happen at a given moment.
- We do not enforce that a certain place must remain with b or more tokens when the firing of an uncontrollable transition removes tokens from that place and does not put at least b tokens back on it.

Finally it is clear that if φ and ψ yields admissible supervisors, then $\varphi \wedge \psi$ must also yield an admissible supervisor.

6. Application to Multi-Robot Coordination

In this section, we illustrate and evaluate our approach on the running example scenario. We will use the PN represented in Figs. 2 and 3, and write a specifications according to Subsection 5.2 to control it.

First, formula φ_1, introduced in Ex. 3.

Second, if the number of broken and dead robots gets above a threshold k_{broken}, then we should prioritise the use of the available robots for repair jobs.

$$\begin{aligned} &\varphi_2 = \mathtt{G}((M(broken\_robots)+ \\ &M(dead\_robots) \geq k_{broken}) \Rightarrow \\ &(X(\bigwedge_{j_i \in J}(\neg assign\_new\_j_i \wedge \neg assign\_failed\_j_i)))) \end{aligned} \tag{24}$$

Third, jobs that have failed should be given priority over new jobs, as they have been in the system for more time.

$$\begin{aligned} \varphi_3 = &\mathtt{G}((\sum_{j_i \in J} M(failed\_j_i\_queue) \geq 1) \Rightarrow \\ &(X(\bigwedge_{j_i \in J} \neg assign\_new\_j_i))) \end{aligned} \tag{25}$$

The above rules are general, regardless of the number $|J|$ of types of job. One can also define prioritisation between different jobs. For example, between jobs 1 and 2, we might want to provide higher priority for job 1, such that we do not allow for allocation of jobs of type 2 if either there are less jobs of type 2 in the queue, or there are at least three jobs of type 1 queueing.

$$\begin{aligned} \varphi_4 = \mathtt{G}(((&M(new\_j_1\_queue) \geq M(new\_j_2\_queue)) \\ &\vee (M(new\_j_1\_queue) \geq 3)) \Rightarrow (X(\neg assign\_new\_j_2))) \end{aligned}$$

(26)

However, in the cases where there are more jobs of type 2 queueing, and there is at most one job of type 1 queueing, we do not allow for allocation of jobs of type 1. This allows the system to balance the distribution of jobs, and ensuring that when the load for jobs of type 1 is low, jobs of type 2 will be processed.

$$\varphi_5 = \mathtt{G}(((M(new\_j_1\_queue) < M(new\_j_2\_queue))\wedge (M(new\_j_1\_queue) \leq 1)) \Rightarrow (X(\neg assign\_new\_j_1))) \quad (27)$$

We note that the above rules involve both linear constraints and events, and are intuitively written from their natural language specification. This means that they cannot be directly stated as GMECs (which is based on place invariants that need to hold in *all* markings of the system, and here we are defining "conditional" rules that have to be enforced only in a subset of the markings of the system), nor as language specifications only using the set of events.

The overall specification is $\varphi = \varphi_1 \wedge ... \wedge \varphi_5$. In order to translate the LTL formulas into a DFA, we use the Spot ω-automaton library[2], which implements state-of-the-art translation methods. Given that the overall specification is a conjunction of five rules, one has two different options of building the supervisor PN:

1. Build A_φ for the conjunction of all the rules, and apply Algorithm 1 to the PN model and A_φ, building G_φ; or
2. For each rule φ_i, build A_{φ_i}, and incrementally apply Algorithm 1 to compose each A_{φ_i} with the PN model, building $G_{\varphi_1,...,\varphi_5}$.

Given that our composition represents the largest restriction of the PN behaviour that satisfies the specification, options 1) and 2) are equivalent in terms of the language of the resulting supervisor. We will investigate the pros and cons of these 2 approaches throughout this section. The implementation of our approach is available online[3]. To generate the reachability graph, we used the Tina toolbox[4], which provides tools for efficient generation of large state spaces.

The DFA for each φ_i has 2 states and 4 DNF transition labels, while the DFA for the conjunction φ has 10 states and 1000 DNF transition labels.

In Table 1, we report on the number of places and transitions of the PNs, along with the number of reachable markings, for the original model, and both the PN G_φ built using a conjunction of the specifications and the PN $G_{\varphi_1,...,\varphi_5}$ build incrementally. In the cases where the reachability graph could not be generated within 15 minutes[5] we write timeout in the corresponding table entry. These results show that we can compactly represent large state spaces using PNs. Furthermore, as the number of jobs increases, the size of the PN model increases linearly, and if the number of robots or number of specific jobs allowed in the system increase, the size of the PN remains constant. We now analyse the size of the supervisors. One can see that we are able to control systems with very large state spaces, and generate compact PN representations of their supervisors. This would not be possible using approaches based on state space enumeration.

For the models where we could generate the reachability graph, we can observe that a large percentage of the transitions generated by Algorithm 1 is dead, a discussed in 4.4.2. By removing these transitions identified as dead, we finish with a compact model representing the controlled system.

We also note the difference in terms of number of transitions in the final supervisor, for the 2 different supervisor construction approaches. The construction of the supervisors for the conjunction of the specifications mitigates the exponential growth of the transitions. Thus, when there are many specifications, the best approach is doing the conjunction of subsets of specifications such that each conjunction does not have more than 20 atomic propositions and can be translated in feasible time. Then, for those conjunctions, use the incremental approach to generate the final supervisor.

Finally, we note that for a fixed number of job types, increasing the number of robots or of allowed job of a given type does not increase the size of the supervisor structure, as it simply corresponds to adding more tokens to the model. Furthermore, note that we manage to scale up to a team of 32 robots, with a maximum of $|J| \times n_{j_i} = 6 \times 16 = 96$ tasks in the system. This is a substantially sized model: a state-based approach already has an order of one million states for 8 robots and 16 tasks, as shown in Table 1. For the 32 robot case, we could not generate the state space. However, note that 32 robots with 2 possible states already generates $2^3 2 > 10^9$ states. Our system is much more complex than that, including more possible states for robots and also task states. We also note that the computation of the supervisor is done only once before deployment, and it can then be run online very efficiently following the feedback loop described in Subsection 3.3.

7. Conclusions and Further Work

We described a method for multi-robot coordination based on the construction of PN supervisors that are guaranteed to fulfil state/event safe LTL specifications. We provided a novel semantics for LTL in PNs, where atomic propositions can be linear constraints on the markings of the system; showed how we can compose the PN with the DFA obtained from the safe LTL formula, obtaining a PN representation of a supervisor candidate; and discussed how admissibility can be checked, or guaranteed a priori by using a restriction to the specification language. This work develops a framework where both the multi-robot system and the specifications are represented in suitable and well-accepted formalisms, which allow for the automatic synthesis of supervisors from the specification of complex coordination rules, along with formal guarantees on the overall behaviour of the team.

Further work includes dealing with other relevant issues

[2] https://spot.lrde.epita.fr/

[3] https://github.com/bfalacerda/ltl_sup_con

[4] http://projects.laas.fr/tina/

[5] The experiments were performed on a laptop computer equipped with an Intel® Core™ i7 CPU @ 2.70GHz x 8, and 16GB of RAM.

n_r	$\lvert J\rvert$	n_{j_i}	$\lvert P\rvert$	$\lvert T\rvert$	$\lvert R(G)\rvert$	$\lvert P_\varphi\rvert$	$\lvert T_\varphi\rvert$	$\lvert T^{live}_\varphi\rvert$	$\lvert P_{\varphi_1,\dots,\varphi_5}\rvert$	$\lvert T_{\varphi_1,\dots,\varphi_5}\rvert$	$\lvert T^{live}_{\varphi_1,\dots,\varphi_5}\rvert$	$\lvert R(G_\varphi)\rvert$
4	2	4	35	17	11,316	45	2,540	414	45	24,192	978	10,998
6	2	4	35	17	31,162	45	2,540	475	45	24,192	1,162	45,010
8	2	4	35	17	66,932	45	2,540	475	45	24,192	1,162	111,160
8	2	6	35	17	295,536	45	2,540	475	45	24,192	1,162	441,512
8	2	8	35	17	886,664	45	2,540	475	45	24,192	timeout	1.05×10^6
8	3	8	43	22	timeout	53	3,872	timeout	53	30,848	timeout	timeout
8	6	8	67	37	timeout	77	6,536	timeout	77	50,816	timeout	timeout
32	6	16	67	37	timeout	77	6,536	timeout	77	50,816	timeout	timeout

Table 1
Model and supervisor sizes for different values of n_r (number of robots), $|J|$ (number of job classes), and n_{j_i} (maximum number of jobs of each class that can be in the system). $k_{broken} = n_r/2$.

in the the theory of SC, such as liveness or observability, and investigating more efficient approaches for admissibility checking such as [37]. Furthermore, we just described techniques for checking admissibility. We did not tackle the problem of finding *least restrictive* supervisors from the PN resulting of our composition. This is because in general PNs are not closed under this operation [4]. An interesting and relevant line of research is investigating subclasses of PNs for which one can verify admissibility more efficiently and build least restrictive supervisors for our composition. Furthermore, it is worth investigating how one can extend the language restriction to ensure admissibility, both looking at a less restrictive definition of controllable SP-GEQ-C, and to other types of safe LTL specifications where a priori admissibility can be guaranteed. Finally, we intend to implement our approach on real robots in order to further illustrate its suitability for multi-robot coordination, and extend it to also include optimisation of quantitative metrics of performance, along with the qualitative behaviour specification presented in this work.

Acknowledgements

This work was supported in part by the FCT ISR/LARSyS strategic funding UID/EEA/5009/2013, FCT grant FRH/BD/45046/2008, and by UK Research and Innovation and EPSRC through the Robotics and Artificial Intelligence for Nuclear (RAIN) research hub [EP/R026084/1].

References

[1] Abdulla, P.A., Iyer, S., Nylén, A., 2004. SAT-solving the coverability problem for Petri nets. Formal Methods in System Design 24.

[2] Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., 2011. Cardinality networks: a theoretical and empirical study. Constraints 16.

[3] Basile, F., Chiacchio, P., Giua, A., 2006. Suboptimal supervisory control of petri nets in presence of uncontrollable transitions via monitor places. Automatica 42.

[4] Cassandras, C.G., Lafortune, S., 2006. Introduction to Discrete Event Systems. Springer-Verlag New York, Inc.

[5] Chao, C., Thomaz, A., 2016. Timed Petri nets for fluent turn-taking over multimodal interaction resources in human-robot collaboration. The International Journal of Robotics Research 35, 1330–1353.

[6] Costelha, H., Lima, P., 2012. Robot task plan representation by Petri nets; Modelling, identification, analysis and execution. Autonomous Robots 33, 337–360. URL: https://doi.org/10.1007/s10514-012-9288-x, doi:10.1007/s10514-012-9288-x.

[7] Dias, M.B., Zlot, R., Kalra, N., Stentz, A., 2006. Market-based multi-robot coordination: A survey and analysis. Proceedings of the IEEE 94, 1257–1270.

[8] Ding, X.C., Smith, S.L., Belta, C., Rus, D., 2014. Optimal control of Markov decision processes with linear temporal logic constraints. IEEE Trans. on Automatic Control 59.

[9] Fainekos, G.E., Girard, A., Kress-Gazit, H., Pappas, G.J., 2009. Temporal logic motion planning for dynamic robots. Automatica 45, 343–352.

[10] Faruq, F., Lacerda, B., Hawes, N., Parker, D., 2018. Simultaneous task allocation and planning under uncertainty, in: Proc. of the 2018 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Madrid, Spain.

[11] Giua, A., 2013. Supervisory control of Petri nets with language specifications, in: Seatzu, C., Silva, M., van Schuppen, J.H. (Eds.), Control of Discrete-Event Systems – Lecture Notes in Control and Information Sciences. Springer. volume 433. chapter 12.

[12] Giua, A., DiCesare, F., 1991. Supervisory design using Petri nets, in: 30th IEEE Conf. on Decision and Control.

[13] Giua, A., DiCesare, F., 1994a. Blocking and controllability of Petri nets in supervisory control. IEEE Trans. on Automatic Control 39.

[14] Giua, A., DiCesare, F., 1994b. Petri net structural analysis for supervisory control. IEEE Trans. on Robotics and Automation 10.

[15] Giua, A., DiCesare, F., 1995. Decidability and closure properties of weak Petri net languages in supervisory control. IEEE Trans. on Automatic Control 40.

[16] Giua, A., DiCesare, F., Silva, M., 1992. Generalized mutual exclusion contraints on nets with uncontrollable transitions, in: IEEE Int. Conf. on Systems, Man and Cybernetics.

[17] Glocker, M., Reinl, C., Von Stryk, O., 2006. Optimal task allocation and dynamic trajectory planning for multi-vehicle systems using nonlinear hybrid optimal control. IFAC Proceedings Volumes 39, 38–43.

[18] Gromyko, A., Pistore, M., Traverso, P., 2006. A tool for controller synthesis via symbolic model checking, in: 8th Int. Workshop on Discrete Event Systems.

[19] Guo, M., Dimarogonas, D.V., 2015. Multi-agent plan reconfiguration under local ltl specifications. The International Journal of Robotics Research 34, 218–235.

[20] Iordache, M., Antsaklis, P., 2003. Design of T-liveness enforcing supervisors in Petri nets. IEEE Trans. on Automatic Control 48.

[21] Iordache, M., Antsaklis, P., 2006a. Supervision based on place invariants: A survey. Discrete Event Dynamic Systems 16.

[22] Iordache, M., Antsaklis, P., 2006b. Supervisory Control of Concurrent Systems: A Petri Net Structural Approach. Birkhäuser Boston.

[23] Iordache, M., Moody, J., Antsaklis, P., 2002. Synthesis of deadlock

prevention supervisors using Petri nets. IEEE Trans. on Robotics and Automation 18.

[24] Jiang, S., Kumar, R., 2006a. Diagnosis of repeated failures for discrete event systems with linear-time temporal-logic specifications. IEEE Trans. on Automation Science and Engineering 3.

[25] Jiang, S., Kumar, R., 2006b. Supervisory control of discrete event systems with CTL* temporal logic specifications. SIAM Journal on Control and Optimization 44.

[26] Kloetzer, M., Mahulea, C., 2014. A petri net based approach for multi-robot path planning. Discrete Event Dynamic Systems 24, 417–445.

[27] Kress-Gazit, H., Fainekos, G.E., Pappas, G.J., 2009. Temporal logic-based reactive mission and motion planning. IEEE Trans. on Robotics 25.

[28] Kupferman, O., Vardi, M., 2001. Model checking of safety properties. Formal Methods in System Design 19.

[29] Lacerda, B., 2013. Supervision of Discrete Event Systems Based on Temporal Logic Specifications. Ph.D. thesis. Instituto Superior Técnico.

[30] Lacerda, B., Faruq, F., Parker, D., Hawes, N., 2019. Probabilistic planning with formal performance guarantees for mobile service robots. International Journal of Robotics Research .

[31] Lacerda, B., Lima, P.U., 2009. LTL plan specification for robotic tasks modelled as finite state automata, in: Workshop ADAPT at AAMAS '09 - Agent Design: Advancing from Practice to Theory.

[32] Lacerda, B., Lima, P.U., 2011. Designing Petri net supervisors from LTL specifications, in: Robotics: Science and Systems VII.

[33] Lacerda, B., Lima, P.U., 2014. On the notion of uncontrollable marking in supervisory control of Petri nets. IEEE Trans. on Automatic Control 59.

[34] Lacerda, B., Parker, D., Hawes, N., 2017. Multi-objective policy generation for mobile robots under probabilistic time-bounded guarantees, in: Proc. of the 27th Int. Conf on Automated Planning and Scheduling (ICAPS), Pittsburgh, PA, USA.

[35] Lahijanian, M., Maly, M.R., Fried, D., Kavraki, L.E., Kress-Gazit, H., Vardi, M.Y., 2016. Iterative temporal planning in uncertain environments with partial satisfaction guarantees. IEEE Transactions on Robotics 32, 583–599.

[36] Lahijanian, M., Svorenová, M., Morye, A.A., Yeomans, B., Rao, D., Posner, I., Newman, P., Kress-Gazit, H., Kwiatkowska, M., 2018. Resource-performance tradeoff analysis for mobile robots. IEEE Robotics and Automation Letters 3, 1840–1847.

[37] Ma, Z., Li, Z., Giua, A., 2015. A method to verify the controllability of language specifications in Petri nets based on basis marking analysis, in: 54th IEEE Conf. on Decision and Control.

[38] Mahulea, C., Kloetzer, M., 2018. Robot planning based on boolean specifications using Petri net models. IEEE Transactions on Automatic Control 63, 2218–2225. doi:10.1109/TAC.2017.2760249.

[39] Mansouri, M., Lacerda, B., Hawes, N., Pecora, F., 2019. Multi-robot planning under uncertain travel times and safety constraints, in: Proc. of the 28th Int. Joint Conf. on Artificial Intelligence (IJCAI).

[40] Moody, J., Antsaklis, P., 1998. Supervisory control of discrete event systems using Petri nets. Springer.

[41] Moody, J., Antsaklis, P., 2000. Petri net supervisors for DES with uncontrollable and unobservable transitions. IEEE Trans. on Automatic Control 45.

[42] Nikou, A., Boskos, D., Tumova, J., Dimarogonas, D.V., 2018. On the timed temporal logic planning of coupled multi-agent systems. Automatica 97, 339–345.

[43] Nunes, E., Gini, M., 2015. Multi-robot auctions for allocation of tasks with temporal constraints, in: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI).

[44] Pnueli, A., 1981. The temporal semantics of concurrent programs. Theoretical Computer Science 13.

[45] Ramadge, P., Wonham, W., 1989. The control of discrete event systems. Proc. of the IEEE 77. doi:10.1109/5.21072.

[46] Reisig, W., Rozenberg, G., 1998. Lectures on Petri Nets Part I: Basic Models: Advances in Petri Nets. Springer.

[47] Schillinger, P., Bürger, M., Dimarogonas, D.V., 2018. Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. The international journal of robotics research 37, 818–838.

[48] Seow, K.T., 2007. Integrating temporal logic as a state-based specification language for discrete-event control design in finite automata. IEEE Trans. on Automation Science and Engineering 4.

[49] Sistla, A., 1994. Safety, liveness and fairness in temporal logic. Formal Aspects of Computing 6.

[50] Tsalatsanis, A., Yalcin, A., Valavanis, K.P., 2012. Dynamic task allocation in cooperative robot teams. Robotica 30, 721–730.

[51] Ulusoy, A., Smith, S.L., Ding, X.C., Belta, C., Rus, D., 2013. Optimality and robustness in multi-robot path planning with temporal logic constraints. The International Journal of Robotics Research 32, 889–911.

[52] Zavlanos, M.M., Pappas, G.J., 2008. Dynamic assignment in distributed motion planning with local coordination. IEEE Transactions on Robotics 24, 232–242.

[53] Ziparo, V.A., Iocchi, L., Lima, P.U., Nardi, D., 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.

Bruno Lacerda received his Ph.D. in Electrical and Computing Engineering from the Instituto Superior Técnico, University of Lisbon, Portugal, in 2013. Between 2013 and 2017, he was a Research Fellow at the School of Computer Science, University of Birmingham, UK. Currently, he is a Senior Researcher at the Oxford Robotics Institute, University of Oxford, UK. His research interests lie in the use of formal approaches to specify and synthesise high-level robot controllers. To achieve this goal, he his particularly interested in using temporal logics, Petri nets, supervisory control theory and planning under uncertainty.

Pedro U. Lima received his Ph.D. in Electrical Engineering from the Rensselaer Polytechnic Institute, Troy, NY, USA, in 1994. Currently, he is a Full Professor at Instituto Superior Técnico, Universidade de Lisboa, Portugal. He is also a researcher at the Institute for Systems and Robotics, where he is coordinator of the Intelligent Robots and Systems group. His research interests lie in the areas of discrete event systems and decision-making under uncertainty, namely their applications to multi-robot systems. Pedro Lima was a Trustee of the RoboCup Federation (2003-2011), and was the General Chair of RoboCup2004, held in Lisbon. He was the President of the Portuguese Robotics Society (2009-2011), and is a senior member of the IEEE. He is the co-author of two books. Pedro Lima was awarded a 6-month Chair of Excellence at the Universidad Carlos III de Madrid, Spain, by its Board of Governors. He has also been very active in the promotion of Science and Technology to the society, through the organization of Robotics events in Portugal, including the Portuguese Robotics Open since 2001.