2025 IEEE 64th Conference on Decision and Control (CDC)
December 10-12, 2025. Rio de Janeiro, Brazil

Regret-Optimal Supervisory Control of Partially-Known Discrete-Event Systems

Jianing Zhao†, Bohan Cui†, Dimos V. Dimarogonas, Rupak Majumdar and Xiang Yin

2025 IEEE 64th Conference on Decision and Control (CDC) | 979-8-3315-2627-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/CDC57313.2025.11312480

***Abstract*— This paper addresses a novel optimal supervisory control problem for reachability tasks in partially-known discrete-event systems (DES). We consider a setting where the supervisor lacks prior knowledge of feasible events in certain states and must discover this information by visiting them. To assess performance in this context, we study regret as a metric that quantifies the difference between the actual cost incurred and the optimal cost achievable with full knowledge. We formalize this problem and propose the algorithm to compute an optimal supervisor that guarantees reachability while minimizing regret. Our results demonstrate that regret serves as a meaningful performance measure for supervisory control in partially-known DES, and our method is both correct and effective in practice.**

I. Introduction

Discrete-Event Systems (DES) are an important class of systems with discrete state spaces and event-triggered dynamics [1]. Supervisory Control Theory (SCT) is a powerful formal framework widely used for the synthesis of DES controllers under some desired specifications [2]–[4]. One important branch in SCT is the synthesis of optimal supervisors with respect to some performance measures. This problem, known as the optimal supervisory control problem, has been widely investigated in the literature [5]–[10]. In [7], the authors proposed an optimal supervisory control framework considering both the occurrence cost and disablement cost. The supervisor's objective is to reach marked states optimally in terms of the worst-case total accumulated cost.

In most existing works on Supervisory Control Theory (SCT), uncertainties are modeled as uncontrollable events. While this approach is suitable for representing disturbances, it falls short in scenarios where the system structure itself is unknown and the supervisor must actively explore the system to uncover the actual configuration. Relying solely on worst-case analysis can result in an overly conservative supervisor. In fact, a strategy that incurs a slightly higher worst-case cost may yield significantly lower regret by leveraging the potential benefits of exploring unknown regions [11].

In the paper, we formulate and solve a novel optimal supervisory control problem by considering scenarios where the supervisor operates in a *partially-known* DES: the supervisor knows all system states but lacks prior information about feasible transitions from certain states unless explored. Different from nondeterministic DES, where outcomes are random and may vary even under identical conditions, a partially-known DES involves initial uncertainty about the system's structure but remains deterministic. The main results of this paper are as follows: i) We propose a formal model for partially-known DES and formulate the regret-optimal supervisory control problem; ii) We define a novel tripartite transition system to capture all potential knowledge evolution paths under different control decisions and actual environments; iii) We develop a game-theoretical approach to obtain an optimal solution that minimizes regret.

Our work is closely related to graph games with quantitative objectives [12], [13], particularly the regret minimization problem [11], [14]–[16]. However, their setting assumes the environment-player's strategy is unrestricted, allowing it to change decisions freely each time it revisits the same state. This assumption does not align with the partially-known environment scenario considered in our work. One of our previous works [11] started the study of partially-known environment for temporal logic specification. However, only the path planning problem is solved in [11] and the effect of uncontrollable events has not been investigated.

II. Supervisory Control of Fully-Known DES

A. System Model

We follow the standard notations in DES [1] and consider a *fully-known* discrete-event system modeled by a deterministic finite-state automaton (DFA) $G = (X, \Sigma, \delta_G, x_0, X_m)$, where X is the finite set of states, Σ is the finite set of events, $\delta_G : X \times \Sigma \to X$ is the partial transition function, $x_0 \in X$ is the initial state, and $X_m \subseteq X$ is the set of marked states. The transition function δ_G can also be extended to $\delta_G : X \times \Sigma^* \to X$ in the usual manner. The language generated by G from state x is defined by $\mathcal{L}(G, x) = \{s \in \Sigma^* : \delta_G(x, s)!\}$. We also define $\mathcal{L}(G, Q) := \bigcup_{x \in Q} \mathcal{L}(G, x)$. The language generated by G is $\mathcal{L}(G) := \mathcal{L}(G, x_0)$, and the language marked by G is $\mathcal{L}_m(G) = \{s \in \mathcal{L}(G) : \delta_G(s) \in X_m\}$. For any $s \in \mathcal{L}(G)$, we write $\delta_G(x_0, s)$ simply as $\delta_G(s)$. For any $x \in X$, we define $\Lambda_G(x) = \{\sigma \in \Sigma : \delta_G(x, \sigma)!\}$ as the set of active events at x. For any $s \in \mathcal{L}(G)$, we also write $\Lambda_G(\delta_G(s))$ as $\Lambda_G(s)$ for simplicity.

In the supervisory control framework, the event set Σ is partitioned as $\Sigma = \Sigma_c \dot\cup \Sigma_{uc}$, where Σ_c is the set of controllable events and Σ_{uc} is the set of uncontrollable events. A control decision $\gamma \in 2^\Sigma$ is said to be admissible if $\Sigma_{uc} \subseteq \gamma$. Without loss of generality, we consider the

This work was supported by the National Natural Science Foundation of China (62573291,62173226,62188101). † indicates the equal contribution.

J. Zhao, B. Cui and X. Yin are with the School of Automation and Intelligent Sensing, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: {jnzhao,bohan_cui,yinxiang}@sjtu.edu.cn.

D. V. Dimarogonas is with the School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm SE 10044, Sweden. E-mail: dimos@kth.se.

R. Majumdar is with the Max-Planck Institute for Software Systems, Kaiserslautern 67663, Germany. E-mail: rupak@mpi-sws.org.

Algorithm 1: Min-Max Game (`SolveMinMax`)

Input: A BTS $\mathcal{B}$ with the set of marked states X_m and a cost function $w_{\mathcal{B}}$
Output: Optimal strategy $\pi^\star : Q_X \to Q_Y$ and the optimal minmax value minmax

foreach $x \in Q_X$ **do**
  if $x \in X_m$ **then**
    $\mathsf{Val}^{(0)}(x) \leftarrow 0$ and $\pi^{(0)}(x) \leftarrow \Lambda_G(x) \cup \Sigma_{uc}$
  else
    $\mathsf{Val}^{(0)}(x) \leftarrow \infty$
repeat
  foreach $y \in Q_Y$ **do**
    $\mathsf{Val}^{(k+1)}(y) \leftarrow \max_{x' \in \mathrm{Succ}(y)} \Big(\mathsf{Val}^{(k)}(x') + w_{\mathcal{B}}(y, x')\Big)$
  foreach $x \in Q_X$ **do**
    $\mathsf{Val}^{(k+1)}(x) \leftarrow \min_{y \in \mathrm{Succ}(x)} \Big(\mathsf{Val}^{(k)}(y) + w_{\mathcal{B}}(x, y)\Big)$
    $\pi^{(k+1)}(x) \leftarrow \arg\min_{y \in \mathrm{Succ}(x)} \Big(\mathsf{Val}^{(k)}(y) + w_{\mathcal{B}}(x, y)\Big)$
  $k \leftarrow k + 1$
until $\forall z \in Q_X \cup Q_Y : \mathsf{Val}^{(k+1)}(z) = \mathsf{Val}^{(k)}(z)$;
return $\pi^{(k)}, \mathsf{Val}^{(k)}(x_0)$

control decisions in which *at most one controllable event* is included and $\Gamma = \big\{\gamma \in 2^{\Sigma} : \Sigma_{uc} \subseteq \gamma \wedge |\gamma \setminus \Sigma_{uc}| \leq 1\big\}$ as the set of admissible control decisions. Then, a supervisor is a mapping $S : \mathcal{L}(G) \to \Gamma$. We denote by $\Psi(G)$ the set of all supervisors. Moreover, we use the notation S/G to represent the controlled system and the language generated by S/G is denoted by $\mathcal{L}(S/G)$. The language marked by S/G is denoted by $\mathcal{L}_m(S/G) = \mathcal{L}(S/G) \cap \mathcal{L}_m(G)$.

In this paper, we aim to synthesis a supervisor enforcing *reachability objective*, which requires that the controlled DES will eventually visit the marked states, i.e.,

$$\begin{aligned} &\forall s \in \mathcal{L}(S/G), \exists n \in \mathbb{N}, \forall t \in \mathcal{L}(S/G)/s : \\ &|t| \geq n \Rightarrow \overline{\{st\}} \cap \mathcal{L}_m(G) \neq \emptyset \end{aligned} \tag{1}$$

Define $\Psi_W(G)$ as the set of all winning supervisors that enforces reachability.

We consider the optimal supervisory control in a quantitative manner. For each event, we define the cost function $w : \Sigma \to \mathbb{R}_{\geq 0}$ describing the cost incurred whenever the DES generates an event. Given a finite string $t = \sigma_1 \cdots \sigma_n \in \mathcal{L}(G)$, we define $\mathsf{cost}(t) = \sum_{i=1}^{n} w(\sigma_i)$ as the total cost along the string s. For a string $s \in \mathcal{L}_m(G)$, we define its cost as the total cost when it first reaches the set of marked states X_m, i.e., $\mathsf{cost}(s) = \mathsf{cost}(t)$, where

$$t \in \overline{\{s\}} \cap \mathcal{L}_m(G) \wedge t' \notin \mathcal{L}_m(G), \forall t' \in \overline{\{t\}} \setminus \{t\} \tag{2}$$

Given a winning supervisor $S \in \Psi_W(G)$, we define the cost of the controlled system S/G as

$$\mathsf{cost}(S/G) = \max_{s \in \mathcal{L}_m(S/G)} \mathsf{cost}(s) \tag{3}$$

Then, we aim to synthesize an optimal supervisor $S^\star$ that minimizes the cost with the reachability objective, i.e.,

$$\mathsf{cost}(S^\star/G) \leq \mathsf{cost}(S/G), \ \forall S \in \Psi_W(G) \tag{4}$$

B. Controller Synthesis

In this subsection, we review the optimal supervisory control synthesis algorithm for reachability based on the following structure of bipartite transition system [17].

Definition 1 (Bipartite Transition System): Given system G, we construct its bipartite transition system (BTS) as

$$\mathcal{B} = (Q_X, Q_Y, x_0, h_{XY}, h_{YX}, \Sigma, \Gamma)$$

where $Q_X = X \cup \{d\}$ is the set of X-states, where d is the deadlocked state; $Q_Y = X \times \Gamma$ is the set of Y-states; $x_0 \in Q_Y$ is the initial X-state; $h_{XY} : Q_X \times \Gamma \to Q_Y$ is the transition function from X-states to Y-states define by: for any $x \in Q_X$, $\gamma \in \Gamma$ and $y = (x, \gamma) \in Q_Y$, $y = h_{XY}(x, \gamma)$; $h_{YX} : Q_Y \times (\Sigma \cup \{\epsilon\}) \to Q_X$ is the transition function from Y-states to X-states define by: for any $y = (x, \gamma) \in Q_Y$, we have (i) if $\Lambda_G(x) \cap \gamma \neq \emptyset$, we define $h_{YX}(y, \sigma) = \delta(x, \sigma), \ \forall \sigma \in \Lambda_G(x) \cap \gamma$; and (ii if $\Lambda_G(x) \cap \gamma = \emptyset$, we define $h_{YX}(y, \epsilon) = d$. Σ is the set of events in G; Γ is the set of admissible control decisions in G.

Note that the marked states in G are implicitly included in X-states of $\mathcal{B}$, i.e., $X_m \subset Q_X$. We use $\mathrm{Succ}(\cdot)$ to denote the successor-states of a given state in the graph, i.e.,

- for each $x \in Q_X$, we have
$$\mathrm{Succ}(x) = \{h_{XY}(x, \gamma) : \gamma \in \Gamma\}$$
- for each $x \in Q_Y$, we have
$$\mathrm{Succ}(x) = \{h_{YX}(y, \sigma) : \sigma \in \Sigma \cup \{\epsilon\} \text{ s.t. } h_{YX}(y, \sigma)!\}$$

To capture the optimality, we define a cost function for $\mathcal{B}$ as follows: for each $x \in Q_X \setminus \{d\}$ and each $y \in h_{XY}(x)$, we define $w_{\mathcal{B}}(x, y) = 0$; for each $y \in Q_Y$ and each $x' = h_{YX}(y, \sigma) \in h_{YX}(y)$, we define $w_{\mathcal{B}}(y, x') = w(\sigma)$.

Then the optimal supervisor can be obtained by solving a standard *min-max reachability game*, i.e., Algorithm 1. Given the optimal strategy $\pi^\star$ returned by Algorithm 1, in execution, the optimal supervisor $S^\star$ can work as follows: at each instant, $S^\star$ remembers the current X-state x and pick the decision γ such that $h_{XY}(x, \gamma) = \pi^\star(x)$. Then we update the current to Y-state according to γ and wait for the next event to occur. After that, we update the current state again to X-state and so forth.

III. Partially-Known DES

Although uncertainties in environments can be captured by the uncontrollable events in the full-known DES, there are also scenarios where there are initially unknown states in the system that could become known after being visited by system trajectories. We capture such kind of *information uncertainty* by the following partially-known DES.

Definition 2 (Partially-Known DES): A partially-known DES (PK-DES) is a 6-tuple $\mathbb{G} = (X, \Sigma, \delta, \Delta, x_0, X_m)$, where X is the set of states; Σ is the set of events; $\delta : X \times \Sigma \to X$ is the transition function; x_0 is the initial state; X_m is the set of marked states; different from the DFA, $\Delta : X \to 2^{2^{\Sigma}}$ is the *event-pattern function* that assigns each state a family of activated events.

Essentially, PK-DES is used to describe the *possible world* for the system. That is, the system has some prior information regarding the possible events of each unknown state but does not know which ones are activated before the system trajectory actually visits it. Therefore, in the PK-DES $\mathbb{G}$, for each state $x \in X$, we have $\Delta(x) = \{o_1, o_2, \ldots, o_{|\Delta(s)|}\}$ where each $o_i \in 2^\Sigma$ is called an *event-pattern* representing a possible set of activated events at state x. For convenience, we refer each $o_i \in \Delta(x)$ to as an *observation* at state x, since the system "observes" the activated events when visiting state x. Therefore, for each $x \in X$, we say x is a *known state* if $|\Delta(x)| = 1$; and *unknown state* if $|\Delta(x)| > 1$. Accordingly, we partition the state space as $X = X_k \dot\cup X_{uk}$ where X_k is the set of known states and X_{uk} is the set of unknown states. We refer the visit to an unknown state to as an *exploration*.

Definition 3 (Compatible DES): Given $\mathbb{G}$, a DFA $G = (X, \Sigma, \delta_G, x_0, X_m)$ is a compatible DES with $\mathbb{G}$, denoted by $G \in \mathbb{G}$, if for any $x \in X$, we have i) $\Lambda_G(x) \in \Delta(x)$; and ii) for each $\sigma \in \Lambda_G(x)$, we have $\delta_G(x, \sigma) = \delta(x, \sigma)$.

In the partially-known setting, a supervisor cannot be synthesized only based on the finite string generated by the system. In addition, the observed successor-pattern at each state should also be taken into consideration. If the system trajectory visits a known state, then there would not be any new information obtained, since $\Delta(x)$ is already a singleton. Only when the system trajectory visits unknown state, it will gain new information and successor-pattern at this state will become known from then on.

To capture the result of an exploration, we call a tuple $\kappa = (x(\kappa), o(\kappa)) \in X \times 2^\Sigma$ as a *knowledge state*, where $o \in \Delta(x)$ is the set of activated events of x that becomes known to the agent after exploring x. We denote by $\mathsf{Kw} = \{\kappa \in X \times 2^\Sigma : o(\kappa) \in \Delta(x(\kappa))\}$ the set of all possible knowledge states. A *history* in $\mathbb{G}$ is a finite sequence $\hbar = (x_0, o_0)\sigma_0(x_1, o_1)\sigma_1 \cdots (x_n, o_n) \in (\mathsf{Kw} \cdot \Sigma)^* \mathsf{Kw}$ such that i) for any $i = 0, 1, \ldots, n-1$, we have $\sigma_i \in o_i$ and $x_{i+1} = \delta(x_i, \sigma_i)$; ii) for any $i, j = 1, \ldots, n$, we have $x_i = x_j \Rightarrow o_i = o_j$. For such history $\hbar$, we call $\sigma_1\sigma_2 \cdots \sigma_{n-1} \in X^*$ its string. We denote by $\mathcal{H}(\mathbb{G})$ and $\mathcal{L}(\mathbb{G})$ the set of all finite histories and paths in PK-DES $\mathbb{G}$, respectively.

Given knowledge on the PK-DES will be accumulated along a history, which is captured by the following concept of *knowledge set*. A knowledge set $\mathcal{K} = \langle \kappa_1, \ldots, \kappa_{|\mathcal{K}|} \rangle$ where $\kappa_i \in \mathcal{K}$ is an *ordered set* of knowledge states such that

$$\forall \kappa, \kappa' \in \mathcal{K} : x(\kappa) = x(\kappa') \Rightarrow o(\kappa) = o(\kappa').$$

We denote by $\mathbb{KW}$ the set of all knowledge sets. With a slight abuse of notation, for each $x \in X$, we write $x \in \mathcal{K}$ if $(x, o) \in \mathcal{K}$ for some observation $o \in \Delta(x)$. We denote by $o_\mathcal{K}(x) \in \Delta(x)$ the unique observation such that $(x, o_\mathcal{K}(x)) \in \mathcal{K}$.

During the system execution, the system maintains a knowledge set to record its exploration history. Once a new unknown state is explored, the knowledge set is updated according to the following function. Given a knowledge set $\mathcal{K} \in \mathbb{KW}$ and a knowledge state $\kappa \in \mathsf{Kw}$, we have

$$\texttt{update}(\mathcal{K}, \kappa) = \begin{cases} \mathcal{K}, & \text{if } x(\kappa) \in \mathcal{K} \\ \langle \kappa_1, \ldots, \kappa_{|\mathcal{K}|}, \kappa \rangle, & \text{otherwise} \end{cases} \quad (5)$$

Finally, given a knowledge set, the system could refine the PK-DES by eliminating uncertainties that have been explored. Given a PK-DES $\mathbb{G}$ and a knowledge set $\mathcal{K} \in \mathbb{KW}$, the refined PK-DES is a new PK-DES

$$\mathbb{G}_\mathcal{K} = (X, \Sigma, \delta, \Delta', x_0, X_m) \quad (6)$$

such that for any $x \in X$, we have

$$\Delta'(x) = \begin{cases} \{o_\mathcal{K}(x)\}, & \text{if } x \in \mathcal{K} \\ \Delta(x), & \text{if } x \notin \mathcal{K} \end{cases}$$

Therefore, under the setting of partially-known DES, the supervisor should decide the control decision based on both of what the system has visited and what it has known.

Definition 4 (Strategic Supervisor): A strategic supervisor is a function $\mathbb{S} : \mathcal{H}(\mathbb{G}) \to \Gamma$ such that for any $\hbar = \kappa_0\sigma_1\kappa_1\sigma_2 \cdots \kappa_n$ where $\kappa_i = (x_i, o_i)$, we have $\mathbb{S}(\hbar) = \{\sigma_c\} \cup \Sigma_{uc}$, $\sigma_c \in o_n \cap \Sigma_c$, i.e., the supervisor makes a control decision based on the observed set of activated events at x_n. We denote by $\Phi(\mathbb{G})$ the set of all strategic supervisors in $\mathbb{G}$.

Given a strategic supervisor $\mathbb{S} \in \Phi(\mathbb{G})$ and an actual DES $G \in \mathbb{G}$, the controlled system, denoted by $\mathbb{S}/G$, whose language $\mathcal{L}(\mathbb{S}/G)$ is recursively defined as: i) $\epsilon \in \mathcal{L}(\mathbb{S}/G)$; ii) for any $\hbar = \kappa_0\sigma_1\kappa_1\sigma_2 \cdots \kappa_n$, we have $\sigma_1\sigma_2 \cdots \sigma_n \in \mathcal{L}(\mathbb{S}/G)$ iff $\sigma_1\sigma_2 \cdots \sigma_{n-1} \in \mathcal{L}(\mathbb{S}/G)$ and $\sigma_n \in \mathbb{S}(\hbar)$.

In the partially-known setting, since the actual DES $G \in \mathbb{G}$ is unknown *a priori*, we aim to synthesize a strategic supervisor $\mathbb{S} \in \Phi(\mathbb{G})$ such that

$$\forall G \in \mathbb{G} : \mathcal{L}(\mathbb{S}/G) \text{ satisfies (1)} \quad (7)$$

We denote by $\Phi_W(\mathbb{G}) \subseteq \Phi(\mathbb{G})$ the set of all winning strategic supervisors for the reachability objective.

To evaluate the performance of a strategic supervisor $\mathbb{S}$, a natural approach is to consider the *worst-case cost* similar to the definition of (3) among all possible actual DES, i.e.,

$$\mathsf{Cost}_{\text{worst}}(\mathbb{S}) := \max_{G \in \mathbb{G}} \mathsf{cost}(\mathbb{S}/G) \quad (8)$$

However, this metric cannot capture the potential benefit obtained from exploring unknown states. To address this issue, we propose to use *regret* to evaluate the performance of a strategic supervisor, which is defined as follows.

Definition 5 (Regret): Given a partially-known DES $\mathbb{G}$, the regret of a supervisor $\mathbb{S} \in \Phi(\mathbb{G})$ is

$$\mathsf{Reg}(\mathbb{S}) = \max_{G \in \mathbb{G}} \left(\mathsf{cost}(\mathbb{S}/G) - \min_{\mathbb{S}' \in \Phi_W(G)} \mathsf{cost}(\mathbb{S}'/G) \right)$$

Intuitively, for each strategic supervisor $\mathbb{S} \in \Phi(\mathbb{G})$ and each possible actual DES $G \in \mathbb{G}$, $\mathsf{cost}(\mathbb{S}/G)$ is the actual cost incurred when applying $\mathbb{S}$ to this specific DES, while $\min_{\mathbb{S}' \in \Phi_W(G)} \mathsf{cost}(\mathbb{S}'/G)$ is the *best-response cost* if G had been known to the system with hindsight. Therefore, their difference is the regret of applying $\mathbb{S}$ in G. Since the system does not know which $G \in \mathbb{G}$ the actual DES is, the regret of $\mathbb{S}$ is defined as the worst-case regret among all $G \in \mathbb{G}$.

Problem 1 (Regret-Optimal Supervisory Control): Given a partially-known DES $\mathbb{G}$, synthesize a strategic supervisor $\mathbb{S} \in \Phi_W(\mathbb{G})$ such that

$$\forall \mathbb{S}' \in \Phi_W(\mathbb{G}) : \mathsf{Reg}(\mathbb{S}) \leq \mathsf{Reg}(\mathbb{S}'). \quad (9)$$

IV. Main Results

A. Tripartite Transition System

We aim to incorporate the knowledge set into the state space of the game arena and explicitly split the movements of the control decision, the system execution, and the nondeterminism of the environment, which leads to the following notion of *tripartite transition system*.

Definition 6 (Tripartite Transition System): Given a PKDES $\mathbb{G}$, its tripartite transition system (TTS) is a tuple

$$\mathcal{G} = (V_X, V_Y, V_Z, v_0, f_{XZ}, f_{ZY}, f_{YX}, \Sigma, \Gamma)$$

where

- $V_X = X \times \mathbb{KW}$ is the set of X-states;
- $V_Y = X \times \mathbb{KW} \times \Gamma$ is the set of Y-states;
- $V_Z = X \times \mathbb{KW} \times X \cup \{d\}$ is the set of Z-states;
- $v_0 = (x_0, \mathcal{K}_0) \in Q_X$ is the initial state with $\mathcal{K}_0$ being the initial knowledge set;
- $f_{XY} : V_X \times \Gamma \to V_Y$ is the transition function from X-states to Y-states defined by: for any $v_x = (x, \mathcal{K}) \in V_X$, $\gamma \in \Gamma$, and $v_y = (x, \mathcal{K}, \gamma) \in V_Y$, we have
$$v_y = f_{XY}(v_x, \gamma) \tag{10}$$
- $f_{YZ} : V_Y \times (\Sigma \cup \{\epsilon\}) \to V_Z$ is the transition function from Y-states to Z-states defined by: for any $v_y = (x, \mathcal{K}, \gamma) \in V_Y$, we have
  - if $\gamma \cap o_{\mathcal{K}}(x) \neq \emptyset$, we define
  $$f_{YZ}(v_y, \sigma) = (x, \mathcal{K}, \delta(x, \sigma)),\ \forall \sigma \in \gamma \cap o_{\mathcal{K}}(x) \tag{11}$$
  - if $\gamma \cap o_{\mathcal{K}}(x) = \emptyset$, we define $f_{YZ}(v_x, \epsilon) = d$
- $f_{ZX} : V_Z \times 2^{\Sigma} \to V_X$ is the transition function from Z-states to X-states defined by: for any $v_z = (x, \mathcal{K}, x') \in V_Z \backslash \{d\}$ and any $o \in \Delta(x')$, we define
$$v_x = f_{ZX}(v_z, o) = (x', \mathcal{K}') \tag{12}$$
with $\mathcal{K}' = \texttt{update}(\mathcal{K}, (x', o))$;
- Σ is the set of events in G;
- Γ is the set of admissible control decisions in G.

Intuitively, the graph is tripartite with three types of states: the X-states from which the system chooses a feasible control decision; the Y-states from which the system is executed according the control decision chosen before; and the Z-states from which the actual event-pattern is decided in the possible world. Note that the knowledge sets are update *monotonely* since they are ordered sets. Consider the Z-states with at least two successors, i.e., $v_z \in V_Z$ satisfying $\text{Succ}(v_z) \geq 2$, which we call as "Z-states with decisions". Essentially, the knowledge sets in $\mathcal{G}$ will be updated after each Z-state with decisions. Then we build the following property for $\mathcal{G}$.

Proposition 1: In the TTS $\mathcal{G}$, there are no cycles encompassing two different Z-states with decisions.

B. Plays and Strategies

We denote by $V = V_X \cup V_Y \cup V_Z$ the state space of $\mathcal{G}$. Furthermore, we define the set of marked states in $\mathcal{G}$ as $V_m = \{(x, \mathcal{K}) \in V_X : x \in X_m\}$. Given $\mathcal{G}$, we call a finite sequence $\rho = v_x^0 v_y^0 v_z^0 v_x^1 v_y^1 v_z^1 \cdots v_x^n \in (V_X \cdot V_Y \cdot V_Z)^* V_X$ a *play* if $v_x^0 = v_0$ and there are $\gamma_i \in \Gamma$, $\sigma_i \in \Sigma \cup \{\epsilon\}$ and $o_i \in 2^{\Sigma}$ such that $v_y^i = f_{XY}(v_x^i, \gamma_i)$, $v_z^i = f_{YZ}(v_y^i, \sigma_i)$, and $v_x^{i+1} = f_{ZX}(v_z^i, o_i)$ hold for each $i = 0, 1, \ldots, n-1$ and $v_x^i = v_x^n$ for $i = n$. It is obvious that each play in $\mathcal{G}$ induces a history in $\mathbb{G}$, i.e., $\hbar_\rho = (x_0, o_0)\sigma_0(x_1, o_1)\sigma_1 \cdots (x_n, o_n) \in \mathcal{H}(\mathbb{G})$. For convenience, we still use ρ to denote the partial play that does not necessarily end with a X-state.

Since only edges from V_z to V_x represent actual movements, we define a weight function for $\mathcal{G}$ as $w_{\mathcal{G}} : V \times V \to \mathbb{R}_{\geq 0}$ where for any $v_z = (x, \mathcal{K}, x')$ and $v_x = (x', \mathcal{K}')$, we have $w_{\mathcal{G}}(v_z, v_x) = w(x, x')$, $w_{\mathcal{G}}(v_x, v_y) = w_{\mathcal{G}}(v_y, v_z) = 0$. The cost of a play $\rho = v_0 v_1 \cdots v_n \in V^*$ is defined as $\mathsf{cost}_{\mathcal{G}}(\rho) = \sum_{i=0}^{n-1} w_{\mathcal{G}}(v_i, v_{i+1})$.

Given the above TTS $\mathcal{G}$, the strategies are functions

$$\pi_x : V^* V_X \to V_Y \cup \{\text{Null}\},\ \pi_y : V^* V_Y \to V_Z,\ \pi_z : V^* V_Z \to V_X$$

for X-player, Y-player and Z-player, respectively. We denote by $\mathfrak{S}_X(\mathcal{G})$, $\mathfrak{S}_Y(\mathcal{G})$, $\mathfrak{S}_Z(\mathcal{G})$ the sets of all X-strategies, Y-strategies and Z-strategies, respectively. In particular, we say a strategy π is *positional* if $\forall \rho, \rho' : \mathsf{last}(\rho) = \mathsf{last}(\rho') \Rightarrow \pi(\rho) = \pi(\rho')$, where $\mathsf{last}(\cdot)$ denotes the last state of a sequence. We denote by $\mathfrak{S}_j^1(\mathcal{G})$ the set of all positional strategies for j-player, where $j = X, Y, Z$. Given strategies $\pi_x \in \mathfrak{S}_X, \pi_y \in \mathfrak{S}_Y, \pi_z \in \mathfrak{S}_Z$, the outcome play $\rho_{\pi_x, \pi_y, \pi_z}$ is the unique sequence $v_0 v_1 \cdots v_n \in V^* V_X$ such that

- $\forall i < n : v_i \in V_X \Rightarrow \pi_x(v_0 v_1 \cdots v_i) = v_{i+1}$;
- $\forall i < n : v_i \in V_Y \Rightarrow \pi_y(v_0 v_1 \cdots v_i) = v_{i+1}$;
- $\forall i < n : v_i \in V_Z \Rightarrow \pi_z(v_0 v_1 \cdots v_i) = v_{i+1}$;
- $\pi_x(v_0 v_1 \cdots v_n) = \text{Null}$.

Here we remark that, since the objective of the supervisor is to satisfy the reachability w.r.t. the set V_m, once the system trajectory visits V_m, it is unncessary to restrict the behaviors of the system. To this end, in the TTS $\mathcal{G}$, for a play ρ satisfying $\mathsf{last}(\rho) \in \text{Null}$, we set $\pi_x(\rho) = \text{Null}$.

Next, we aim to characterize the sets of strategies for the X, Y, Z-players that are sufficient to solve the problem.

For the Y-player, since there is no restriction for the uncontrollable events and no causal relation between two different events, it is sufficient to consider $\Pi_Y = \mathfrak{S}_Y^1(\mathcal{G})$ for all the possible behaviors generated by Y-player.

For the Z-player, it cannot play arbitrarily since the actual DES is fixed. Therefore, the Z-player must commit to a specific event-pattern it chooses at each unknown state when the game begins. To this end, we define the following notion of *strongly positional strategy*. We say a Z-strategy $\pi_z \in \mathfrak{S}_Z(\mathcal{G})$ is strongly positional if for any two plays $\rho, \rho' \in V^* V_Z$ where $\pi_z(\rho) = (x, \mathcal{K})$ and $\pi_z(\rho') = (x', \mathcal{K}')$, we have i) π_z is positional; and ii) $x = x' \Rightarrow o_{\mathcal{K}}(x) = o_{\mathcal{K}'}(x')$. We denote by $\Pi_Z \subseteq \mathfrak{S}_Z(\mathcal{G})$ the set of all stronly positional strategies for the Z-player.

Finally, we aim to find a winning X-strategy that ensures the outcome play visits the state in V_m. Therefore, we define

$$\Pi_X = \left\{ \pi_x \in \mathfrak{S}_X(\mathcal{G}) : \begin{array}{l} \mathsf{last}(\rho_{\pi_x, \pi_y, \pi_z}) \in V_m, \\ \forall \pi_y \in \Pi_Y, \forall \pi_z \in \Pi_Z \end{array} \right\} \tag{13}$$

Now, similarly to Definition 5, we define the regret of an X-strategy $\pi_x \in \Pi_X$ in $\mathcal{G}$ as (14). It suffices to synthesize

$$\mathsf{Reg}_{\mathcal{G}}(\pi_x) = \max_{\pi_z \in \Pi_Z} \left(\max_{\pi_y \in \Pi_Y} \mathsf{cost}_{\mathcal{G}}(\rho_{\pi_x,\pi_y,\pi_z}) - \min_{\pi'_x \in \Pi_X} \max_{\pi_y \in \Pi_Y} \mathsf{cost}_{\mathcal{G}}(\rho_{\pi'_x,\pi_y,\pi_z}) \right) \tag{14}$$

an X-strategy $\pi_X \in \Pi_X$ that minimizes $\mathsf{Reg}_{\mathcal{G}}(\pi_x)$. In what follows, we present the solution by introducing two algorithms for micro and macro-strategy synthesis.

C. Strategy Synthesis

Due to the possible existence of cycles in $\mathcal{G}$, there is generally an infinite number of winning strategies for X-player, which makes enumerating all of them infeasible. To tackle this issue, we propose to compute *micro-strategies* to capture the strategies that are sufficient to obtain the regret-optimal X-strategy. Based on Proposition 1, we know that, the evolution of Z-states is monotone and for any plays in $\mathcal{G}$ that end with the same state, they must visit the same Z-states with decisions in the same order. Therefore, it suffices to compute the X-strategies between different *layers of Z-states*. To formally capture this, we define the set of initial states for all knowledge sets as

$$V_I = \{v_0\} \cup \{\mathrm{Succ}(v_z) \in V_X : \mathrm{Succ}(v_z) \geq 2, \forall v_z \in V_Z\} \tag{15}$$

For each $v_I \in V_I$, we define the set of all end states as

$$V_F(v_I) = \{v_x \in V_m : \mathcal{K}(v_x) = \mathcal{K}(v_I)\} \cup \left\{ v_x \in V_X : \begin{array}{l} \exists v_z \in V_Z \text{ s.t. } \mathcal{K}(v_z) = \mathcal{K}(v_I) \wedge \\ v_x \in \mathrm{Succ}(v_z) \wedge \mathcal{K}(v_x) \neq \mathcal{K}(v_z) \end{array} \right\} \tag{16}$$

For convenience, in what follows, we use a *tree* in the TTS $\mathcal{G}$ to represent a strategy and we denote by $\mathcal{T}$ the set of all trees in $\mathcal{G}$. For each $\mathcal{T} \in \mathbb{T}$, we denote by $V_{\mathcal{T}}$ and $E_{\mathcal{T}}$ its sets of states and edges. We define the cost of $\mathcal{T}$, denoted by $\mathsf{cost}_{\mathbb{T}}(\mathcal{T})$, as the maximum cost of its plays from the root to the leaves. We denote by $\rho_{\mathcal{T}}$ the play with the maximum cost, i.e., $\mathsf{cost}_{\mathcal{G}}(\rho_{\mathcal{T}}) = \mathsf{cost}_{\mathbb{T}}(\mathcal{T})$. Furthermore, for each $\mathcal{T} \in \mathbb{T}$, it is a map $\mathcal{T} : V_{\mathcal{T}} \to \mathbb{T}$ that assigns each state in $\mathcal{T}$ a subtree. Specifically, we have $\mathcal{T}(v_I) = \mathcal{T}$, where v_I is the root of $\mathcal{T}$.

Definition 7 (Micro-Strategy): Given the TTS $\mathcal{G}$ and for each $v_I \in V_I$, a micro-strategy $\mathcal{T}_{v_I} \in \mathbb{T}$ for X-player is a winning strategy (strategy-tree) such that

i) the root of $\mathcal{T}_{v_I}$ is v_I;
ii) for each $v_x \in V_{\mathcal{T}_{v_I}} \cap V_X$, there is only one successor $v_y \in \mathrm{Succ}(v_x)$ in $\mathcal{G}$ such that $(v_x, v_y) \in E_{\mathcal{T}_{v_I}}$;
iii) for each $v_y \in V_{\mathcal{T}_{v_I}} \cap V_Y$, for any successor $v_z \in \mathrm{Succ}(v_y)$ in $\mathcal{G}$, we have $(v_y, v_z) \in E_{\mathcal{T}_{v_I}}$;
iv) for each $v_z \in V_{\mathcal{T}_{v_I}} \cap V_Z$, all of the successors $v_x \in \mathrm{Succ}(v_z)$ in $\mathcal{G}$ are the leaves;
v) All $v_x \in V_{\mathcal{T}_{v_I}} \cap V_m$ are leaves.

For convenience, for each micro-strategy $\mathcal{T} \in \mathbb{T}$, we denote by $Acc(\mathcal{T})$ the set of all leaves of $\mathcal{T}$.

Intuitively, each micro-strategy captures the partial strategy for X-player in $\mathcal{G}$ that ensures for each $v_I \in V_I$ either the reachability of V_m or a new exploration. For any two micro-strategies $\mathcal{T}_{v_I}$ and $\mathcal{T}'_{v_I}$, we say they are similar if they have the same leaves. For each $v_I \in V_I$, we call the micro-strategy with the minimum cost as the *critical micro-strategy*.

Algorithm 2: Find All Critical Micro-Strategies (FACMiS)

Input: The TTS $\mathcal{G}$ and an initial state v_I
Output: All micro-strategies from v_I to $V_F(v_I)$

Compute $V_F(v_I)$ as in (16);
Define $\Pi_{v_I} : 2^{V_F(v_I)} \to \mathbb{T} \times \mathbb{R}_{\geq 0}$;
Call DFS$(v_I, \{v_I\}, 0)$;
return Π_{v_I}
procedure DFS(v, V_{vis}, C)
Define subtree $\mathcal{T}_v$: *Succ*$(v) \to \mathbb{T}$;
$\mathcal{T}_v \leftarrow$ Null;
if $v \in V_F(v_I)$ **then**
  $Acc \leftarrow V_{vis} \cap V_F(v_I)$;
  if $Acc \notin \Pi_{v_I}$ **then**
    $\Pi_{v_I}(Acc) \leftarrow (\mathcal{T}_v, \mathsf{C})$;
  else
    $(\mathcal{T}, c) \leftarrow \Pi_{v_I}(Acc)$;
    if $\mathsf{C} < c$ **then**
      $\Pi_{v_I}(Acc) \leftarrow (\mathcal{T}_v, \mathsf{C})$;
  return Null
if $v \in V_X$ **then**
  Define $\tilde{\mathsf{C}} \leftarrow \infty$, $\tilde{v}_y \leftarrow$ Null, $\tilde{T} \leftarrow$ Null;
  foreach $v_y \in$ *Succ*(v) **do**
    if $v_y \notin V_{vis}$ **then**
      $V_{vis} \leftarrow V_{vis} \cup \{v_y\}$;
      $T \leftarrow$ DFS$(v_y, V_{vis}, \mathsf{C} + w_{\mathcal{G}}(v, v_y))$;
      $V_{vis} \leftarrow V_{vis} \backslash \{v_y\}$;
      if $\mathsf{C} + w_{\mathcal{G}}(v, v_y) +$ *cost*$_{\mathbb{T}}(T) < \tilde{\mathsf{C}}$ **then**
        $\tilde{\mathsf{C}} \leftarrow \mathsf{C} + w_{\mathcal{G}}(v, v_y) + \mathsf{cost}_{\mathbb{T}}(T)$;
        $\tilde{v}_y \leftarrow v_y$;
        $\tilde{T} \leftarrow T$;
  if $\tilde{v}_y \neq$ *Null* **then**
    $\mathcal{T}_v(\tilde{v}_y) \leftarrow \tilde{T}$;
if $v \in V_Y \cup V_Z$ **then**
  foreach $v' \in$ *Succ*(v) **do**
    if $v' \notin V_{vis}$ **then**
      $V_{vis} \leftarrow V_{vis} \cup \{v'\}$;
      $T \leftarrow$ DFS$(v', V_{vis}, \mathsf{C} + w_{\mathcal{G}}(v, v'))$;
      $V_{vis} \leftarrow V_{vis} \backslash \{v'\}$;
      $\mathcal{T}_v(v') \leftarrow T$;
return $\mathcal{T}_v$

Now, we present the algorithm for finding all critical micro-strategies as Algorithm 2.

Proposition 2: For each $v_I \in V_I$, all its the critical micro-strategies are returned by Algorithm 2.

With the critical micro-strategies for *one layer of Z-states with decisions*, we next introduce the notion of *macro-strategy* that connects the *different layers of Z-states with decisions* by the micro-strategies.

Definition 8 (Macro-Strategy): Given $\mathcal{G}$, a macro-strategy is a map $\xi : V_I \to \{\Pi_{v_I} : v_I \in V_I\}$ such that for each $v_I \in V_I$, we have $\xi(v_I) = \mathcal{T}_{v_I}$ for some micro-strategy $\mathcal{T}_{v_I} \in \Pi_{v_I}$.

Given a partial macro-strategy ξ, we denote by $\mathsf{dom}(\xi) \subseteq V_I$ the domain of ξ. For each ξ, we define

$$\Upsilon(\xi) = \left(\{v_0\} \cup \bigcup_{v' \in \mathsf{dom}(\xi)} (Acc(\xi(v')) \cap V_X) \right) \backslash \mathsf{dom}(\xi)$$

Algorithm 3: Find Regret-Optimal Macro-Strategy

Input: The TTS $\mathcal{G}$ and an initial state v_0
Output: The regret-optimal macro-strategy $\xi^\star$ and the minimized regret $\mathsf{Reg}^\star$

Construct an empty macro-strategy $\xi : V_I \to \mathbb{T}$ with $\mathsf{dom}(\xi) = \emptyset$;
$\mathsf{Reg}^\star \leftarrow \infty$ and $\xi^\star \leftarrow \xi$;
Call DFS2(v_0, ξ);
procedure DFS2(v, ξ)
if $v \notin \mathit{dom}(\xi)$ **then**
  foreach $\mathcal{T} \in \Pi_v$ **do**
    $\xi(v) \leftarrow \mathcal{T}$;
    ADVANCE(ξ);
    $\xi(v) \leftarrow \emptyset$;
else
  ADVANCE(ξ);
procedure ADVANCE(ξ)
if $\Upsilon(\xi) = \emptyset$ **then**
  $r \leftarrow$ REGRET(ξ);
  if $r < \mathit{Reg}^\star$ **then**
    $\mathsf{Reg}^\star \leftarrow r$ and $\xi^\star \leftarrow \xi$;
else
  Choose any $v' \in \Upsilon(\xi)$;
  Call DFS2(v', ξ);

as *the set of frontier nodes* of ξ, representing the X-states for which the micro-strategies need to be assigned next. Based on this, we say ξ is a *complete strategy* if $\Upsilon(\xi) = \emptyset$. Since $V_F(v_I) \subseteq V_I, \forall v_I \in V_I$, it is sufficient to form a *complete X-strategy* with one complete macro-strategy with corresponding micro-strategies, each of which starts from either initial state v_0 or one leave of the last micro-strategy.

To compute regret of a given complete macro-strategy ξ, we first characterize the set of all Z-strategies Π_Z as follows. Given the unknown states $x_1, \ldots, x_n \in X_{uk}$, we construct

$$\Theta = \{(x_1, o_1) : o_1 \in \Delta(x_1)\} \times \cdots \times \{(x_n, o_n) : o_n \in \Delta(x_n)\}$$

as the set of all knowledges to be explored at the unknown states. For each $\theta = \langle (x_1, o_1), \cdots, (x_n, o_n) \rangle \in \Theta$, we define π_z^θ as: for any $v_z = (x, \mathcal{K}, x_i)$ where $x_i \in X_{uk}$, we have

$$\pi_z^\theta(v_z) = (x_i, \mathcal{K}'),\ \mathcal{K}' = \mathtt{update}(\mathcal{K}, (x_i, o_i))$$

For $v_z = (x, \mathcal{K}, x')$ where $x' \in X_k$, we have $\pi_z^\theta(v_z) = (x', \mathcal{K})$. By th above construction, we have $\Pi_Z = \{\pi_z^\theta : \theta \in \Theta\}$.

Given a Z-strategy π_z^θ, the TTS $\mathcal{G}$ is induced to a tree $\mathcal{G}_{\pi_z^\theta}$ which is essentially a game between X and Y. Therefore, we call Algorithm 1 to compute the best-response cost against π_z^θ, i.e., $\mathsf{minmax}(\mathcal{G}_{\pi_z^\theta}) \leftarrow \mathtt{SolveMinMax}(\mathcal{G}_{\pi_z^\theta}, V_m, w_{\mathcal{G}})$. Given a complete macro-strategy ξ and a Z-strategy π_z^θ, the TTS $\mathcal{G}$ is induced to a tree $\mathcal{G}_{\xi,\pi_z^\theta}$. We denote by $\rho(\mathcal{G}_{\xi,\pi_z^\theta})$ the play in tree $\mathcal{G}_{\xi,\pi_z^\theta}$ with the maximum cost $\mathsf{cost}(\rho(\mathcal{G}_{\xi,\pi_z^\theta}))$, which can be directly computed by a tree dynamic programming. Then, we define

$$\mathrm{REGRET}(\xi) = \max_{\pi_z^\theta \in \Pi_Z} \left(\mathsf{cost}(\rho(\mathcal{G}_{\xi,\pi_z^\theta})) - \mathsf{minmax}(\mathcal{G}_{\pi_z^\theta})\right)$$

where we use $Acc(\mathcal{G}_{\xi,\pi_z^\theta})$ to denote all the leaves of $\mathcal{G}_{\xi,\pi_z^\theta}$.

Now, we present the algorithm for finding the regret-optimal macro-strategy as Algorithm 3.

Theorem 1: Given the TTS $\mathcal{G}$, the X-strategy formed by the macro-strategy returned by Algorithm 3 and the micro-strategies returned by Algorithm 2 minimizes the regret for X-player, i.e., Problem 1 is solved with minimized $\mathsf{Reg}^\star$.

Due to space constraint, the reader is referred to `https://jnzhaooo.github.io/files/regret.pdf` for detailed proofs as well as an illustrative example for our control synthesis algorithm.

V. Conclusion

In this paper, we formulate and solve the regret-optimal supervisory control problem for partially-known discrete-event systems. To model the system's interaction with uncertain environments, we introduce a tripartite transition structure that captures all possible actual environments. By decomposing the problem into a two-stage reachability game, we develop the solution algorithm. Our results demonstrate that regret serves as a more meaningful performance metric than conventional approaches for handling information that is deterministic yet initially unknown. This result extends optimal SCT beyond traditional worst-case analysis.

References

[1] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Springer, 2008.
[2] M. Reniers and K. Cai, "Supervisory control theory with event forcing," *IEEE Transactions on Automatic Control*, 2024.
[3] A. M. Mainhardt and A.-K. Schmuck, "Assume-guarantee synthesis of decentralised supervisory control," *IFAC-PapersOnLine*, vol. 55, no. 28, pp. 165–172, 2022.
[4] Y. Xie, S. Li, and X. Yin, "Optimal synthesis of opacity-enforcing supervisors for qualitative and quantitative specifications," *IEEE Transactions on Automatic Control*, vol. 69, no. 8, pp. 4958–4973, 2023.
[5] Y. Ji, X. Yin, and S. Lafortune, "Optimal supervisory control with mean payoff objectives and under partial observation," *Automatica*, vol. 123, p. 109359, 2021.
[6] L. V. Alves, P. N. Pena, and R. H. Takahashi, "Planning on discrete event systems using parallelism maximization," *Control Engineering Practice*, vol. 112, p. 104813, 2021.
[7] R. Sengupta and S. Lafortune, "An optimal control theory for discrete event systems," *SIAM Journal on control and Optimization*, vol. 36, no. 2, pp. 488–541, 1998.
[8] Z. Ma and K. Cai, "Optimal secret protections in discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 67, no. 6, pp. 2816–2828, 2021.
[9] Y. Xie, X. Yin, and S. Li, "Opacity enforcing supervisory control using nondeterministic supervisors," *IEEE Transactions on Automatic Control*, vol. 67, no. 12, pp. 6567–6582, 2021.
[10] P. Lv, Z. Xu, Y. Ji, S. Li, and X. Yin, "Optimal supervisory control of discrete event systems for cyclic tasks," *Automatica*, vol. 164, p. 111634, 2024.
[11] J. Zhao, K. Zhu, M. Feng, S. Li, and X. Yin, "No-regret path planning for temporal logic tasks in partially-known environments," *The International J. Robotics Research*, p. 02783649251315758, 2025.
[12] K. Chatterjee, M. Randour, and J.-F. Raskin, "Strategy synthesis for multi-dimensional quantitative objectives," *Acta informatica*, vol. 51, no. 3, pp. 129–163, 2014.
[13] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic model checking and autonomy," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 385–410, 2022.
[14] E. Filiot, T. L. Gall, and J.-F. Raskin, "Iterated regret minimization in game graphs," in *International Symposium on Mathematical Foundations of Computer Science*, pp. 342–354, 2010.
[15] P. Hunter, G. A. Pérez, and J.-F. Raskin, "Reactive synthesis without regret," *Acta Informatica*, vol. 54, no. 1, pp. 3–39, 2017.
[16] M. Cadilhac, G. A. Pérez, and M. v. d. Bogaard, "The impatient may use limited optimism to minimize regret," in *International Conference on Foundations of Software Science and Computation Structures*, pp. 133–149, Springer, 2019.
[17] X. Yin and S. Lafortune, "Synthesis of maximally permissive supervisors for partially-observed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 5, pp. 1239–1254, 2015.