Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

On-Line Synthesis of Permissive Supervisors for Partially Observed Discrete Event Systems under scLTL Constraints ⋆

Ami Sakakibara *,** **Toshimitsu Ushio** *

* *Graduate School of Engineering Science, Osaka University, 1-3 Machikaneyama, Toyonaka, Osaka, 560-8531, Japan (e-mail: sakakibara@hopf.sys.es.osaka-u.ac.jp; ushio@sys.es.osaka-u.ac.jp)*
** *JSPS Research Fellow*

Abstract: We consider a supervisory control problem of a discrete event system (DES) under partial observation, where a control specification is given by a fragment of linear temporal logic. We design an on-line supervisor that dynamically computes its control action with the complete information of the product automaton of the DES and an acceptor for the specification. The concepts of controllability and observability are defined by means of a ranking function defined on the product automaton, which decreases its value if an accepting state of the product automaton is being approached. The proposed on-line control scheme leverages the ranking function and a permissiveness function, which represents a time-varying permissiveness level. As a result, the on-line supervisor achieves the specification, being aware of the tradeoff between its permissiveness and acceptance of the specification, if the product automaton is controllable and observable.

Keywords: On-line supervisory control, discrete event systems, partial observation, linear temporal logic, ranking function, automata.

1. INTRODUCTION

The supervisory control theory for discrete event systems (DESs) has been widely studied since its initiation in Ramadge and Wonham (1987). Supervisor synthesis turns out to be computationally hard when the controlled system is large or has much complex aspects. Researchers have overcome this difficulty by designing supervisors on-line. Chung et al. (1992, 1993, 1994) proposed a method to generate limited lookahead trees on-the-fly instead of constructing a complete supervisor; their methods have been extended to the settings of partial observation (Hadj-Alouane et al., 1996) or time-varying DESs (Grigorov and Rudie, 2006). Another extension is to design on-line supervisors for partially observed DESs, where the supervisor modifies appropriate control actions precomputed in the case of full observation (Heymann and Lin, 1994; Prosser et al., 1998).

In the supervisory control framework, control requirements are typically given by formal languages, i.e., subsets of event sequences generated by the system. Practically, we need to translate desired behavior of the system into formal languages, which is a hard task. For this reason, linear temporal logic (LTL) is paid much attention to as a formal specification language for control problems, thanks to its rich expressiveness (Belta et al., 2017; Tumova and Dimarogonas, 2016; Jiang and Kumar, 2006). It is practically acceptable to restrict the specification language to a fragment of LTL like syntactically co-safe LTL (scLTL), for which synthesis problems can be solved in much less complexity than for the case of the general LTL (Kupferman and Y. Vardi, 2001).

In Sakakibara and Ushio (2020), we consider a supervisory control problem of a DES under an scLTL constraint. We propose an on-line control scheme, where we leverage a *ranking function* that enables us to find desirable behavior with respect to the scLTL specification. The concept of ranking functions is similar with that of Lyapunov functions, which play a great role in determining control strategies. The key idea is that, if the rank decreases along a trajectory, we regard it as good. Ranking functions are useful in solving games played on a graph and reachability analysis of automata, which sometimes give solutions to LTL-related problems. We define a ranking function on the product automaton of the DES and the specification automaton so that its value decreases if an accepting state of the product automaton is being approached. To implement the tradeoff between supervisor permissiveness and specification acceptance, we additionally introduce a *permissiveness function* that provides a time-varying permissiveness level. If we have a higher permissive level, the supervisor may enable events that do not necessarily lead to the achievement of the specification. By referring to the permissiveness level together with the ranking function at each step of the on-line control process, the supervisor computes more permissive control patterns.

This paper extends the on-line supervisory control scheme proposed in Sakakibara and Ushio (2020) to the setting of partial observation. After each observation, the supervisor

⋆ This work was supported by JST ERATO Grant Number JPMJER1603, Japan, and JSPS KAKENHI Grant Number JP19J13487, Japan.

dynamically computes its control action with the information of the fully observed product automaton, on which the ranking function is defined. Furthermore, we characterize the concepts of controllability and observability by means of the ranking function. The supervisor forces the DES to satisfy the scLTL specification if the product automaton is controllable and observable.

The rest of this paper is organized as follows. Section 2 gives fundamental definitions and notations. Section 3 formulates a supervisory control problem for scLTL specifications. Section 4 briefly explains the ranking function and its related properties. Section 5 proposes our on-line control scheme. Section 6 demonstrates the scheme with a simple example. Finally, Section 7 concludes the paper.

2. PRELIMINARIES

For a set T, we denote by $|T|$ its cardinality. T^* (resp., T^ω) represents a set of finite (resp., infinite) sequences over T. For a finite or infinite sequence τ over T, let $\tau[j]$ be the $(j+1)$-st element of τ. For any $j, k \in \mathbb{N}$ with $j \leq k$, we denote by $\tau[j \dots k]$ the sequence $\tau[j]\tau[j+1]\dots\tau[k]$. We write $\tau' \preceq \tau$ if τ' is a prefix of τ. For a finite sequence $\tau \in T^*$, $\|\tau\|$ stands for the length of τ.

2.1 Discrete Event Systems

A discrete event system (DES) is a tuple
$$G = ((X, \Sigma, \delta, x_0), AP, L),$$
where X is the set of states, Σ is the set of events, a partial function $\delta : X \times \Sigma \to X$ is the transition function, $x_0 \in X$ is the initial state, AP is the set of atomic propositions, and $L : X \to 2^{AP}$ is the labeling function. G is said to be finite if X, Σ, and AP are all finite. We write $\delta(x, \sigma)!$ if a transition from x with σ is defined. For each $x \in X$, let $\Sigma(x) = \{\sigma \in \Sigma : \delta(x, \sigma)!\}$. We denote by a triple (x, σ, x') a transition from $x \in X$ to $x' = \delta(x, \sigma)$ for some $\sigma \in \Sigma(x)$. Moreover, the transition function is extended to a sequence of inputs: for $\sigma \in \Sigma$ and $s \in \Sigma^*$, $\delta(x, \varepsilon) = x$ and $\delta(x, s\sigma) = \delta(\delta(x, s), \sigma)$.

Let $\mathcal{L}(G) = \{s \in \Sigma^* : \delta(x_0, s)!\}$ be the set of all finite event sequences generated by G. An infinite sequence $\rho \in X(\Sigma X)^\omega$ is called a *run* if, for any $j \in \mathbb{N}$, $\rho[2(j+1)] \in \delta(\rho[2j], \rho[2j+1])$. A finite sequence $h \in X(\Sigma X)^*$ is called a *history* if $h \in X$ or, for h with $\|h\| \geq 3$, $h[2(j+1)] \in \delta(h[2j], h[2j+1])$ for any $j \in \{0, 1, \dots, \frac{\|h\|-3}{2}\}$. The set of runs (resp., histories) starting from the initial state x_0 is defined as $\mathsf{Runs}(G)$ (resp., $\mathsf{His}(G)$). DES G is said to be *deadlock-free* if, for any $s \in \mathcal{L}(G)$, there exists $\sigma \in \Sigma$ such that $\delta(x_0, s\sigma)!$.

The event set is partitioned into disjoint subsets $\Sigma = \Sigma_c \cup \Sigma_{uc}$, where Σ_c (resp., Σ_{uc}) is the set of controllable (resp., uncontrollable) events. We define $\Sigma_c(x) = \Sigma(x) \cap \Sigma_c$ and $\Sigma_{uc}(x) = \Sigma(x) \cap \Sigma_{uc}$ for each $x \in X$. We have another partition of the event set $\Sigma = \Sigma_o \cup \Sigma_{uo}$ with the set Σ_o (resp., Σ_{uo}) of observable (resp., unobservable) events. Let $\mathcal{P} : \Sigma^* \to \Sigma_o^*$ be a natural projection defined inductively as follows:
$$\mathcal{P}(\varepsilon) = \varepsilon,$$
$$\forall s \in \Sigma^*, \forall \sigma \in \Sigma,\ \mathcal{P}(s\sigma) = \begin{cases} \mathcal{P}(s)\sigma & \text{if } \sigma \in \Sigma_o, \\ \mathcal{P}(s) & \text{if } \sigma \in \Sigma_{uo}. \end{cases}$$

We also define the inverse $\mathcal{P}^{-1} : 2^{\Sigma_o^*} \to 2^{\Sigma^*}$ as $\mathcal{P}^{-1}(T) = \{s \in \Sigma^* : t \in T, \mathcal{P}(s) = t\}$. The observable behavior of the DES G is given by $\mathcal{P}(\mathcal{L}(G)) = \{s = \mathcal{P}(t) \in \Sigma_o^* : t \in \mathcal{L}(G)\}$.

For the set AP, a *letter* $\nu \in 2^{AP}$ stands for a subset of atomic propositions. A *word* is a finite or infinite string of letters. Each run in $\mathsf{Runs}(G)$ generates a sequence of letters, namely a word over 2^{AP}, obtained by the labeling function. We extend the labeling function to express such words: for run ρ, $L(\rho) = L(\rho[0])L(\rho[2])\dots$. The extension for histories is defined similarly.

2.2 Syntactically Co-Safe Linear Temporal Logic

Linear temporal logic (LTL) is useful to describe qualitative control specifications. In this paper, we focus on syntactically co-safe LTL (scLTL), a subclass of LTL. Formally, an scLTL formula φ over the set AP of atomic propositions is defined as
$$\varphi ::= \text{true} \mid a \mid \neg a \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \mathbf{U} \varphi_2,$$
where $a \in AP$, $\varphi, \varphi_1, \varphi_2$ are scLTL formulas. In addition, we usually use a temporal operator $\Diamond$, which is defined by $\Diamond\varphi := \text{true}\mathbf{U}\varphi$.

The semantics of scLTL is defined over an infinite word (Baier and Katoen, 2008). Intuitively, $\bigcirc\varphi$ is true if φ holds on the word from the *next* step and $\varphi_1\mathbf{U}\varphi_2$ if φ_1 keeps to be satisfied *until* φ_2 turns true at some step. For an scLTL formula φ over AP and an infinite word $w \in (2^{AP})^\omega$, we write $w \models \varphi$ if w satisfies φ. For a DES G and an scLTL formula φ, we say G satisfies φ, denoted by $G \models \varphi$, if $L(\rho) \models \varphi$ for all $\rho \in \mathsf{Runs}(G)$.

Although scLTL formulas are evaluated over infinite words, it is known that we only need to check whether an input word has a *good prefix* of the formula. Any scLTL formula can be translated into a corresponding deterministic finite automaton (DFA), which is an acceptor for the good prefixes (Kupferman and Y. Vardi, 2001). For an scLTL formula φ, let $A_\varphi = ((X_A, \Sigma_A, \delta_A, x_{A,0}), F_A)$ be its corresponding DFA, where X_A is the set of states, $\Sigma_A = 2^{AP}$ is the input alphabet, a total function $\delta_A : X_A \times \Sigma_A \to X_A$ is the transition function, $x_{A,0} \in X_A$ is the initial state, and $F_A \subseteq X_A$ is the set of accepting states. For any $w \in (2^{AP})^\omega$, we have
$$w \models \varphi \iff \exists w' \in (2^{AP})^*, w' \preceq w \wedge \delta_A(x_{A,0}, w') \in F_A. \tag{1}$$

3. FORMULATION

In this paper, we formulate a supervisory control problem with scLTL specifications. A controller, called a *supervisor*, enables some controllable events at each state (Ramadge and Wonham, 1987). For each state $x \in X$, we define $\Gamma(x) = \{\gamma \subseteq \Sigma(x) : \Sigma_{uc}(x) \subseteq \gamma\}$, where $\gamma \in \Gamma(x)$ is called a *control pattern* at x. Let $\Gamma = \bigcup_{x \in X} \Gamma(x)$ be the set of all control patterns. The supervisor determines a control pattern after each observation. Formally, we define a *supervisor under partial observation* $\mathcal{P}$ as a mapping $\mathcal{S} : \mathcal{P}(\mathcal{L}(G)) \to \Gamma$.

Definition 1. (Supervised behavior). Let $\mathcal{S}$ be a supervisor under partial observation $\mathcal{P}$ for a DES $G =$

$((X, \Sigma, \delta, x_0), AP, L)$. The closed-loop behavior of the DES G under the control by $\mathcal{S}$, denoted by $\mathcal{S}/G$, is given by

$$\begin{aligned} &\varepsilon \in \mathcal{L}(\mathcal{S}/G), \text{ and} \\ &\forall s \in \mathcal{L}(\mathcal{S}/G), s\sigma \in \mathcal{L}(\mathcal{S}/G) \\ &\qquad\qquad \iff s\sigma \in \mathcal{L}(G) \wedge \sigma \in \mathcal{S}(\mathcal{P}(s)). \\ &\mathsf{Runs}(\mathcal{S}/G) = \{x_0\sigma_1x_1 \ldots \in \mathsf{Runs}(G) : \sigma_1 \in \mathcal{S}(\varepsilon) \\ &\qquad\qquad \wedge \forall j \geq 1, \sigma_{j+1} \in \mathcal{S}(\mathcal{P}(\sigma_1\sigma_2 \ldots \sigma_j))\}. \end{aligned}$$

Problem 2. Given a finite deadlock-free DES $G = ((X, \Sigma, \delta, x_0), AP, L)$ and an scLTL formula φ over AP, synthesize a supervisor $\mathcal{S}$ under partial observation $\mathcal{P}$ such that $\mathcal{S}/G \models \varphi$.

To solve Problem 2, we design an on-line supervisor, which dynamically computes a control pattern after observing an event occurrence. Our control scheme is divided into two stages; we first execute the preprocessing off-line, and then move on to the on-line control stage, where it stops controlling the DES after detecting a history corresponding to a good prefix of the scLTL specification.

4. RANKING FUNCTION

The specification given by an scLTL formula φ is translated into the equivalent DFA A_φ. Then, we obtain the product automaton P of the DES G and the DFA A_φ, which is computed as follows.

$$P = G \otimes A_\varphi = ((X_P, \Sigma_P, \delta_P, x_{P,0}), F_P),$$

where $X_P = X \times X_A$ is the set of states, $\Sigma_P = \Sigma$ is the set of events, $\delta_P : X_P \times \Sigma_P \to X_P$ is the transition function, $x_{P,0} = (x_0, \delta_A(x_{A,0}, L(x_0)))$ is the initial state, and $F_P = X \times F_A$ is the set of accepting states. For each $x = (x_G, x_A) \in X_P$ and $\sigma \in \Sigma_P$, $\delta_P(x,\sigma) = \big(\delta(x_G, \sigma), \delta_A(x_A, L(\delta(x_G, \sigma)))\big)$ and we define $J_G(x) = x_G$ and $J_A(x) = x_A$. The two kinds of event partitions are inherited from the DES: $\Sigma_{P,c} = \Sigma_c$, $\Sigma_{P,uc} = \Sigma_{uc}$, $\Sigma_{P,o} = \Sigma_o$, and $\Sigma_{P,uo} = \Sigma_{uo}$. Note that $\mathcal{L}(P) = \mathcal{L}(G)$ and $\mathcal{P}(\mathcal{L}(P)) = \mathcal{P}(\mathcal{L}(G))$.

Since the product automaton captures the behavior of the DES and the DFA at the same time, our goal turns out to reach an accepting state of the product automaton. For that purpose, we introduce a ranking function with the existence of uncontrollable transitions (Sakakibara and Ushio, 2020), which decreases its value if an accepting state is being approached.

Definition 3. Let $P = ((X_P, \Sigma_P, \delta_P, x_{P,0}), F_P)$ be a product automaton. A function $\xi : X_P \to \mathbb{N}$ is a *ranking function* for P if

$$\begin{aligned} &\forall x \in X_P, \forall \sigma \in \Sigma_{P,uc}(x), \\ &\qquad \xi(x) \geq \min\left\{\xi(\delta_P(x, \sigma)) + \bar{I}_{F_P}(x), \alpha\right\}, \end{aligned}$$

where $\alpha > |X_P| - |F_P|$ and $\bar{I}_{F_P} : X_P \to \{0, 1\}$ is an indicator function such that $\bar{I}_{F_P}(x) = 1$ if and only if $x \notin F_P$.

In Sakakibara and Ushio (2020), we propose an algorithm to compute a ranking function for the product automaton. Here, we show important results related to the ranking function ξ with the upper bound $\alpha = |X_P| - |F_P| + 1$.

Proposition 4. For any $x \in X_P$, $x \in F_P$ if and only if $\xi(x) = 0$.

Proposition 5. For any $x \in X_P$,

$$\xi(x) < \alpha \implies \exists s \in \Sigma_P^*, \delta_P(x, s) \in F_P.$$

Proposition 6. For any $x \in X_P$,

$$\begin{aligned} &0 < \xi(x) < \alpha \\ &\qquad \implies \{\sigma \in \Sigma_P(x) : \xi(x) > \xi(\delta_P(x, \sigma))\} \neq \emptyset. \end{aligned}$$

Proposition 7. For any $x \in X_P \setminus F_P$,

$$\Sigma_{P,uc}(x) \subseteq \{\sigma \in \Sigma_P(x) : \xi(x) > \xi(\delta_P(x, \sigma))\}.$$

Proposition 6 ensures that a lower-ranked successor always exists. Moreover, as mentioned in Proposition 7, each successor associated with an uncontrollable event has a lower rank than that of the current state.

In general, the existence of supervisors under partial observation depends on the controllability and observability of the specification language. Although these properties are defined by means of languages in the conventional supervisory control theory (Cassandras and Lafortune, 2008), here we characterize them with the ranking function.

Definition 8. The product automaton P is said to be *controllable* (with respect to ξ) if $\xi(x_{P,0}) < \alpha$.

Definition 9. The product automaton P is said to be *observable* (with respect to ξ and $\mathcal{P}$) if

$$\begin{aligned} &\forall s, s' \in \mathcal{L}(P),\ \forall \sigma \in \Sigma_P, \\ &\qquad \mathcal{P}(s) = \mathcal{P}(s') \wedge \xi(\delta_P(x_{P,0}, s)) > \xi(\delta_P(x_{P,0}, s\sigma)) \\ &\qquad\qquad \wedge s'\sigma \in \mathcal{L}(P) \\ &\qquad\qquad \implies \xi(\delta_P(x_{P,0}, s')) > \xi(\delta_P(x_{P,0}, s'\sigma)). \end{aligned}$$

If the context is clear, we simply say P is controllable or observable without referring to ξ and $\mathcal{P}$. The observability condition requires that, if an event is defined after different sequences with the same observation, then all of the transitions triggered by the event agree with each other in a sense of whether the product automaton gets closer to accepting states or not.

We characterize transitions of the product automaton with respect to the ranking function. Let $(x, \sigma, x') \in X_P \times \Sigma_P \times X_P$ be a transition defined in the product automaton.

- (x, σ, x') is *legal* (with respect to ξ) if $\xi(x) > \xi(x')$.
- (x, σ, x') is *neutral* (with respect to ξ) if $\xi(x) \leq \xi(x') < \alpha$.
- (x, σ, x') is *illegal* (with respect to ξ) if $\xi(x') = \alpha$.

It is possible to lead the product automaton to reach an accepting state if we always choose legal transitions. On the other hand, however, we are likely to obtain more *permissive* supervisors if we allow not only legal transitions but also neutral ones to be enabled. Permissiveness is one of the most important concepts in the supervisory control theory, where we often aim to design a supervisor that enables as many events as possible Cassandras and Lafortune (2008).

5. ON-LINE SUPERVISORY CONTROL UNDER PARTIAL OBSERVATION

In this section, we explain the on-line supervisory control scheme for Problem 2, given the product automaton P and the ranking function ξ. Notice that we have a tradeoff between permissiveness of the supervisor and achievement of the specification. The supervisor becomes more permissive if it enables events triggering neutral transitions. However, infinitely many occurrences of neutral transitions results in livelock, i.e., the product automaton may stay within states x with $0 < \xi(x) < \alpha$ while it always holds the possibility of reaching an accepting state but actually suspends going there.

To take the tradeoff into consideration, we introduce a criterion for how many neutral transitions we allow to be enabled. More precisely, the supervisor we design determines its control action on-line, being aware of a time-varying permissiveness level, which is referred to together with the ranking function to improve permissiveness of supervisors. Here, we introduce a function that quantifies the permissiveness level.

Definition 10. A *permissiveness function* is a function $\eta : \mathbb{N} \to \mathbb{R}$ that satisfies the following three conditions.

(1) $\eta(0) \leq \alpha$;
(2) $\eta(k) \geq \eta(k+1)$ for any $k \in \mathbb{N}$;
(3) $\eta(\bar{k}) = 0$ for some $\bar{k} \in \mathbb{N}$.

That is, the permissiveness level decreases as time goes by and will eventually be exhausted.

With partially observed information, the supervisor cannot know which state the product automaton is currently in. We define the unobservable reach (Hadj-Alouane et al., 1996) from a state $x \in X_P$ under an event subset $\Sigma' \subseteq \Sigma_P$ as follows.

$$\mathsf{UR}_{\Sigma'}(x) = \{x' \in X_P : u \in (\Sigma_{uo} \cap \Sigma')^*, x' = \delta_P(x, u)\}.$$

For a subset $X' \subseteq X_P$, let $\mathsf{UR}_{\Sigma'}(X') = \bigcup_{x \in X'} \mathsf{UR}_{\Sigma'}(x)$. The set of *next states* after each observation is defined recursively as follows: for $s_o \in \mathcal{P}(\mathcal{L}(P))$ and $\sigma_o \in \Sigma_o$,

$$\mathsf{NS}(\varepsilon) = \{x_{P,0}\},$$
$$\mathsf{NS}(s_o\sigma_o) = \{x' = \delta_P(x, \sigma_o) \in X_P : x \in \mathsf{UR}_{\Sigma_P}(\mathsf{NS}(s_o))\}.$$

The key idea of our on-line control scheme is made up of the following two rules. First, we always enable events that trigger legal transitions no matter which state the DES is in or how much the permissiveness level currently is. Second, it is possible to allow neutral transitions if we have enough permissiveness level. These concepts are realized by two different control actions given by, for each $x \in X_P$ and $k \in \mathbb{N}$,

$$\begin{aligned}\hat{\gamma}_{leg}(x) =& \{\sigma \in \Sigma_P : \exists s \in \Sigma_{uo}^*, \delta_P(x, s)! \wedge \delta_P(x, s\sigma)! \\ &\wedge \xi(\delta_P(x, s)) > \xi(\delta_P(x, s\sigma))\}, \\ \hat{\gamma}_{per}(x, k) =& \{\sigma \in \Sigma_P : \forall s \in \Sigma_{uo}^*, \\ &\delta_P(x, s\sigma)! \implies \xi(\delta_P(x, s\sigma)) < \eta(k)\}.\end{aligned}$$

Applying $\hat{\gamma}_{leg}(x)$ results in occurrences of only legal transitions within $\mathsf{UR}_{\Sigma_P}(x)$ while the supervisor enables events in $\hat{\gamma}_{per}(x)$, which potentially trigger neutral transitions, if the permissiveness level is high. The supervisor refers to the permissiveness function to determine whether such additional events are currently acceptable. From Proposition 6, we have the following proposition.

Proposition 11. If the product automaton is controllable, then for any $x \in X_P \setminus F_P$, we have $\hat{\gamma}_{leg}(x) \neq \emptyset$.

Proposition 12. Assume that the product automaton is controllable and observable. Then, for any $x \in X_P \setminus F_P$, we have

$$\begin{aligned}&\forall \sigma \in \hat{\gamma}_{leg}(x), \forall s \in \Sigma_{uo}^*, \\ &\quad \delta_P(x, s)! \wedge \delta_P(x, s\sigma)! \implies \xi(\delta_P(x, s)) > \xi(\delta_P(x, s\sigma)).\end{aligned}$$

Proof. Let $x \in X_P \setminus F_P$. From Proposition 11, then, $\hat{\gamma}_{leg}(x) \neq \emptyset$. Suppose that, for some $\sigma \in \hat{\gamma}_{leg}(x)$, we have

$$\begin{aligned}\exists s \in \Sigma_{uo}^*, &\delta_P(x, s)! \wedge \delta_P(x, s\sigma)! \\ &\wedge \xi(\delta_P(x, s)) \leq \xi(\delta_P(x, s\sigma)).\end{aligned} \tag{2}$$

From the definition of $\hat{\gamma}_{leg}(x)$, the event σ satisfies

$$\begin{aligned}\exists s' \in \Sigma_{uo}^*, &\delta_P(x, s')! \wedge \delta_P(x, s'\sigma)! \\ &\wedge \xi(\delta_P(x, s)) > \xi(\delta_P(x, s\sigma)).\end{aligned} \tag{3}$$

Let $t_x \in \mathcal{L}(P)$ such that $\delta_P(x_{P,0}, t_x)$. Then, we have $\mathcal{P}(\mathcal{L}(t_x s)) = \mathcal{P}(\mathcal{L}(t_x s'))$ but for the event σ both Eqs. (2) and (3) hold, which contradicts the assumption that the product automaton is observable.

In the on-line control scheme, the supervisor keeps the set $\mathsf{NS}(s_o)$ of states where the product automaton is estimated to be based on the observation $s_o \in \mathcal{P}(\mathcal{L}(P))$. After observing an observable event $\sigma_o \in \Sigma_o$, the supervisor updates the set to $\mathsf{NS}(s_o\sigma_o)$ and computes a control pattern to be applied for the control of $\mathsf{UR}_\Sigma(\mathsf{NS}(s_o\sigma_o))$. Then, based on the idea mentioned above, the supervisor $\hat{\mathcal{S}}$ computes a control pattern satisfying, for each $s_o \in \mathcal{P}(\mathcal{L}(P))$,

$$\hat{\mathcal{S}}(s_o) = \hat{\gamma}_{leg}(\mathsf{NS}(s_o)) \cup \hat{\gamma}_{per}(\mathsf{NS}(s_o), \|s_o\|), \tag{4}$$

where

$$\hat{\gamma}_{leg}(\mathsf{NS}(s_o)) = \bigcup_{x \in \mathsf{NS}(s_o)} \hat{\gamma}_{leg}(x),$$
$$\hat{\gamma}_{per}(\mathsf{NS}(s_o)) = \bigcap_{x \in \mathsf{NS}(s_o)} \hat{\gamma}_{per}(x).$$

From Proposition 11, we have the following proposition.

Proposition 13. If the product automaton is controllable, then for any $s_o \in \mathcal{P}(\mathcal{L}(P))$, $\hat{\gamma}_{leg}(\mathsf{NS}(s_o)) \neq \emptyset$.

Proposition 14. Assume that the product automaton is controllable and observable. Then, for any $s_o \in \mathcal{P}(\mathcal{L}(P))$ and any $x \in \mathsf{NS}(s_o)$,

$$\forall x' \in \mathsf{UR}_{\Sigma_P}(x), \forall \sigma \in \hat{\gamma}_{leg}(\mathsf{NS}(s_o)) \setminus \hat{\gamma}_{leg}(x), \neg\delta_P(x', \sigma)!. \tag{5}$$

Proof. We prove the proposition by contradiction. Let $s_o \in \mathcal{P}(\mathcal{L}(P))$. By the controllability of P and Proposition 13, we have $\hat{\gamma}_{leg}(\mathsf{NS}(s_o)) \neq \emptyset$. Suppose that, for some $x' \in \mathsf{UR}_{\Sigma_P}(x)$, there exists $\sigma \in \hat{\gamma}_{leg}(\mathsf{NS}(s_o)) \setminus \hat{\gamma}_{leg}(x)$ with $\delta_P(x', \sigma)!$. This means that for some $y \in \mathsf{NS}(s_o)$ with $y \neq x$, we have $\sigma \in \hat{\gamma}_{leg}(y)$ and $\delta_P(x', \sigma)!$. Equivalently, both of the following conditions hold for some σ:

1) $\exists t_x \in \mathcal{P}^{-1}(s_o), x = \delta_P(x_{P,0}, t_x) \wedge \exists u \in \Sigma_{uo}^*, \delta_P(x, u\sigma)!$.
2) $\exists t_y \in \mathcal{P}^{-1}(s_o), y = \delta_P(x_{P,0}, t_y) \wedge \exists s_y \in \Sigma_{uo}^*, \delta_P(y, s_y)! \wedge \delta_P(y, s_y\sigma)! \wedge \xi(\delta_P(y, s_y)) > \xi(\delta_P(y, s_y\sigma))$.

In the above conditions, we have $\mathcal{P}(t_x u) = \mathcal{P}(t_x)\varepsilon = s_o$ and $\mathcal{P}(t_y y_s) = \mathcal{P}(t_y)\varepsilon = s_o$. To sum up, we have

$$\begin{aligned}\exists t'_x, t'_y \in \mathcal{L}(P),\ &\mathcal{P}(t'_x) = \mathcal{P}(t'_y) \wedge t'_y\sigma \in \mathcal{L}(P) \\ &\wedge \xi(\delta_P(y, t'_y)) > \xi(\delta_P(y, t'_y\sigma)) \wedge t'_x\sigma \in \mathcal{L}(P).\end{aligned}$$

On the other hand, however, it holds that $\xi(\delta_P(x, t'_x)) \leq \xi(\delta_P(x, t'_x\sigma))$ because $\sigma \notin \hat{\gamma}_{leg}(x)$. We now have the contradiction to the observability condition of the product automaton.

Algorithm 1 An on-line supervisory control algorithm for a DES G under an scLTL constraint φ, given the product automaton $P = ((X_P, \Sigma_P, \delta_P, x_{P,0}), F_P)$, a ranking function $\xi : X_P \to \mathbb{N}$, and a permissiveness function η

if P is controllable and observable **then**
  $NS \leftarrow \{x_{P,0}\}$, $k \leftarrow 0$
  while $NS \not\subseteq F_P$ **do**
    $\gamma_k \leftarrow \emptyset, \bar{\gamma}_k \leftarrow \emptyset, UR \leftarrow \emptyset, NS' \leftarrow \emptyset$
    for all $x \in NS$ **do**
      LegExpand$(x, \gamma_k, \bar{\gamma}_k, UR, NS')$
    for all $\sigma \in \Sigma_{P,c} \setminus (\gamma_k \cup \bar{\gamma}_k)$ **do**
      $UR_\sigma \leftarrow \emptyset, NS'_\sigma \leftarrow \emptyset,$ st $\leftarrow$ false
      for all $x \in NS$ **do**
        ReExpand$(x, \gamma_k \cup \{\sigma\}, UR_\sigma, NS'_\sigma, \eta(k),$ st$)$
        if st **then**
          break
      if $\neg$st **then**
        $\gamma_k \leftarrow \gamma_k \cup \{\sigma\}$
        $UR \leftarrow UR \cup UR_\sigma$
        $NS' \leftarrow NS' \cup NS'_\sigma$
    $\sigma_o \leftarrow \mathsf{observe}(G, \gamma_k)$
    $NS \leftarrow \{x' \in NS' : x \in UR, x' = \delta_P(x, \sigma_o)\}$
    $k \leftarrow k + 1$
else
  There is no supervisor for G that guarantees φ under partial observation.

function LegExpand$(x \in X_P, \gamma, \bar{\gamma}, UR, NS')$
  if $x \notin UR$ **then**
    $UR \leftarrow UR \cup \{x\}$
    for all $\sigma \in \Sigma_P(x)$ **do**
      if $\xi(x) > \xi(\delta_P(x, \sigma))$ **then**
        $\gamma \leftarrow \gamma \cup \{\sigma\}$
        if $\sigma \in \Sigma_{uo}$ **then**
          LegExpand$(\delta_P(x, \sigma), \gamma, \bar{\gamma}, UR, NS')$
        else
          $NS' \leftarrow NS' \cup \{\delta_P(x, \sigma)\}$
      else if $\xi(\delta_P(x, \sigma)) = \alpha$ **then**
        $\bar{\gamma} \leftarrow \bar{\gamma} \cup \{\sigma\}$

function ReExpand$(x, \gamma, UR, NS', \eta_k,$ st$)$
  if $x \notin UR \wedge \neg$st **then**
    $UR \leftarrow UR \cup \{x\}$
    for all $\sigma \in \gamma \cap \Sigma_P(x)$ **do**
      if $\xi(\delta_P(x, \sigma)) < \eta_k$ **then**
        if $\sigma \in \Sigma_{uo}$ **then**
          ReExpand$(\delta_P(x, \sigma), \gamma, UR, NS', \eta_k,$ st$)$
        else
          $NS' \leftarrow NS' \cup \{\delta_P(x, \sigma)\}$
      else
        st $\leftarrow$ true

From Proposition 14, we have the following lemma.

Lemma 15. If the product automaton is controllable and observable, we have, for any $s_o \in \mathcal{P}(\mathcal{L}(P))$,

$$\forall x \in \mathsf{NS}(s_o),\ \mathsf{UR}_{\hat{\gamma}_{leg}(\mathsf{NS}(s_o))}(x) = \mathsf{UR}_{\hat{\gamma}_{leg}(x)}(x).$$

Corollary 16. If the product automaton is observable, then for any $s_o \in \mathcal{P}(\mathcal{L}(P))$,

$$\forall x \in \mathsf{NS}(s_o), \max_{x' \in \mathsf{UR}_{\hat{\gamma}_{leg}(\mathsf{NS}(s_o))}(x)} \xi(x') < \xi(x).$$

5.1 On-line Control Algorithm

Main part. The on-line supervisory control scheme is described in Algorithm 1. The supervisor keeps the sets NS of next states after the observation so far, which is initialized with $\{x_{P,0}\}$. At each step k, the on-line supervisor computes a control pattern γ_k that will be applied to NS, the corresponding unobservable reach UR, and the set NS' of *new* next states after potential occurrences of observable events. The functions LegExpand and ReExpand compute $\hat{\gamma}_{leg}(NS)$ and $\hat{\gamma}_{per}(NS)$, respectively (but for now we skip the detailed explanations). After computed, the control pattern γ_k is issued to the DES G, which executes one of the enabled events. The new observation is represented by the function $\mathsf{observe}(G, \gamma_k)$, according to which the supervisor updates information related to the memory NS and time step k and then determines the next control action.

Subfunctions. In the main part of Algorithm 1, we first compute $\hat{\gamma}_{leg}(NS)$ by the function LegExpand. Then, by the function ReExpand, we additionally examine whether other controllable events can be added to the next control pattern. The functions LegExpand and ReExpand expands states in NS in a depth-first-search mannar until an observable event is detected.

The function LegExpand expands an input state $x \in X_P$ and updates $\gamma, \bar{\gamma}, UR$, and NS' if necessary. When a legal transition with an event σ is detected during the search, the event is added to γ. After the call of LegExpand in Algorithm 1, we have $\hat{\gamma}_{leg}(NS)$ and the unobservable reach of NS under the control pattern $\hat{\gamma}_{leg}(NS)$. We move on to the other function ReExpand, which examines each controllable event σ_c that has not been in $\hat{\gamma}_{leg}(NS)$. Unlike LegExpand, we have a global boolean variable st, initialized with false. It is necessary to examine all states that may be visited if σ_c, the currently examined controllable event, is added to γ_k. If we find the event σ_c cannot be in $\hat{\gamma}_{per}(NS)$, then the variable st turns to be true.

5.2 Correctness of Algorithm 1

The on-line supervisor refers to the current rank $\xi(x)$ and the current permissiveness level $\eta(k)$ to take into consideration the tradeoff between permissiveness and acceptance of the specification. More precisely, the more permissiveness level we have, the more neutral transitions we allow to be enabled. Since the permissiveness level decreases with the elapse of time, the supervisor enables fewer events as time goes by. In the following, we show the correctness of Algorithm 1.

Lemma 17. If the product automaton is controllable, then for any $s_o \in \mathcal{P}(\mathcal{L}(P))$, $\hat{\mathcal{S}}(s_o) \neq \emptyset$.

Proof. By Eq. (4), for any $s_o \in \mathcal{P}(\mathcal{L}(P))$, we have $\hat{\mathcal{S}}(s_o) \supseteq \hat{\gamma}_{leg}(\mathsf{NS}(s_o))$, which is always nonempty as mentioned in Proposition 13.

Lemma 18. For any $s_o \in \mathcal{P}(\mathcal{L}(P))$ and any $x \in \mathsf{UR}_\Sigma(\mathsf{NS}(s_o))$, we have $\hat{\mathcal{S}}(s_o) \in \Gamma(J_G(x))$.

Proof. From Proposition 7, we have $\hat{\gamma}_{leg}(x) \in \Gamma(J_G(x))$ for any $x \in X_P$. Then, by Eq. (4) and the definition of $\hat{\gamma}_{leg}(\mathsf{NS}(s_o))$, the lemma holds.

Proposition 19. Assume that the product automaton P is observable. For any $x \in X_P$ with $0 < \xi(x) < \alpha$, there exists $k \in \mathbb{N}$ such that, for any $l \geq k$, $\hat{\gamma}_{per}(x, l) = \emptyset$.

Proof. Note that, for each $x \in X_P$ and $k \in \mathbb{N}$,

$$\hat{\gamma}_{per}(x,k) = \bigcap_{x' \in \mathsf{UR}_\Sigma(x)} \{\sigma \in \Sigma_P(x') : \xi(\delta_P(x',\sigma)) < \eta(k)\}.$$

By the second and third conditions of Definition 10, on the other hand, there exists $\bar{k} \in \mathbb{N}$ such that, for all $\bar{l} \geq \bar{k}$, $\eta(\bar{l}) = 0$. Since the ranking function ξ returns nonnegative values, $\{\sigma \in \Sigma_P : \xi(\delta_P(x,\sigma)) < 0\} = \emptyset$ for any $x \in X_P$. That is, we have $\hat{\gamma}_{per}(x,\bar{l}) = \bigcap_{x' \in \mathsf{UR}_\Sigma(x)} \emptyset = \emptyset$ for all $\bar{l} \geq \bar{k}$.

Lemma 20. Assume that the product automaton P is observable. In the while loop of Algorithm 1, neutral transitions of the product automaton occur only finitely often.

Proof. We prove the lemma by contradiction. Suppose that neutral transitions occur infinitely often. That is, there exists $s \in \Sigma^\omega$ such that

$$\forall k \in \mathbb{N}, \exists l \geq k, \quad \xi(\delta_P(x_{P,0}, s[0 \dots l])) \leq \xi(\delta_P(x_{P,0}, s[0 \dots l+1])). \tag{6}$$

Let $\sigma_{l+1} := s[l+1]$ for some $l \in \mathbb{N}$ that satisfies the above inequality, $x_{l+1} := \delta_P(x_{P,0}, s[0 \dots l])$, and $t_l := \mathcal{P}(s[0 \dots l])$. Then, we have

$$\sigma_{l+1} \in \hat{\mathcal{S}}(t_l) = \hat{\gamma}_{leg}(\mathsf{NS}(t_l)) \cup \hat{\gamma}_{per}(\mathsf{NS}(t_l), \|t_l\|).$$

Since transition $(x_l, \sigma_{l+1}, x_{l+1})$ is neutral, it holds that

$$\sigma_{l+1} \in \hat{\gamma}_{per}(\mathsf{NS}(t_l), \|t_l\|). \tag{7}$$

By Eq. (6), therefore, there exist infinitely many $l \in \mathbb{N}$ satisfying Eq. (7), which contradicts Proposition 19.

Lemma 21. Assume that P is controllable and observable. In Algorithm 1, an accepting state of the product automaton is eventually reached under the control by the on-line supervisor $\hat{\mathcal{S}}$.

Proof. Recall that $\eta(0) \leq \alpha$ and that e is nonincreasing, as mentioned in Definition 10. By the controllability of P, we have $\xi(x_{P,0}) < \alpha$. Since the on-line supervisor $\hat{\mathcal{S}}$ never allows illegal transitions, we have, for any $s \in \mathcal{L}(\hat{\mathcal{S}}/G)$, $\xi(\delta_P(x_{P,0}, s)) < \alpha$. By Proposition 5, then, it is always possible to lead the product automaton to an accepting state by some appropriate event sequence. From Lemma 20, while the on-line computation is running, the supervisor $\hat{\mathcal{S}}$ observes neutral transitions only finitely often. Let $\bar{k} \in \mathbb{N}$ be the step index such that $\eta(\bar{k}) = 0$. Then, for any observation $s_o \in \mathcal{P}(\mathcal{L}(P))$ with $\|s_o\| > \bar{k}$, we have $\hat{\mathcal{S}}(s_o) = \hat{\gamma}_{leg}(\mathsf{NS}(s_o))$. In other words, the supervisor $\hat{\mathcal{S}}$ chooses only legal transitions after time step $\bar{k}$. Since the rank always decreases during each legal transition, eventually a state ranked as 0, namely, an accepting state is reached.

Theorem 22. $\hat{\mathcal{S}}/G \models \varphi$ if P is controllable and observable.

Proof. From Lemma 21, the on-line supervisor forces the plant DES G to generate event sequences with which the product automaton eventually reaches an accepting state. Note that, when an accepting state of the product automaton is reached, then the corresponding word is accepted by the DFA A_φ. By Eq. (1), any run that has the corresponding history as a prefix satisfies the scLTL formula φ.

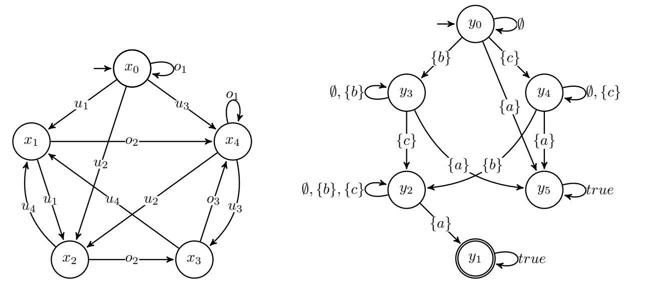

Fig. 1. (Left) The DES G_{ex} discussed in Section 6, where $\Sigma_o = \{o_1, o_2, o_3\}$ and $\Sigma_{uo} = \{u_1, u_2, u_3, u_4\}$. (Right) The DFA $A_{\varphi_{ex}}$ translated from φ_{ex}, where $F_A = \{y_1\}$.

6. ILLUSTRATIVE EXAMPLE

In this section, we demonstrate the proposed method with a simple example. Consider a DES G_{ex} depicted in the left of Fig. 1, where $X = \{x_0, x_1, x_2, x_3, x_4\}$; $\Sigma_o = \{o_1, o_2, o_3\}$ and $\Sigma_{uo} = \{u_1, u_2, u_3, u_4\}$; $AP = \{a, b, c\}$; $L(x_0) = L(x_4) = \emptyset$, $L(x_1) = \{a\}$, $L(x_2) = \{b\}$, and $L(x_3) = \{c\}$. We assume that all events are controllable and the initial state is x_0. For the DES G_{ex}, we impose an scLTL formula $\varphi_{ex} = \Diamond a \wedge \neg a \mathbf{U} b \wedge \neg a \mathbf{U} c$ as a control specification. φ_{ex} requires to eventually go to a state labeled with a after visiting both b-states and c-states. We use a tool Spot [1] to translate φ_{ex} into a DFA, which is shown in the right of Fig. 1.

Let $\eta(k) = \max\{-k+5, 0\}$. Then, at the initial step of the on-line control scheme, the unobservable reach from $NS = \{x_{P,0} = (x_0, y_0)\}$ is computed as shown in Fig. 2. After the call of LegExpand, we have $UR = \{(x_0, y_0), (x_2, y_3)\}$, $NS' = \{(x_3, y_2)\}$, and $\gamma_0 = \hat{\gamma}_{leg}((x_0, y_0)) = \{u_2, o_2\}$. Since the current permissiveness level is $\eta(0) = 5$, the function ReExpand adds u_3, o_1, o_3 to γ_0. The sets UR and NS' are also updated with the related states. We do not expand the state (x_1, y_5) any more because its rank hits the upper bound.

Assume that $\mathsf{observe}(G_{ex}, \gamma_0) = o_1$, according to which the supervisor updates NS to $\{(x_0, y_0), (x_4, y_0)\}$ and ends up obtaining $\gamma_1 = \gamma_0$ after the computation at the next step. Then, assume that the event o_1 is observed again, i.e., $\mathsf{observe}(G_{ex}, \gamma_1) = o_1$. Although we start from the same set $NS = \{(x_0, y_0), (x_4, y_0)\}$ as the previous step, we have a different result. Since $\eta(2) = 3$, the function ReExpand does not add to the current control pattern γ_2 events triggering neutral transitions to states ranked as 3. At the end of the computation for $k = 2$, we have $\gamma_2 = \{u_2, o_2\}$ and $NS' = \{(x_3, y_2)\}$.

Similarly, after observing $o_2 = \mathsf{observe}(G_{ex}, \gamma_2)$, only legal transitions are enabled because all states reachable from (x_3, y_2) have a rank lower than $\eta(3) = 2$. At the end of the computation at step 3, we have $NS' \subseteq F_P$. No matter which event is observed, then, the supervisor stops the control.

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

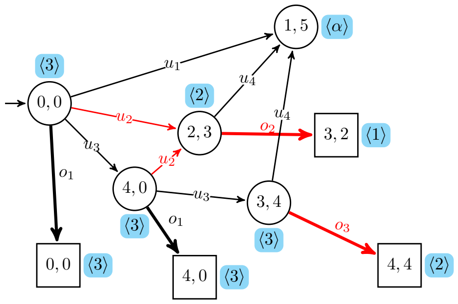

Fig. 2. The computation tree of the unobservable reach at the initial step ($k = 0$), where circle and rectangle nodes represent states in UR and NS', respectively, bold arrows are transitions triggered by an observable event, and legal transitions are colored by red. A node label of the form (i, j) stands for state $(x_i, y_j) \in X_P$. The rank of each state is shown by an additional label of the form $\langle \cdot \rangle$.

7. CONCLUSION

We propose a novel on-line supervisory control scheme of partially observed DESs to achieve a control specification given by scLTL formulas. We introduce the controllability and observability based on the ranking function, which derives a sufficient condition for the existence of the on-line supervisor. In the on-line computation, the supervisor computes the unobservable reach after each observation and refers to the ranking function together with the permissiveness function. Depending on the permissiveness level at each step, the supervisor improves its permissiveness if possible. It is future work to extend the proposed scheme to cases of general or quantitative LTL and to establish a verification method of the observability.

REFERENCES

Baier, C. and Katoen, J.P. (2008). *Principles of Model Checking*. MIT Press.

Belta, C., Yordanov, B., and Aydin Gol, E. (2017). *Formal Methods for Discrete-Time Dynamical Systems*. Springer International Publishing.

Cassandras, G.C. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer US, 2 edition.

Chung, S.L., Lafortune, S., and Lin, F. (1992). Limited lookahead policies in supervisory control of discrete event systems. *IEEE Trans. Autom. Control*, 37(12), 1921–1935.

Chung, S.L., Lafortune, S., and Lin, F. (1993). Recursive computation of limited lookahead supervisory controls for discrete event systems. *Discret. Event Dyn. Syst. Theory Appl.*, 3(1), 71–100.

Chung, S.L., Lafortune, S., and Lin, F. (1994). Supervisory control using variable lookahead policies. *Discret. Event Dyn. Syst. Theory Appl.*, 4(3), 237–268.

Grigorov, L. and Rudie, K. (2006). Near-optimal online control of dynamic discrete-event systems. *Discret. Event Dyn. Syst. Theory Appl.*, 16(4), 419–449.

Hadj-Alouane, N.B., Lafortune, S., and Lin, F. (1996). Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation. *Discret. Event Dyn. Syst. Theory Appl.*, 6(4), 379–427.

Heymann, M. and Lin, F. (1994). On-line control of partially observed discrete event systems. *Discret. Event Dyn. Syst. Theory Appl.*, 4(3), 221–236.

Jiang, S. and Kumar, R. (2006). Supervisory control of discrete event systems with CTL* temporal logic specifications. *SIAM J. Control Optim.*, 44(6), 2079–2103.

Kupferman, O. and Y. Vardi, M. (2001). Model checking of safety properties. *Formal Methods in System Design*, 19(3), 291–314.

Prosser, J.H., Kam, M., and Kwatny, H.G. (1998). Online supervisor synthesis for partially observed discrete-event systems. *IEEE Trans. Autom. Control*, 43(11), 1630–1634.

Ramadge, P.J. and Wonham, W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1), 475–498.

Sakakibara, A. and Ushio, T. (2020). On-line permissive supervisory control of discrete event systems for scLTL specifications. *IEEE Control Systems Letters*, 4(3), 530–535.

Tumova, J. and Dimarogonas, D.V. (2016). Multi-agent planning under local LTL specifications and event-based synchronization. *Automatica*, 70, 239–248.

Appendix A. COMPUTATION OF RANKING FUNCTION

Here, we briefly explain the results in Sakakibara and Ushio (2020), where we propose an algorithm to compute a ranking function given the product automaton P of the DES and the DFA. We obtain a ranking function by Algorithm A.2.

Algorithm A.2 Computation of a ranking function

Input: A product automaton $P = ((X_P, \Sigma_P, \delta_P, x_{P,0}), F_P)$
Output: A ranking function $\xi : X_P \to \mathbb{N}$
$\alpha \leftarrow |X_P| - |F_P| + 1$
for all $x \in X_P$ **do**
  $\xi(x) \leftarrow 0$
while $\exists x \in X_P$ s.t. $\xi(x) < \mathsf{up}_\alpha(\hat{\xi}(x), x)$ **do**
  $\xi(x) \leftarrow \mathsf{up}_\alpha(\hat{\xi}(x), x)$

As initialization, we set $\xi(x) = 0$ for each $x \in X_P$ and $\alpha = |X_P| - |F_P| + 1$. Then, we go on to update the values of ξ by using functions $\hat{\xi} : X_P \to \mathbb{N}$ and $\mathsf{up}_\alpha : \mathbb{N} \times X_P \to \mathbb{N}$, defined as follows. For each $x \in X_P$,

$$\hat{\xi}(x) = \begin{cases} \min_{\sigma \in \Sigma_{P,c}} \xi(\delta_P(x, \sigma)) & \text{if } \Sigma_{P,uc}(x) = \emptyset, \\ \max_{\sigma \in \Sigma_{P,uc}} \xi(\delta_P(x, \sigma)) & \text{otherwise.} \end{cases}$$

For any $r \in \mathbb{N}$ and $x \in X_P$,

$$\mathsf{up}_\alpha(r, x) = \begin{cases} r + 1 & \text{if } x \notin F_P \wedge r < \alpha, \\ r & \text{otherwise.} \end{cases}$$

To sum up, the current rank is incremented if the current state is not accepting with at least one uncontrollable event defined and the rank has not hit the upper bound; otherwise the rank does not change.