Local Mean Payoff Supervisory Control for Discrete Event Systems

Yiding Ji, *Member, IEEE*, Xiang Yin, *Member, IEEE*, and Stéphane Lafortune, *Fellow, IEEE*

***Abstract*—This article investigates quantitative supervisory control with local mean payoff objectives on discrete event systems modeled as weighted automata. Weight flows are generated as new events occur, which are required to satisfy some quantitative conditions. We focus on mean weights (payoffs) over a finite number of events, which serve as a measure for the stability or robustness of weight flows. The range of events to evaluate the mean payoff is termed a window, which slides as new events occur. Qualitative requirements such as safety and liveness are also necessary along with quantitative requirements. Supervisory control is employed to manipulate the operation of the system so that the requirements are satisfied. We consider two different scenarios based on whether the window size is fixed or not. Correspondingly, we formulate two supervisory control problems, both of which are solved sequentially by first tackling the qualitative issues and then the quantitative ones. The automaton model is then transformed to a two-player game between the supervisor and the environment, where safety and liveness are enforced. Based on the intermediate results, several quantitative objectives are defined to formulate two games, which correspond to the two proposed supervisory control problems. Finally, we synthesize provably correct supervisors by solving the games and completely resolve both problems.**

***Index Terms*—Algorithmic game theory, automata, discrete event systems, mean payoff, supervisory control.**

I. Introduction

IN THE context of discrete event systems (DES), supervisory control is a central topic. The plant under control is usually modeled as a finite discrete structure, and a specification is given as the desired behavior of the plant. The supervisor restricts the behavior of the plant by enabling or disabling some events so that the specification is achieved [6], [44].

Manuscript received October 17, 2020; revised February 16, 2021; accepted April 17, 2021. Date of publication April 27, 2021; date of current version April 26, 2022. This work was supported in part by the United States National Science Foundation under Grant CNS-1738103, also by the National Natural Science Foundation of China under Grants 62061136004, 61803259, and 61833012. Recommended by Associate Editor K. Cai. *(Corresponding author: Xiang Yin.)*

Yiding Ji is with the Division of Systems Engineering, Boston University, Boston, MA 02215 USA (e-mail: jiyiding@umich.edu).

Xiang Yin is with the Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: yinxiang@sjtu.edu.cn).

Stéphane Lafortune is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: stephane@umich.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TAC.2021.3075186.

Digital Object Identifier 10.1109/TAC.2021.3075186

Ever since supervisory control theory was initiated, it has been thoroughly investigated in various models of DES, including automata [41], [49], Petri nets [11], [26], [27], and other structures [12], [45]. As an important extension, supervisory control under partial observation has attracted considerable attention; for recent references, see, e.g., [1], [2], [5], [9], [16], [38], [39], and [47]. Particularly, a uniform supervisory control approach was proposed in [48] to enforce a series of properties on partially observed DES. Other mechanisms of supervisory control have also been developed, such as decentralized control [22], [46], distributed control [21], supervisor reduction [24], control of timed DES [36], learning-based supervisor synthesis [10], [50], compositional control [31], control under attacks [25], [30], [43], and so on. In parallel with qualitative analysis, quantitative supervisory control has also been studied, where some quantitative measures are introduced to evaluate the supervisor's performance. A classic topic is optimal supervisory control/stabilization; see, e.g., [14], [15], [29], [32], and [33] for works covering different perspectives.

In many engineering applications, the system generates or consumes certain resources during its operation. It is critical to ensure that the long-run average rate or total amount of resource generation/consumption remains reasonable. Supervisory control may be employed to enforce such objectives. Specifically, optimal makespan or throughput supervisors were discussed in [40] and [42], which considered timed automata and limit average time of strings. More recently, some works investigated optimal supervisory control under a game theoretic framework [19], [34], [35]; however, they all focus on asymptotic properties while ignoring transient properties.

Consider supervisory control in power management systems for hybrid electric vehicles (HEVs); see, e.g., [28]. The supervisory controller tunes the torque so that either a positive or negative torque is demanded from the powertrain according to the driving mode. Power is either generated by the electric machine or absorbed from the driveline to charge the battery. Specifically, the rate of power supply should remain high enough for the stable operation of the vehicle.

Another example is data transmission through a communication network modeled as a DES. Each packet transmission can be modeled as an event while the event weights could represent the number of bits contained in each packet. The information flow is generated when packets are transmitted through the network. At each stage, the sender transmits certain packets according to the receiver's capacity. After those packets are successfully received, the sender moves on to the next stage to start another round of transmission. We may imagine that there is a sliding "window" over the network, where the window size indicates the number of available time slots to send packets. The window size may vary dynamically at each stage depending on the real-time network status. Unfortunately, the network is not trustworthy and some packets may be lost due to malfunction or disturbances. Therefore, the integrity of data would be seriously affected if a high volume of data is transmitted in a small number of windows and the packets in those windows are lost. In that sense, it makes sense to bound the amount of bits transmitted per window to improve the *robustness* of network flows against disturbances.

Motivated by the above situations, we explore local mean payoff supervisory control on weighted discrete event systems in this article. Each event is associated with a weight which represents a certain resource of interest. With the occurrence of events, weight flows are generated, also under the control of the supervisor. Specifically, we consider two supervisory control scenarios where the supervisor regulates the local mean payoff to reduce fluctuation beyond prespecified bounds within a finite number of events. For both scenarios, qualitative requirements like safety and liveness are also imposed, i.e., no undesired state is reached and the system never terminates. The horizon to evaluate the mean payoff is called as the *window* which is sliding with new events occurring.

In the first scenario, the supervisor ensures that local mean payoff over a finite number of transitions lies above certain threshold, which is termed a *desirable window*. In the second scenario we consider a variant called *N-step desirable window* which requires that the weight flows satisfy the bounds within a window of fixed size N. This naturally comes from practical situations where stable or robust flows should be achieved in uniformly bounded steps or when the surveillance of flow status is taken in every fixed time unit.

Both problems are solved in a sequence. As a first step, we transform the system model from a weighted automaton to a two-player game between the supervisor and the environment. Then we introduce the generic definition of *weighted bipartite transition system (WBTS)*, which is the game graph. Then a special WBTS is constructed, where we define some relevant concepts so as to tackle the safety and liveness issues.

Though the two problems look similar at first glance, they are solved in totally different manners to resolve quantitative issues. Two different games are formulated, corresponding to the two problems. In the first case, results from *total payoff games* [4] are leveraged to compute the supervisor's winning region and an algorithm is proposed to synthesize supervisors. In the second case, *window payoff functions* are defined and another game is formulated. Then we derive the final solution based on properly unfolding the game graph. Herein, we provide systematic methods to synthesize winning strategies for both games and show that if the supervisor has strategies to win the game, then there exist solutions to the corresponding local mean payoff supervisory control problem. Note that the solutions to the two problems are incomparable; thus, each solution is not applicable to the other problem.

Under the framework of local mean payoff supervisory control in this article, a supervisor issues the current command based on the mean payoff within a limited lookahead, generating a path. Then it issues a new sequence of decisions "within the window" upon the occurrence of a new event and a new path is generated. *Limited lookahead supervisory control* has been studied in DES [8], where the supervisor is only capable of observing limited future events. This is similar to our framework in the sense of evaluating the supervisor's decisions within a limited horizon. However, only qualitative specifications like safety and nonblockingness are considered in existing works of limited lookahead supervisory control; so our framework is significantly different from theirs.

The problem formulations and solution procedures in this article are also inspired by the literature in algorithmic game theory in computer science [3], especially quantitative games like mean payoff games [3] and mean payoff games with window objectives [7]. Some works leverage results from algorithmic game theory to investigate problems in DES, such as [17], [19], [34], and [35]. However, they either consider different supervisory control objectives like limit mean payoff or total payoff [19], [34], [35] or investigate a totally different problem like opacity enforcement [17]. To the best of our knowledge, this article is the first to consider local mean payoff supervisory control problems under full observation in DES. A more recent paper [20] adopted a similar setting while investigating local mean payoff supervisory control under partial observation.

The following sections are organized as follows. Section II describes the system model. Section III formulates two problems: *supervisory control under desirable windows* and *supervisory control under N-step desirable windows*. In Section IV, we transform the proposed problems into two-player games and introduce the WBTS based on which those problems are partially solved and the qualitative requirements are enforced. Section V completely solves the first proposed problem by introducing and solving a quantitative game. In Section VI, we formulate and analyze another game to completely solve the second proposed problem. Finally, Section VII concludes the article.

A preliminary version of this article with partial results appears in [18], which only considers the second problem discussed in this article. The improvements of the current work are twofold: A new problem of local mean payoff supervisory control is discussed under "unfixed" desirable windows; some necessary proofs and further analysis concerning the second problem are provided as well, which are missing in [18].

II. System Model

We consider a quantitative discrete event system modeled by a weighted finite-state automaton

$$G = (X, E, f, x_0, \omega)$$

where X is the finite state space, E is the finite set of events, $f : X \times E \to X$ is the partial transition function, $x_0 \in X$ is the initial state, and $\omega : E \to \mathbb{Z}$ is the weight function that assigns an integer vector to each event. The weight reflects change of the quantitative resource associated with each event, which may be positive or nonpositive. The domain of f can be extended to $X \times E^*$ in the standard manner [6] and we still denote the extended function by f. The language generated by G is $\mathcal{L}(G) = \{s \in E^* : f(x_0, s)!\}$ where ! means "is defined." The function ω is additive and its domain can be extended to E^* by letting $\omega(\epsilon) = 0$, $\omega(se) = \omega(s) + \omega(e)$ for all $s \in E^*$ and $e \in E$. In this article, we denote by W the maximum absolute value of event weights in G, i.e., $W = \max_{e\in E} |\omega(e)|$.

In G, if $f(x_1, e) = x_2$ for some $x_1, x_2 \in X$, and $e \in E$, then we write $x_1 \xrightarrow{e} x_2$ for simplicity. A *run* is a finite or infinite sequence of alternating states and events in the form: $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} x_n$. A run is *initial* if it starts from the initial state of G. We denote by $\mathrm{Run}(G)$ and $\mathrm{Run}_{\mathrm{inf}}(G)$ the set of runs and infinite runs in G, respectively. For index $1 \leq i \leq n$, we call $x_i \xrightarrow{e_i} \cdots \xrightarrow{e_n} x_{n+1}$ a *suffix* of r and $x_1 \xrightarrow{e_1} \cdots \xrightarrow{e_i} x_{i+1}$ a *prefix* of r. In addition, for indexes j and m such that $1 \leq j < m \leq n$, we call $x_j \xrightarrow{e_j} x_{j+1} \xrightarrow{e_{j+1}} \cdots \xrightarrow{e_m} x_{m+1}$ a *fragment* of r, which is a run by itself. Furthermore, we let $r(j, m)$ stand for the run fragment starting from x_j and ending in x_{m+1}.

A run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} x_n$ is a *cycle* if $x_1 = x_n$, and r is a *simple cycle* if $\forall i, j \in \{1, 2, \ldots, n-1\}$, $i \neq j \Rightarrow x_i \neq x_j$. If r is a cycle, the corresponding string $e_1e_2 \cdots e_{n-1}$ forms a *loop*, and the loop is simple if r is simple. A run is *acyclic* if none of its fragments is a cycle; otherwise, it is *cyclic*.

We discuss *safety* in a state-based manner and let $X_{\mathrm{us}} \subset X$ be the set of unsafe states in G. The readers may refer to [13] for how to convert a language-based specification to a state-based one on an automaton. Marked states usually represent states of particular interest and concern language nonblockingness, which is not the focus of this article. Therefore, state marking is not included in our system model. Instead, we consider a *weak* version of *liveness:* G is live if its language $\mathcal{L}(G)$ is live, i.e., $\forall s \in \mathcal{L}(G)$, $\exists u \in E$, s.t. $su \in \mathcal{L}(G)$. That is, a transition is always defined out of any state in G; thus, every finite run may be extended to an infinite one. This condition is not restrictive as it may be relaxed by adding observable self-loops at states where no active events are defined. We will omit the word "weak" in the following context when there is no confusion.

The system G is controlled by a *supervisor* which dynamically enables and disables events so that some specification is achieved [6]. Formally, a supervisor is a function $S : \mathcal{L}(G) \to 2^E$ and we denote by $\mathbb{S}$, the set of supervisors. The event set E is partitioned as $E = E_c \cup E_{\mathrm{uc}}$, where E_c is the set of controllable events and E_{uc} is the set of uncontrollable events. A control decision $\gamma \in 2^E$ is *admissible* if $E_{\mathrm{uc}} \subseteq \gamma$, i.e., no uncontrollable event is disabled. Denote by Γ the set of all admissible control decisions. In this article, all events are *observable* and only admissible control decisions are considered; so controllability is preserved. We use S/G to represent the controlled system under S and, accordingly, denote by $\mathcal{L}(S/G)$ the language generated in S/G and $\mathrm{Run}(S/G)$ the set of runs in S/G, respectively. As marked states are not involved in G, we do not consider the standard nonblockingness of supervisors [6]. In the remainder of the article, a supervisor is called *safe* and (weakly) *live* if its supervised system is both safe and live.

Given a run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_n} x_{n+1}$ in G, its *(total weight/payoff* is $\sum_{i=1}^{n} \omega(e_i)$ and its *mean weight/payoff* is $\frac{1}{n}\sum_{i=1}^{n} \omega(e_i)$. As illustrated in Section I, the mean weight within a sliding "window" provides a measure of stability or robustness of weight flows, while the window size reflects the length of the horizon within which we evaluate those properties. In contrast to the limit mean payoff which evaluates the "global" asymptotic performance of the system, we focus on the local mean payoff in this article. Note that the local mean payoff is an approximation of the limit mean payoff since the former will essentially become the latter when the size of the windows approaches infinity.

In this article, we require the local mean weight to be above a given threshold and consider two scenarios: one is over a bounded number of events and the other is over a fixed number of events. Correspondingly, we have the following two definitions to evaluate the local mean payoff.

Definition 1 (Desirable Window): Given G and mean payoff bound $v \in \mathbb{Z}$, a finite run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_m} x_{m+1}$ in G forms a desirable window if $\exists \ell \leq m$ such that $\frac{1}{\ell}\sum_{i=1}^{\ell} \omega(e_i) \geq v$.

A desirable window is formed if the mean payoff turns to be no less than a given bound within a finite number of events. On the other hand, we say r in Definition 1 forms an *undesirable window* if $\forall 1 \leq \ell \leq m$, $\frac{1}{\ell}\sum_{i=1}^{\ell} \omega(e_i) < v$. If we interpret an undesirable window as deviation from the preferred reference or disturbance of the normal performance, then it should be compensated or mitigated by supervisory control.

Definition 2 (N-Step Desirable Window): Given system G, fixed window size $N \in \mathbb{N}^+$, and mean payoff bound $v \in \mathbb{Z}$, a finite run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_N} x_{N+1}$ in G forms an N-step desirable window if $\exists \ell \leq N$ such that $\frac{1}{\ell}\sum_{i=1}^{\ell} \omega(e_i) \geq v$.

As is seen, an *N*-step desirable window is a special desirable window since the length of the desirable window is fixed. In the remainder of the article, we assume $N \geq 2$ to avoid the case where a one-step desirable window can be trivially determined by checking each individual event weight in G. Both Definition 1 and Definition 2 are defined for finite runs. Then we let the windows slide with new event occurrences and evaluate the local mean weight for infinite runs.

Definition 3 (Desirable-Window Infinite Run): Given system G and mean payoff bound $v \in \mathbb{Z}$, a run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \in Run_{inf}(G)$ is a desirable-window infinite run if $\exists i \geq 1$ such that $\forall j \geq i, \exists m_j \geq 0$, we have that run fragment $r(j, j + m_j)$ forms a desirable window.

Definition 4 (N-Step Desirable-Window Infinite Run): Given system G, maximum window size $N \in \mathbb{N}^+$ and mean payoff threshold $v \in \mathbb{Z}$, a run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \in Run_{inf}(G)$ is an N-step desirable-window infinite run if $\exists i \geq 1$ such that $\forall j \geq i$, we have that $r(j, j + N)$ forms an N-step desirable window.

Both Definitions 3 and 4 characterize local mean payoff objectives defined over a finite number of events, which are in contrast to the limit (global) mean payoff objective defined over an infinite number of events in [19]. Furthermore, it may

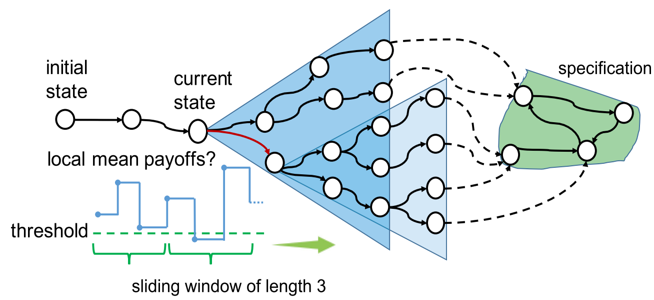


Fig. 1. Sliding windows and the local mean payoffs.

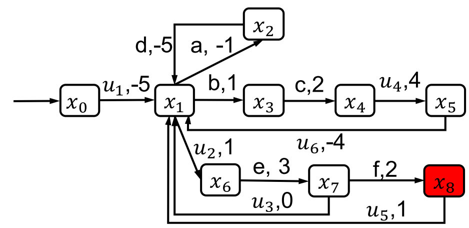


Fig. 2. Weighted automaton G in Example 1.

be tolerable to allow violations of the mean payoff bound for a finite number of times in some applications. Therefore, it seems more practical to enforce the local mean payoff objective after the system has been operating for a while. That is why we require that desirable windows (N-step desirable windows) be perpetually achieved from certain position x_i, not necessarily the initial state x_0 of G, in Definition 3 (Definition 4). In other words, both Definitions 3 and 4 are *independent* of finite run prefixes. When the system is live, desirable or N-step desirable windows may appear infinitely often. Again we assume that $N \geq 2$ in Definition 4.

Note that the inequalities in both Definitions 1 and 2 are the same as $\frac{1}{\ell}\sum_{i=1}^{i=\ell}(\omega(e_i) - v) \geq 0$, i.e., we may subtract v from each event weight and equivalently evaluate whether the mean payoff is above 0. In the following discussion, we just assume $v = 0$ without loss of generality. Mean payoff of runs with sliding windows of length three is illustrated in Fig. 1. As is seen, the local mean payoff is evaluated every three events and the window slides to the next position after event e occurs.

III. Problem Formulation

When safety is violated or the local mean payoffs of some runs lie outside the prescribed bound, supervisory control is employed to mitigate those issues. In this section, we formulate two local mean payoff supervisory control problems: *supervisory control under desirable windows* and *supervisory control under N-step desirable windows*. In both problems, supervisors enforce qualitative and quantitative specifications.

Problem 1 (Supervisory Control Under Desirable Windows): Given system G with unsafe state set X_{us} and mean payoff bound $v \in \mathbb{Z}$, design a supervisor $S \in \mathbb{S}$ such that: 1) S/G is both safe and live; 2) for all $r \in Run_{inf}(S/G)$, r is a desirable-window infinite run.

In addition to safety and liveness, Problem 1 requires that every infinite run in the supervised system is a desirable-window infinite run. Then we fix the size of the desirable windows and formulate Problem 2 as follows.

Problem 2 (Supervisory Control Under N-Step Desirable Windows): Given system G with the unsafe state set X_{us} and fixed window size $N \in \mathbb{N}^+$, design a supervisor $S \in \mathbb{S}$ such that: 1) S/G is both safe and live; 2) for all $r \in R_{inf}(S/G)$, r is an N-step desirable-window infinite run.

Remark 1: Given an infinite run r as in Definition 3, suppose x_j with $j > 1$ is the first position where a nonnegative total payoff (desirable window) is achieved, i.e., $\sum_{i=1}^{j}\omega(e_i) \geq 0$ and $\sum_{i=1}^{j'}\omega(e_i) < 0$ for all $j' < j$. By some derivation, we know that $\sum_{i=j'}^{j}\omega(e_i) \geq 0 > \sum_{i=1}^{j'-1}\omega(e_i)$ holds for any $j' < j$; otherwise, it contradicts with x_j being the first place where a desirable window is achieved. So any run fragment $r(j,' j)$ also forms a desirable window. This fact is called *inductive property* and we will apply it in the following sections.

Though the two problems only differ in whether the length of desirable windows is fixed, they will be addressed in completely different methods in terms of satisfying the quantitative properties. In what follows, we first solve Problem 1 and then proceed to Problem 2. For each problem, we tackle the qualitative requirements before the quantitative ones. We close the discussion of this section with the following example.

Example 1: Consider the weighted automaton G in Fig. 2, with the only unsafe state x_8. The set of controllable events is $E_c = \{a, b, c, d, e, f\}$ and the set of uncontrollable events is $E_{uc} = \{u_1, u_2, u_3, u_4, u_5, u_6\}$. The weight of each event is drawn along with the event in the figure.

Obviously, the run $x_1 \xrightarrow{a} x_2 \xrightarrow{d} x_1 \xrightarrow{a} x_2 \xrightarrow{d} \cdots$ is not a desirable-window infinite run since none of its fragment is a desirable window. If we fix the window size as $N = 3$, then the run $x_1 \xrightarrow{u_2} x_6 \xrightarrow{e} x_7 \xrightarrow{u_3} x_1 \xrightarrow{u_2} x_6 \xrightarrow{e} x_7 \xrightarrow{u_3} \cdots$ is a three-step desirable-window infinite run. However, $x_1 \xrightarrow{b} x_3 \xrightarrow{c} x_4 \xrightarrow{u_4} x_5 \xrightarrow{u_6} x_1 \xrightarrow{b} x_3 \xrightarrow{c} x_4 \xrightarrow{u_4} x_5 \xrightarrow{u_6} \cdots$ is not a three-step desirable-window infinite run as its fragment $x_5 \xrightarrow{u_6} x_1 \xrightarrow{b} x_3 \xrightarrow{c} x_4$ is not a three-step desirable window due to $\omega(u_6) < 0$, $\omega(u_6 b) < 0$ and $\omega(u_6 bc) < 0$. Moreover, unsafe state x_8 is reached under some strings. Hence, supervisory control is necessary to restrict the behaviors of G. We will solve Problems 1 and 2 on G in the remaining sections of the article.

IV. Weighted Bipartite Transition System

In order to solve Problems 1 and 2, we first transform the automaton model in Section II to a two-player game between the supervisor and the system (environment). This section tackles the logical requirements and sets the basis for solving both problems. The weighted bipartite transition system (*WBTS*) is defined as the game graph, then a special WBTS is proposed, which enforces the safety and liveness conditions.

Definition 5 (Weighted Bipartite Transition System): A WBTS with respect to system G is a tuple $T = (Q_Y, Q_Z, E, \Gamma, f_{yz}, f_{zy}, \omega, y_0)$ such that we have the following.

- $Q_Y \subseteq X$ is the set of states where the supervisor plays.
- $Q_Z \subseteq X \times \Gamma$ is the set of states where the environment plays, we let $Sta(z)$ and $Ctr(z)$ denote the two components of $z \in Q_Z$, so $z = (Sta(z), Ctr(z))$.
- E is the set of events.
- Γ is the set of control decisions.
- $f_{yz} : Q_Y \times \Gamma \to Q_Z$ is the transition function from Q_Y states to Q_Z states where for $y \in Q_Y, \gamma \in \Gamma$, and $z \in Q_Z$, we have that $f_{yz}(y, \gamma) = (y, \gamma)$.
- $f_{zy} : Q_Z \times E \to Q_Y$ is the transition function from Q_Z states to Q_Y states where, for $z = (y, \gamma) \in Q_Z$, $e \in E$, and $y' \in Q_Y$, we have that $f_{zy}(z, e) = y' \Leftrightarrow [e \in \gamma] \wedge [y' = f(y, e)]$.
- $\omega : E \to \mathbb{Z}$ is the event weight function inherited from G and labels f_{zy} transitions.
- $y_0 \in Q_Y$ is the initial state and $y_0 = x_0$.

The above concept is inspired by the bipartite transition system defined in [48]. A WBTS T presents a game between the supervisor and the environment. A Q_Y state (Y-state) is where the supervisor plays by making control decisions. Since the supervisor has full observation, Y-states are from the system's state space X. We call a $y \in Q_Y$ *safe* if $y \notin X_{\text{us}}$. A Q_Z state (Z-state) consists of a Y-state plus a control decision, where the environment plays by "selecting" enabled events to occur. A f_{yz} transition is defined from Y-states to Z-states to remember the most recent decision of the supervisor. We use $C_T(y) = \{\gamma \in \Gamma : f_{yz}(y, \gamma)!\}$ to stand for the set of control decisions at $y \in Q_Y$. f_{zy} is defined from Z-states to Y-states which are reachable under the executed events in G. Since the supervisor is unable to choose which event to occur, all enabled events are defined at a Z-state. Essentially, we explicitly separate the processes of making a control decision and executing enabled events in T. Finally, ω is the same weight function inherited from G and labels the events associated with f_{zy}.

Given a WBTS T, a run in T is of the form $r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{\gamma_n} z_n \xrightarrow{e_n} y_{n+1}$. We write $y \in r$ and $z \in r$ if y (respectively z) is a Y-state (respectively Z-state) in r. We also denote by $\text{Run}_y(T)$ (respectively $\text{Run}_z(T)$) the set of runs whose last states are Y-states (respectively Z-states). A run is called *initial* if its first state is the initial state of T. We also denote by $\text{Run}_{\text{inf}}(T)$ as the set of infinite runs in T.

Considering a run r in a WBTS T, we say it *generates* a run $y_1 \xrightarrow{e_1} y_2 \xrightarrow{e_2} \cdots \xrightarrow{e_n} y_{n+1}$ in G when the control decisions and Z-states are removed. By Definition 5 and simple induction, we know that the generated run is in G as $\forall i \geq 1$, $y_i \in X$ and $f(y_i, e_i) = y_{i+1} \in X$. This shows the relation of the game structure model and the automaton model.

Then it is natural to consider the *strategies* for both players in the game. Generally, both players make new decisions based on the history of all previous states and decisions, i.e., runs. In a WBTS T, we define the *supervisor's strategy (control strategy)* as $\pi_s : \text{Run}_y(T) \to \Gamma$ and the *environment's strategy* as $\pi_e : \text{Run}_z(T) \to E$. We denote the set of all supervisor's and environment's strategies by Π_s and Π_e, respectively. A player selects a transition at its position following its strategy.

From a Y-state y in T, if the supervisor plays π_s and the environment plays π_e, a unique run is formed. We define $\text{Run}(\pi_s, y, T) = \{y \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{\gamma_{n-1}} z_{n-1} \xrightarrow{e_{n-1}} y_n : n \in \mathbb{N}^+, \forall i < n, \gamma_i = \pi_s(y \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{\gamma_{i-1}} z_{i-1} \xrightarrow{e_{i-1}} y_i)\}$ as the set of runs starting from y and *consistent* with control strategy π_s, i.e., the control decisions are specified by π_s. Similarly, we define the runs consistent with the environment's strategies.

The f_{yz} transitions in a WBTS reflect the events enabled under control decisions, while the f_{zy} transitions reflect the executions of the enabled events. By Definition 5, a control strategy in T works the same as a standard supervisor in supervisory control theory [6]. In what follows, we will use the terms "supervisor" and "supervisor's strategy (control strategy)" interchangeably. Given a control strategy π_s and string s, we will use notations π_s/G and $\pi_s(s)$ to stand for the supervised system under π_s and the control decision made by π_s on occurrence of s, respectively.

Intuitively, a strategy has *memory* if a player makes different decisions when the same state is visited again; otherwise, it is called *memoryless*. In a WBTS T, a control strategy π_s is of finite memory if it can be encoded as a deterministic finite-state Moore automaton $A_M = (M, \delta_m, \delta_s, m_0)$ where M is the finite set of states representing the memory; $\delta_m : M \times (Q_Y \cup Q_Z) \to M$ is the transition function for memory update; $\delta_s : M \times Q_Y \to Q_Z$ reflects the supervisor's choice of successor states. If the supervisor plays strategy π_s at $y \in Q_Y$ with the current memory $m \in M$, then it chooses $z = \delta_s(m, y)$ as the successor. After the supervisor makes the decision, the memory of its strategy is updated to $m' = \delta_m(m, y)$. Formally, we may extract a control strategy π_s from A_M such that $\pi_s(y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{\gamma_{n-1}} z_{n-1} \xrightarrow{e_{n-1}} y_n) = \gamma_n$, $f_{yz}(y_n, \gamma_n) = z_{n+1} = \delta_s(\delta_m(m_0, y_1 z_1 y_2 \cdots z_{n-1}), y_n)$ where the domain of δ_m can be extended to $(Q_Y \cup Q_Z)^*$ naturally. π_s is *memoryless* if $|M| = 1$, i.e., the supervisor's choice only depends on its current states. The memory of an environment's strategy is characterized analogously. The readers may refer to [3] for more details concerning memory of game strategies.

Given a control strategy π_s in a WBTS T, let a string $s = e_1 e_2 \cdots e_n \in \mathcal{L}(\pi_s/G)$ and the occurrence of s induces a run $r(s) = y_0 \xrightarrow{\pi_s(\epsilon)} z_0 \xrightarrow{e_1} y_2 \xrightarrow{\pi_s(e_1)} z_2 \xrightarrow{e_2} \cdots \xrightarrow{e_n} y_n \xrightarrow{\pi_s(e_1 e_2 \cdots e_n)} z_n$ in T. We denote by $Y(s)$ and $Z(s)$ the last Y-state and Z-state of $r(s)$, respectively. Formally speaking, if π_s is in T, then $\forall s \in \mathcal{L}(\pi_s/G), \pi_s(s) \in C_T(Y(s))$.

Let Q be a set of states in a WBTS T, then the supervisor's *attractor* with respect to Q is defined recursively as

$$
\begin{aligned}
&\text{Attr}^T_{s,0}(Q) = Q \\
&\text{Attr}^T_{s,i+1}(Q) \\
&= \{y \in Q_Y \setminus \text{Attr}^T_{s,i}(Q) : \exists y \xrightarrow{\gamma} z \text{s.t.} z \in \text{Attr}^T_{s,i}(Q)\} \\
&\cup \{z \in Q_Z \setminus \text{Attr}^T_{s,i}(Q) : \forall z \xrightarrow{e} y, y \in \text{Attr}^T_{s,i}(Q)\}
\end{aligned}
$$

$$\mathrm{Attr}_s^T(Q) = \bigcup_{i\geq 0} \mathrm{Attr}_{s,i}^T(Q). \tag{1}$$

By definition, the supervisor reaches Q from $\mathrm{Attr}_{s,i}^T(Q)$ by i events *for sure* regardless of the environment's strategies. Therefore, $\mathrm{Attr}_s^T(Q)$ is the *largest* set of states from which the supervisor is able to reach Q within finitely many transitions regardless of the environment's strategies. On the other hand, the supervisor is unable to reach Q from states outside of $\mathrm{Attr}_s^T(Q)$; otherwise, it contradicts the definition of the attractor. It is well known that the attractor can be computed in linear time, provided the game graph is finite [3]; thus, it takes $\mathrm{O}(n(T))$ to compute $\mathrm{Attr}_s^T(Q)$ where $n(T)$ denotes the number of transitions in T. Note that we only add new states that are not in $\mathrm{Attr}_{s,i}^T(Q)$ in each stage of calculating $\mathrm{Attr}_s^T(Q)$. The environment's attractor with respect to Q is defined analogously and denoted by $\mathrm{Attr}_e^T(Q)$.

Given a WBTS T and a set of states Q in T, we introduce a *rank function* $\sigma : Q_Y \cup Q_Z \to \mathbb{N}$ associating with every state the stage at which it is added to the attractor $\mathrm{Attr}_s^T(Q)$

$$\sigma(q) = \begin{cases} i & \text{if } q \in \mathrm{Attr}_{s,i}^T(Q) \text{ for some } i \geq 0 \\ \infty & \text{if } q \notin \mathrm{Attr}_s^T(Q). \end{cases} \tag{2}$$

Here we define the rank for the supervisor's attractor and the rank for the environment's attractor is defined analogously. Since the attractor is calculated in a finite number of steps, σ is always finite and may be obtained when the attractor is calculated. Intuitively, the rank also reflects the "distance" between a state q and the "destination" Q. A similar and more involved concept was proposed in [37] for product automata.

The smaller the $\sigma(q)$ is, the "closer" the q is to Q and $\sigma(q) = 0$ if $q \in Q$. Accordingly, for any Y-state $q \in \mathrm{Attr}_s^T(Q) \setminus Q$, if the supervisor always makes decisions to reach successor q' with $\sigma(q) > \sigma(q')$, then we claim that the supervisor eventually reaches Q after a finite number of steps. Otherwise, there will be an infinite sequence of states $q, q_1, q_2, \cdots \in \mathrm{Attr}_s^T(Q) \setminus Q$ such that $\sigma(q) > \sigma(q_1) > \sigma(q_2) \cdots$, which is infeasible for finite $\sigma(q)$. This further implies that the supervisor always has a strategy to reach Q from $\mathrm{Attr}_s^T(Q) \setminus Q$, by choosing successor states with a decreasing rank. This observation will play a role in solving Problem 1 in the next section.

Given a WBTS T, a Y-state y is called a *terminal* state if it has no successor states. When there are no active events defined at y in G, the supervisor is unable to make control decisions and y is terminal, i.e., $C_T(y) = \emptyset$. Moreover, T is called *complete* if $\forall y \in Q_Y$, y has successors. In addition, a Z-state z is *terminal* if $\nexists e \in E$, s.t. $f_{zy}(z,e)!$, i.e., the supervisor disables all events. Terminal states should be avoided since they contradict with the liveness requirement: There should always be events defined out of a state in the supervised system. If T is complete, then the supervisors in T are always able to make decisions, resulting in a live supervised system.

For a complete WBTS T, we may explicitly "extract" a unique supervisor from it if we specify a control decision at each Y-state in T. We denote this supervisor by S_T which is *realized* by an automaton $G_T = (Q_Y, E, \xi, y_0)$. Here, $\xi : Q_Y \times E \to Q_Y$ is the transition function such that $\forall y \in Q_Y$, $\forall e \in E$: $\xi(y,e) = f_{zy}(f_{yz}(y,\gamma), e)$ if γ is chosen at y and $e \in \gamma$ and γ is chosen at y. y_0 is the initial Y-state of T. We may compute the language of the supervised system as $\mathcal{L}(S_T/G) = \mathcal{L}(S_T \times G)$ where $\times$ is the standard product operation between automata [6].

Algorithm 1: Build T_m w.r.t. G.

Input : G
Output : $T_m = (Q_Y^m, Q_Z^m, E, \Gamma, f_{yz}^m, f_{zy}^m, \omega, y_0)$ w.r.t. G

1 $Q_Y^m = \{y_0\}$, $Q_Z^m = \emptyset$;
2 $DoDFS(y_0, G)$;
3 **while** *there exist Y-states that have no successor* **do**
4   Remove all such Y-states and their predecessor Z-states, take the accessible part;
5 **return** T_m;
Procedure: $DoDFS(y, G)$
6 **for** $\gamma \in \Gamma$ **do**
7     $z = f_{yz}(y, \gamma)$;
8   **if** $Sta(z) \notin X_{us}$ *and z is not a terminal state* **then**
9   add transition $y \xrightarrow{\gamma} z$ to f_{yz}^m;
10   **if** $z \notin Q_Z^m$ **then**
11             $Q_Z^m = Q_Z^m \cup \{z\}$;
12   **for** $e \in \gamma$ **do**
13                 $y' = f_{zy}(z, e)$;
14   add $z \xrightarrow{e} y'$ to f_{zy}^m, its weight is $\omega(e)$;
15   **if** $y' \notin Q_Y^m$ **then**
16                     $Q_Y^m = Q_Y^m \cup \{y'\}$;
17                     $DoDFS(y', G)$;

Given two WBTSs $T_1 = (Q_Y^1, Q_Z^1, E, \Gamma, f_{yz}^1, f_{zy}^1, \omega^1, y_0^1)$ and $T_2 = (Q_Y^2, Q_Z^2, E, \Gamma, f_{yz}^2, f_{zy}^2, \omega^2, y_0^2)$, we say that T_1 is a *subgame* of T_2, denoted by $T_1 \sqsubseteq T_2$, if $Q_Y^1 \subseteq Q_Y^2$, $Q_Z^1 \subseteq Q_Z^2$ and for all $y \in Q_Y^1, z \in Q_Z^1, \gamma \in \Gamma, e \in E$, we have $f_{yz}^1(y,\gamma) = z \Rightarrow f_{yz}^2(y,\gamma) = z$ and $f_{zy}^1(z,e) = y \Rightarrow f_{zy}^2(z,e) = y$. Here the relation of the two weight functions does not really matter. Given a WBTS T and a set of states $Q \subseteq Q_Y \cup Q_Z$, we denote by $T' = T \downharpoonleft Q$ if $T' \sqsubseteq T$ and Q is the state space of T', i.e., the game on T is restricted to a subgame T' whose state space is Q.

Then we propose Algorithm 1 to construct the maximum complete WBTS without terminal Z-states or unsafe Y-states, with respect to automaton G. It is denoted by $T_m = (Q_Y^m, Q_Z^m, E, \Gamma, f_{yz}^m, f_{zy}^m, \omega, y_0)$. The "maximum" is in the graph merging sense, i.e., for any complete WBTS T without terminal Z-states or unsafe Y-states, we have $T \sqsubseteq T_m$. For simplicity, we denote by $|T_m|$ the number of states in T_m and by n_e the number of edges in T_m.

Algorithm 1 is inspired by the algorithm of constructing the all enforcement structure in [48]. The major difference is that the system in [48] is partially observed, while it is fully observed here; so there is no need to consider unobservable reaches under control decisions in this article. The main idea of Algorithm 1 is to recursively build the state space of T_m in a depth-first search manner until no more states are added. Note that we only include nonterminal Z-states without unsafe state components, as done in line 8. We prune away Y-states without successors as well as their preceding Z-states in line 4 so that the final structure is

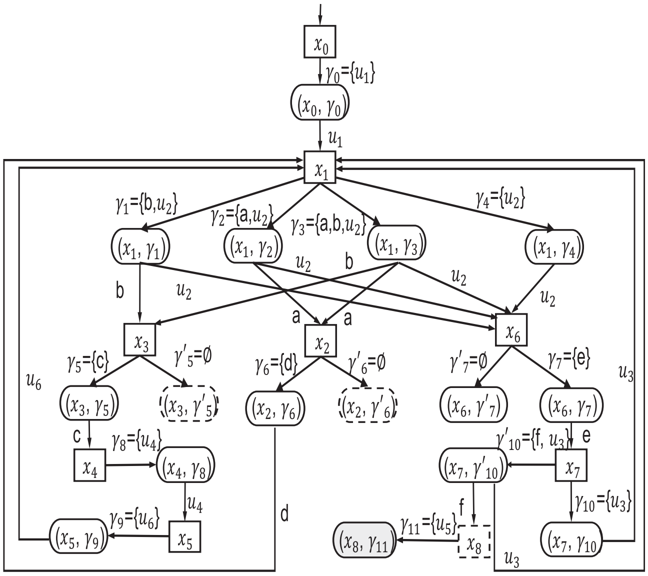


Fig. 3. Resulting structure after DoDFS (without dashed/shaded states).

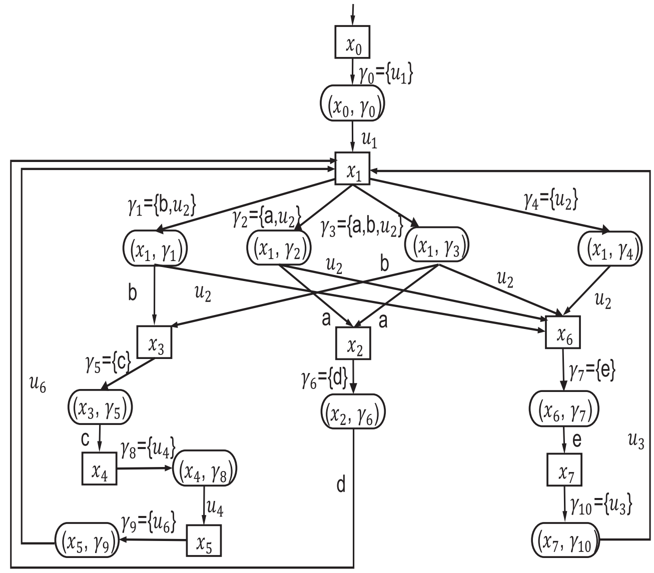


Fig. 4. T_m in Example 2.

complete. Following a similar argument with [48, Theorem V.I], we show the correctness of Algorithm 1 as follows.

Proposition 1: Any control strategy in T_m is safe and live.

Proof: Similar to the proof of [48, Th. V.I] and we just sketch the idea here. By Definition 5, $Sta(z)$ tracks the reachable states under control decision $Ctr(z)$ for $z \in Q_Z^m$. Then by Algorithm 1, if for all $z \in Q_Z^m$, we have $Sta(z) \notin X_{\text{us}}$ and $\exists e \in E$ such that $f_{zy}^m(z, e)!$, i.e., no unsafe states in G are reached and events are always enabled at z, then any control strategy in T_m is always safe and live. ■

Remark 2: We briefly analyze the complexity of Algorithm 1. First, the procedure $DoDFS$ may result in a structure that has, in the worst case, $|X| \cdot 2^{|E_c|} + |X|$ states (Z-states plus Y-states), where $2^{|E_c|}$ is the maximum number of feasible control decisions. The complexity of the pruning process is quadratic in the size of the returned structure after $DoDFS$. Thus, the overall complexity of Algorithm 1 is $\mathrm{O}(|X|^2 \cdot 2^{2|E_c|})$.

So far safety and liveness have been enforced for Problems 1 and 2. Before proceeding to fulfill the quantitative conditions, we end this section with the following example.

Example 2: We revisit Example 1 and build T_m for the system, following Algorithm 1. First, the $DoDFS$ procedure returns the WBTS shown in Fig. 3 . The rectangular states are Y-states while the round rectangular states are Z-states. As is seen, dashed Z-states (x_3, γ_5'), (x_2, γ_6'), and (x_6, γ_7') are not included during the procedure $DoDFS$ at line 8 since they are terminal. The shaded Z-state (x_8, γ_{11}) is not included either (at line 8) since x_8 is an unsafe state. Due to the absence of (x_8, γ_{11}), Y-state x_8 has no successor. After that, Y-state x_8 is removed by the while loop in Algorithm 1, so is (x_7, γ_{10}'); the resulting T_m is shown in Fig. 4 . We may verify that every control strategy in T_m is both safe and live. However, not all of them satisfy the quantitative conditions in Problem 1 or 2; thus, further analysis is necessary to solve the problems.

V. Supervisory Control Under Desirable Windows

With safety and liveness enforced in Section IV, we accomplish the quantitative requirements of Problem 1 in this section. Several objectives are derived and a new game is formulated between the supervisor and the environment. Then we leverage results from total payoff games in the literature to solve the game, which, in turn, solves Problem 1.

To solve Problem 1, the supervisor should only allow runs with desirable windows on G. In parallel, we characterize *n-step desirable windows on T_m* for a given window size $n \in \mathbb{N}^+$

$$Run_d(T_m, n) = \{r \in \text{Run}(T_m) : r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_n} y_{n+1}, \exists \ell \leq n \text{s.t.} \frac{1}{\ell}\sum_{i=1}^{\ell} \omega(e_i) \geq 0\}. \quad (3)$$

When the window size is not given *a priori*, we define the *desirable-window finite runs on T_m* as follows:

$$\text{Run}_d(T_m) = \{r \in \text{Run}(T_m) : r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_n} y_{n+1}, n \in \mathbb{N}^+, \exists \ell \leq n \text{ s.t. } \frac{1}{\ell}\sum_{i=1}^{\ell} \omega(e_i) \geq 0\}. \quad (4)$$

Then it comes to infinite runs in T_m and we introduce the *desirable-window objective* as

$$W_d(T_m) = \{r \in \text{Run}_{\text{inf}}(T_m) : r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots, \exists i \geq 1 \text{s.t.} \forall j \geq i, \exists \ell \geq 1 : \frac{1}{\ell}\sum_{p=0}^{\ell-1} \omega(e_{j+p}) \geq 0\}. \quad (5)$$

Comparing (3) with (4), we find that if desirable-window finite runs are successively formed on an infinite run, then such infinite run is included in $W_d(T_m)$. The supervisor is said to *achieve* $W_d(T_m)$ if it has a strategy π_s such that any infinite run consistent with π_s is in $W_d(T_m)$. In other words, the supervisor perpetually forms desirable-window finite runs on T_m. Correspondingly, we formulate a new game on T_m, where the supervisor wins by achieving the desirable-window objective while the environment wins by preventing the supervisor from achieving it. In fact, infinite runs in $W_d(T_m)$ generate desirable-window infinite runs in the original system G. In what follows, we will study how to achieve $W_d(T_m)$ and show that Problem 1 is solved by supervisors achieving $W_d(T_m)$.

Before proceeding to solve Problem 1, we briefly review the concept of *total payoff games* [4], which is involved in the following analysis. Given a run $r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{\gamma_n} z_n \xrightarrow{e_n} y_{n+1}$ in T_m, its total payoff is $\sum_{i=1}^{n} \omega(e_i)$. Note that the total payoff game is an infinite game where the supervisor wins if it has a strategy to form infinite runs with nonnegative (limit) total payoffs. Then we define the following objective:

$$Tot(T_m) = \{r \in \text{Run}_{\text{inf}}(T_m) : r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots, \\ \limsup_{n\to\infty} \sum_{i=1}^{n} \omega(e_i) \geq 0\}. \tag{6}$$

The supremum in (6) ensures that the limit sum is well defined. Conversely, the environment wins the total payoff game if it has a strategy to prevent the supervisor from achieving (a subset of) $Tot(T_m)$. It is shown in [4] that memoryless strategies are sufficient to win the total payoff game.

We have formulated a game with objective $W_d(T_m)$. Note that our game on T_m with $W_d(T_m)$ may be viewed as a special form of the *bounded window mean payoff game* in [7], where the transitions from the supervisor's states in T_m have zero weight. Results from total payoff games are employed in [7] to calculate the winning regions for bounded window mean payoff games. Since we are dealing with a similar objective, we leverage results from [7] and total payoff games [4] to solve our game. Later on, we also propose an algorithm to synthesize supervisors from the game, which is not discussed in the literature [7]. Two lemmas (i.e.,[7, Lemmas 10 and 11]) are repeated below in our context to establish the connection between a total payoff game with desirable windows; detailed proofs are omitted since they directly follow the lemmas in [7].

Lemma 1: If the supervisor wins for $Tot(T_m)$ from a state in T_m, then it may play the same strategy to form $(|T_m| - 1) \cdot (|T_m| \cdot W + 1)$-step desirable windows from that state.

Suppose π_s is a supervisor's winning strategy for Tot(T_m). The main idea for showing Lemma 1 is to decompose any run consistent with π_s into its acyclic part and cyclic part (some cycles). Then by inspecting the total payoff of run prefixes, we can bound the length of the desirable windows.

Lemma 2: If the environment has a strategy to win the total payoff game from a state in T_m, then it has a strategy π_e to ensure that for every run starting from that state and consistent with π_e, there exists a position in the run such that all suffixes from that position have negative total payoff.

Lemma 2 can be proved by contradiction. The idea is that if this lemma does not hold, then any run consistent with π_e may be decomposed as a sequence of run fragments with a nonnegative total payoff, which implies that the total payoff of the run is also nonnegative; thus, a contradiction with π_e being a winning strategy for the environment.

Algorithm 2: Compute the winning region for $W_d(T_m)$.

Input : T_m
Output : the supervisor's winning region $\mathcal{W}_s^{dw}$ and two sequences of states

1 $n = 0$, $\mathcal{W}_s^{dw}(0) = \emptyset$;
2 $\mathcal{W}^{\text{neg}}(0) = NegWindow(T_m)$;
3 **while** $\mathcal{W}_s^{dw}(n) \neq (Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(n)$ **do**
4   output $\mathcal{W}^{\text{neg}}(n)$;
5   $\mathcal{W}_s^{dw}(n+1) \leftarrow Attr_s^{T_m}((Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(n))$;
6   output $\mathcal{W}_s^{dw}(n+1)$;
7   **for** $q \in \mathcal{W}_s^{dw}(n+1) \setminus \mathcal{W}_s^{dw}(n)$ **do**
8   Label q with $\sigma(q) = n+1$;
9   $\mathcal{W}^{\text{neg}}(n+1) \leftarrow NegWindow(T_m \downharpoonright ((Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}_s^{dw}(n+1)))$;
10   $n \leftarrow n+1$;
11 Return $\mathcal{W}_s^{dw} = \mathcal{W}_s^{dw}(n)$;
Procedure: $NegWindow(T)$
12 $\ell = 0$, $\mathcal{W}_0^{\text{neg}} = \emptyset$
13 **repeat**
14   $\mathcal{W}_{\ell+1}^{\text{neg}} = \mathcal{W}_\ell^{\text{neg}} \cup Attr_e^{T \downharpoonright (Q_Y^m \cup Q_Z^m \setminus \mathcal{W}_\ell^{\text{neg}})}(NegTotal(T \downharpoonright (Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}_\ell^{\text{neg}})))$;
15   $\ell \leftarrow \ell + 1$;
16 **until** $\mathcal{W}_\ell^{\text{neg}} = \mathcal{W}_{\ell-1}^{\text{neg}}$;
17 Return $\mathcal{W}^{\text{neg}} = \mathcal{W}_\ell^{\text{neg}}$;

We term the set of states where the supervisor achieves $W_d(T_m)$ as the (supervisor's) *winning region* for $W_d(T_m)$ and denote it by $\mathcal{W}_s^{dw}$. Based on Lemmas 1 and 2, we slightly adapt [7, Algorithms 4 and 5] to present our Algorithm 2 for computing $\mathcal{W}_s^{dw}$. Specifically, we introduce an index I to label states in $\mathcal{W}_s^{dw}$, which reflects when a state is added to $\mathcal{W}_s^{dw}$ and plays a role in supervisor synthesis later on.

Algorithm 2 computes the set of states where the environment forms undesirable windows via the procedure *NegWindow*. Since the desirable-window objective does not depend on the prefixes of runs, the environment should repeatedly form undesirable windows to force the supervisor to lose the game. We denote by $\mathcal{W}^{\text{neg}}$ the set of states where the environment forms infinite runs with negative total payoff; thus, $W_d(T_m)$ is violated. Obviously, the supervisor should avoid states returned by *NegWindow*. In other words, states not in $NegWindow(T_m)$ and their attractor states contribute to the supervisor's winning region $\mathcal{W}_s^{dw}$. At the beginning, we assume that no state is winning for the supervisor in line 1. Then we continually reduce the environment's potential choices which prevent the supervisor from achieving the desirable-window objective and $\mathcal{W}^{\text{neg}}$ may be shrunk each time $NegTotal(T_m)$ is called. As a result, more states are declared winning for the supervisor by recursively calling $NegTotal(T_m)$ and calculating the attractor of those states, until no new states are added to the winning region $\mathcal{W}_s^{dw}$. This is essentially the computation of a fixed point. In line 8, we index each new state by the first time it is added to $\mathcal{W}_s^{dw}$.

Now we take a closer look at the procedure *NegWindow*. By Lemma 2, it suffices to compute the environment's attractor for the set of states from which the environment achieves a negative total payoff. The routine *NegTotal* calls the *pseudopolynomial value iteration method* developed in [4] (Algorithm 2 and the strategy mentioned in Section 4.3) and returns the states where the environment wins the total payoff game on the current game graph. The idea of the leveraged algorithm is to proceed through nested fixed points and the technical details concerning the algorithm are omitted here for simplicity.

Remark 3: We briefly discuss the complexity of Algorithm 2. Here, we denote by n_e the number of edges in T_m and $\mathbb{O}$ the complexity of procedure $NegTotal$, i.e., the complexity of solving a total payoff game. First, the complexity for procedure $NegWindow$ is $\mathrm{O}(|T_m| \cdot (n_e + \mathbb{O}))$ since we take at most $|T_m|$ times of computation and each computation takes $(n_e + \mathbb{O})$. Then the overall complexity of Algorithm 2 is $\mathrm{O}(\mathbb{O} + |T_m| \cdot (n_e + |T_m| \cdot (n_e + \mathbb{O}))) = \mathrm{O}(|T_m|^2 \cdot (n_e + \mathbb{O})))$. A software tool called ***PRISM-games*** developed in [23] efficiently solves total payoff games, which also helps us to implement Algorithm 2.

The correctness of Algorithm 2 in computing $\mathcal{W}_s^{dw}$ is shown similarly to [7, Algorithms 4 and 5] and the proof is omitted here. The main idea is that the environment prevents the supervisor from winning for $W_d(T_m)$ by denying a nonnegative total payoff from states not in $\mathcal{W}_s^{dw}$. So the desirable-window objective is never achieved from those states, while the supervisor wins the game from $\mathcal{W}_s^{dw}$.

By running Algorithm 2, we collect two sequences of states

$$[\mathcal{W}^{\text{neg}}] = \mathcal{W}^{\text{neg}}(0), \mathcal{W}^{\text{neg}}(1), \ldots, \mathcal{W}^{\text{neg}}(n) \quad (7)$$

$$[\mathcal{W}_s^{dw}] = \mathcal{W}_s^{dw}(0), W^{\text{neg}}(1), \ldots, W_s^{dw}(n). \quad (8)$$

As T_m is finite and Algorithm 2 always terminates, both sequences are finite. Interestingly, they form two "chains": $\mathcal{W}_s^{dw}(i) \subseteq \mathcal{W}_s^{dw}(j)$ and $\mathcal{W}^{\text{neg}}(j) \subseteq \mathcal{W}^{\text{neg}}(i)$ for any $i, j, \ 0 \leq i < j \leq n$. Also we know that $\mathcal{W}^{\text{neg}}(0) = \mathrm{NegWindow}(T_m)$, $\mathcal{W}_s^{dw}(n) = \mathcal{W}_s^{dw}$ and $\mathcal{W}_s^{dw}(k) = \mathrm{Attr}_s^{T_m}((Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1))$ for every $1 \leq k \leq n$ by Algorithm 2. When the supervisor's winning region $\mathcal{W}_s^{dw}$ is not empty and the initial state of T_m is included in $\mathcal{W}_s^{dw}$, we denote by $T_{win}^{dw} = T_m \downharpoonleft \mathcal{W}_s^{dw}$, whose state space is $\mathcal{W}_s^{dw}$. Otherwise, we let T_{win}^{dw} be empty. When T_{win}^{dw} is not empty, Algorithm 3 is presented below to synthesize a supervisor that wins the game and achieves $W_d(T_m)$.

Algorithm 3: Synthesize a Supervisor that Achieves the Desirable-Window Objective.

Input : T_m, $[\mathcal{W}^{\text{neg}}]$, $[\mathcal{W}_s^{dw}]$
Output : a supervisor achieving $W_d(T_m)$
1 $Syn(y_0, T_m)$;
2 Extract the supervisor from the remaining subgame;
Procedure: $Syn(y, T_m)$
3 **if** $y \in \mathcal{W}_s^{dw} \cap Q_Y^m$ *has not be specified a control decision in the current structure* **then**
4 suppose $\sigma(y) = k$, so $y \in \mathcal{W}_s^{dw}(k)$;
5 **if** $y \in Attr_s^{T_m}((Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1))$ *and* $y \notin \setminus((Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1))$ **then**
6 choose $\gamma \in C_{T_m}(y)$ such that $\sigma(y) > \sigma(f_{yz}^m(y,\gamma))$ and $f_{yz}^m(y,\gamma) \in Attr_s^{T_m}((Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1))$;
7 **if** $y \in (Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1)$ **then**
8 apply the method developed in [4] to solve a total payoff game on the subgame $T_{win}^{dw}|(Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1)$ and determine the decision γ specified at y in the total payoff game;
9 remove all control decisions defined at y except γ, and take the accessible part;
10 $z = f_{yz}^m(y,\gamma)$;
11 **for** $e \in \gamma$ **do**
12 $Syn(f_{zy}^m(z,e), T_m)$;

Intuitively, Algorithm 3 specifies a control decision at each Y-state in $\mathcal{W}_s^{dw}$ to lead the supervisor to states where it can perpetually achieve desirable-window runs. More specifically, every Y-state belongs to some $\mathcal{W}_s^{dw}(k)$ and two cases are categorized. First, if the current state y is not yet in $((Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1))$, then in line 6, the supervisor makes a decision to reach a successor that has a lower rank and is in $\mathrm{Attr}_s^{T_m}((Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1))$. Note that by the definition of the attractor [see (1)] and the discussion in Section IV, γ will contribute to leading the supervisor toward $(Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1)$ ultimately. On the other hand, if the supervisor is already in $((Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1))$, then it chooses the decision specified by solving the total payoff game through the method in [4] (Algorithm 2 and Section 4.3), as in line 8. The supervisor follows the decision as it is playing the total payoff game. Since the environment wins the total payoff game from $\mathcal{W}^{\text{neg}}(k-1)$ by Algorithm 2, the supervisor wins the total payoff game from $(Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1)$. Also by Lemma 1, the supervisor may play the same strategy winning the total payoff game to achieve $W_d(T_m)$. Therefore, the control decision γ specified at line 8 contributes to achieving $W_d(T_m)$. Since Procedure Syn runs on the attractor of the supervisor, the environment is unable to force the supervisor out of $\mathcal{W}_s^{dw}$. Syn is recursively called until a control decision is specified at every Y-state. Finally a supervisor is returned in line 2, which is memoryless since making a decision following the attractor requires no memory, as shown in [3], and it is sufficient to win a total payoff game by memoryless strategies, following [4].

Theorem 1: There exists a supervisor that solves Problem 1 if and only if T_{win}^{dw} is not empty.

Proof: ("if") When T_{win}^{dw} is not empty, we denote by S a supervisor returned by Algorithm 3. Then for any Y-state $y \in \mathcal{W}_s^{dw}$, we know that there exists $1 \leq k \leq n$ such that $y \in \mathcal{W}_s^{dw}(k) = \mathrm{Attr}_s^{T_m}((Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}^{\text{neg}}(k-1))$. Then for any infinite run r starting from y and consistent with S in T_m, we have that $r \in W_d(T_m)$ by Algorithm 3. Thus, the run generated by r in the supervised system S/G is a desirable-window infinite run, which implies that S solves Problem 1.

("only if") We show it by contrapositive. When T_{win}^{dw} is empty, we know that in T_m, for any control strategy π_s, there exists an initial run r consistent with π_s such that $r \notin W_d(T_m)$. This further implies that the run generated by r in the supervised system under π_s is not a desirable-window infinite run. That

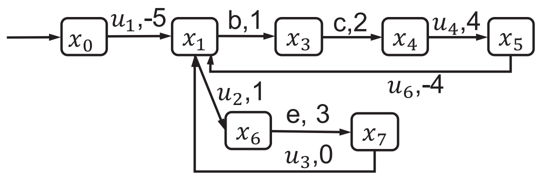


Fig. 5. Supervisor solving Problem 1.

is, no matter what strategy the supervisor plays, the second condition in Problem 1 is not satisfied. Therefore, there does not exist a supervisor solving Problem 1. ∎

Theorem 1 shows the correctness and completeness of our method to solve Problem 1. We are always able to synthesize a supervisor provided that T^{dw}_{win} not empty. At the end of this section, we present an example to illustrate the process of computing the winning region and synthesizing supervisors.

Example 3: We continue Example 2 to completely solve Problem 1 for the system G in Example 1. First, it is easily seen that the run $x_1 \xrightarrow{a} x_2 \xrightarrow{d} x_1 \xrightarrow{a} \cdots$ in Fig. 2 is not a desirable-window infinite run since $\omega(ad) < 0$; thus, supervisory control is necessary to restrict the behaviors of G.

Based on the intermediate results in Example 2, we run Algorithm 2, which returns the supervisor's winning region $\mathcal{W}^{dw}_s$ exactly the state space of T_m in Fig. 4. In other words, the supervisor achieves the desirable-window objective from every state in Fig. 4; thus, we are flexible for supervisor synthesis. Next, we run Algorithm 3 and choose control decision γ_1 at x_1. The resulting supervisor S is extracted from T_m following the argument in Section IV and it is shown in Fig. 5 . We may verify that every infinite run in S/G is a desirable-window infinite run; so S correctly solves Problem 1.

VI. Supervisory Control Under N-Step Desirable Windows

After solving Problem 1, we investigate how to synthesize supervisors for Problem 2 in this section. For this purpose, we first transform the local mean payoff condition in Problem 2 to a properly defined objective for the supervisor on T_m obtained in Section IV. Correspondingly, a new two-player game is formulated and then analyzed. Finally, we characterize the supervisor's winning region for the game and obtain its winning strategies, which completely solves Problem 2.

A. Compute the Supervisor's Winning Region

In T_m, we define the *N-step desirable-window objective* $W_d(T_m, N)$ for both players as

$$W_d(T_m, N) = \{r \in \text{Run}(T_m) : r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots,$$
$$\exists i \geq 1 \text{s.t.} \forall j \geq i, \exists \ell \leq N, \frac{1}{\ell}\sum_{p=0}^{\ell-1} \omega(e_{j+p}) \geq 0\}. \quad (9)$$

Then we form a new game on T_m where the supervisor wins by achieving $W_d(T_m, N)$, which implies that the supervisor perpetually forms runs in $\text{Run}_d(T_m, N)$; see (3). Note that runs in $W_d(T_m, N)$ generate N-step desirable-window infinite runs. So if the supervisor achieves $W_d(T_m, N)$, then it also solves Problem 2. To further evaluate $W_d(T_m, N)$, we introduce the *window payoff functions* in T_m.

Definition 6 (Window Payoff Functions): In T_m with window size $N \in \mathbb{N}^+$, for $0 \leq i \leq N$, define the window payoff function recursively as $h_i : Q^m_Y \cup Q^m_Z \to \mathbb{Z}$ where

$$\forall q \in Q^m_Y \cup Q^m_Z : h_0(q) = 0$$
$$\forall q \in Q^m_Y, \forall 1 \leq i \leq N : h_i(q) = \max_{z \in Q^m_Z, \gamma \in \Gamma}\{h_i(z) : f^m_{yz}(q, \gamma) = z\}$$
$$\forall q \in Q^m_Z, \forall 1 \leq i \leq N : h_i(q) = \min_{y \in Q^m_Y, e \in E}\{\omega(e) + h_{i-1}(y) : f^m_{zy}(q, e) = y\}.$$

The window payoff functions track the best worst-case total weights that the supervisor may achieve from a state in T_m within at most N event occurrences. The supervisor aims to achieve a nonnegative total payoff (also mean payoff) within the next N enabled events, while the environment aims to spoil that goal by achieving a negative payoff. If the current state q is a Y-state (supervisor's position), we maximize the value of $h_i(q)$ for each $1 \leq i \leq N$ by choosing successor states. Note that we do not increase the index i since an f^m_{yz} transition corresponds to a control decision but not an event occurrence. Otherwise, if q is a Z-state (environment's position), we minimize the total payoff to-go so as to calculate $h_i(q)$, where we increase the index as an f^m_{zy} transition indicates one event occurrence. This "min–max" way of defining $h_i(q)$ is due to calculating the worst possible sum of weights after the occurrence of enabled events, and choosing the best possible sum of weights for the supervisor to achieve $W_d(T_m, N)$. By definition, the value of $h_i(q)$ depends on the values of the window payoff functions for the successor states of q. Therefore, we are able to track a run from q in T_m, whose control decisions and i event occurrences lead to $h_i(q)$.

If a state q in T_m is with $h_i(q) \geq 0$ for some $1 \leq i \leq N$, then there is an N-step desirable window [see (3)] starting from q. Therefore, the supervisor achieves $W_d(T_m, N)$ by reaching such states infinitely often and the environment should prevent the supervisor from doing so. Thus, both players are playing a *Büchi*-like game [3]. The determinacy of Büchi games [3] states that only one player wins the game from each state on the game graph. We denote by $\mathcal{W}^{ndw}_s$ the set of states where the supervisor wins the game for $W_d(T_m, N)$, termed winning region. For the supervisor, a state in $\mathcal{W}^{ndw}_s$ is called *winning*, while a state not in $\mathcal{W}^{ndw}_s$ is called *losing*. The complement of $\mathcal{W}^{ndw}_s$ is the environment's winning region for preventing the supervisor from achieving $W_d(T_m, N)$.

Compared with conventional Büchi games [3], we need to ensure that states in $\{q \in Q^m_Y \cup Q^m_Z : h_i(q) \geq 0 \text{ for } 1 \leq i \leq N\}$ are not only reached infinitely often but also *consecutively*. This is due to (9) where a nonnegative weight sum should be enforced repeatedly without any break. For this reason, Algorithm 4 is proposed to recursively compute the supervisor's winning region for $W_d(T_m, N)$. It generalizes the standard divide-and-conquer algorithm for solving Büchi games [3].

Algorithm 4: Compute the Supervisor's Winning Region for the N-step Desirable-Window objective.

Input : T_m, N
Output : the supervisor's winning region $\mathcal{W}_s^{ndw}$
1 $\mathcal{W}_s^{ndw} = \emptyset$, $n = 1$, $W_p^0 = Q_Y^m \cup Q_Z^m$;
2 **while** $\mathcal{W}_s^{ndw} \neq Q_Y^m \cup Q_Z^m$ *and* $\mathcal{W}_p^{n-1} \neq \emptyset$ **do**
3 $\quad \mathcal{W}_p^n = WinLocal(T_m, N)$;
4 $\quad \mathcal{W}_{attr}^n = Attr_s^{T_m}(\mathcal{W}_p^n)$, $\mathcal{W}_s^{ndw} \leftarrow \mathcal{W}_s^{ndw} \cup \mathcal{W}_{attr}^n$;
5 $\quad T_m \leftarrow T_m \downharpoonright [(Q_Y^m \cup Q_Z^m) \setminus \mathcal{W}_s]$, $n = n + 1$;
6 Return $\mathcal{W}_s^{nds}$;
Procedure: $WinLocal(T_m, N)$
7 $\mathcal{W}_g = StableWindow(T_m, N)$;
8 **if** $\mathcal{W}_g = Q_Y^m \cup Q_Z^m$ *or* $\mathcal{W}_g = \emptyset$ **then**
9 $\quad \mathcal{W}_p = \mathcal{W}_g$;
10 **else**
11 $\quad T_m \leftarrow T_m \downharpoonright \mathcal{W}_g$, $\mathcal{W}_p = WinLocal(T_m, N)$;
12 **return** $\mathcal{W}_p$;
Procedure: $StableWindow(T_m, N)$
13 **for** $q \in Q_Y^m \cup Q_Z^m$ *in the current structure* **do**
14 $\quad h_0(q) = 0$;
15 **for** $i = 1 : N$ **do**
16 $\quad$ **for** $q \in Q_Z^m$ **do**
17 $\quad\quad$ calculate $h_i(q)$ by Definition 6;
18 $\quad$ **for** $q \in Q_Y^m$ **do**
19 $\quad\quad$ calculate $h_i(q)$ by Definition 6;
20 **return** $\mathcal{W}_g = \{q \in Q_Y^m \cup Q_Z^m : \exists 1 \leq i \leq N \text{ s.t. } h_i(q) \geq 0\}$;

Initially at line 1, each state in T_m is viewed as a potentially losing state for the supervisor. In line 3, we call procedure $WinLocal$ to compute state set $\mathcal{W}_p^n$ from which the supervisor achieves $W_d(T_m, N)$. Then in line 4, we add new winning states to the supervisor's winning region $\mathcal{W}_s$. Since $W_d(T_m, N)$ does not depend on the finite prefixes of runs consistent with the supervisor's strategies, if the supervisor is winning from $\mathcal{W}_p^n$, it also wins from the attractor of $\mathcal{W}_p^n$, i.e., $\mathcal{W}_{\text{attr}}^n$ calculated in line 4. Hence, the environment must avoid entering $\mathcal{W}_{\text{attr}}^n$ and remain in the subgame described by line 5 to preserve the chance of winning the game. States removed in line 5 may be viewed as the increment of the supervisor's winning region at each iteration. After that, we iterate on the remaining subgame and call again procedure $WinLocal$ to find more winning states for the supervisor; note that T_m gets updated in lines 5 and 11. In this manner, if the supervisor wins the game for $W_d(T_m, N)$ from a state in $\mathcal{W}_s^{ndw}$, then it also does so from all its successor states, which are contained in $\mathcal{W}_s^{ndw}$ as well. Algorithm 4 essentially computes the greatest fixed point. When it terminates, the states not in $\mathcal{W}_s^{ndw}$ are where the environment can falsify the N-step desirable window objective.

Procedure *StableWindow* computes values of window payoff functions for each state in the current game structure and returns $\mathcal{W}_g$ in line 20. Then the supervisor plays the strategy prescribed by $h_i(q) \geq 0$ (following the decisions leading to $h_i(q)$) to ensure a nonnegative sum of weights within N event occurrences from its current state. In general, the supervisor has memory as it needs to "remember" how $h_i(q) \geq 0$ is achieved from state q each time it makes a decision, and it suffices to record at most N states, which is shown in the next subsection.

Theorem 2: Algorithm 4 correctly computes the supervisor's winning region for $W_d(T_m, N)$.

Proof: Let $\mathcal{W}_s^{ndw}$ be the set of states where the supervisor achieves $W_d(T_m, N)$. We show that a state q is returned by Algorithm 4 if and only if $q \in \mathcal{W}_s^{ndw}$. That is, there exists a control strategy $\pi_s \in \Pi_s$ such that for all $\pi_e \in \Pi_e$, the run from q and generated under (π_s, π_e) is in $W_d(T_m, N)$.

"Only if": Algorithm 4 returns $\cup_{n \geq 0} \mathcal{W}_{\text{attr}}^n$. By the definition of attractor and Algorithm 4, we have that $\mathcal{W}_{\text{attr}}^i \cap \mathcal{W}_{\text{attr}}^j = \emptyset$ and $\mathcal{W}_p^i \cap \mathcal{W}_p^j = \emptyset$ for any $i \neq j$. Let $q \in \cup_{n \geq 0} \mathcal{W}_{\text{attr}}^n$, then there exists a unique n such that $q \in \mathcal{W}_{\text{attr}}^n$. By construction, the supervisor has a strategy to reach and stay in $\mathcal{W}_p^n \cup \mathcal{W}_{\text{attr}}^{n-1} \cdots \cup \mathcal{W}_{\text{attr}}^0$ forever afterwards. Since the runs consistent with the supervisor's strategy are infinite, the supervisor will eventually enter some $\mathcal{W}_p^l$, $0 \leq \ell \leq n$. After that, the supervisor always forms nonnegative weight sums within N event occurrences and, thus, achieves $W_d(T_m, N)$ which further implies that $q \in \mathcal{W}_s^{ndw}$.

"If": Suppose that $q \in \mathcal{W}_s^{ndw}$; we show that $q \in \cup_{n \geq 0} \mathcal{W}_{\text{attr}}^n$ by contradiction. If $q \notin \cup_{n \geq 0} \mathcal{W}_{\text{attr}}^n$, then the environment always has a strategy from q to avoid reaching $\cup_{n \geq 0} \mathcal{W}_{\text{attr}}^n$ and, thus, spoils $W_d(T_m, N)$. That is, from any run starting from q, there exist some states along it, whose window payoff functions are negative for all $1 \leq i \leq N$. So the environment may remain outside $\cup_{n \geq 0} \mathcal{W}_{\text{attr}}^n$ and visit such states infinitely often to prevent the supervisor from achieving $W_d(T_m, N)$. However, this means that the supervisor fails to achieve $W_d(T_m, N)$ from q, which contradicts with $q \in \mathcal{W}_s^{ndw}$. ■

By Theorem 2, if the supervisor's winning region $\mathcal{W}_s^{ndw}$ is not empty and the initial state of T_m is included in $\mathcal{W}_s^{ndw}$, then the supervisor has strategies to win the game and achieve $W_d(T_m, N)$ from the initial state. If this is the case, then we denote by $T_{\text{win}}^{ndw} = T_m \downharpoonright \mathcal{W}_s^{ndw}$, whose state space is $\mathcal{W}_s^{ndw}$. Otherwise, we let T_{win}^{ndw} be empty.

Theorem 3: There exists a supervisor that solves Problem 2 if and only if T_{win}^{ndw} is not empty.

Proof: ("If") When T_{win}^{ndw} is not empty, we know that from any state in T_{win}^{ndw}, there exists a control strategy π_s, such that for all environment's strategy π_e, the run starting from the state and consistent with (π_s, π_e) is in $W_d(T_m, N)$. Therefore, an N-step desirable-window infinite run is generated by $r(\pi_s, \pi_e)$ in π_s/G, which implies that π_s solves Problem 2.

("only if") We show it by contrapositive. When T_{win}^{ndw} is empty, then by Theorem 2, we know that for any control strategy π_s, there exists an initial run r consistent with π_s in T_m such that $r \notin W_d(T_m, N)$. This further implies that the run generated by r in the supervised system is not a N-step desirable-window infinite run. So regardless of the supervisor's strategy, there exist runs in the supervised system that violate the second condition of Problem 2, which means that there does not exist a supervisor solving Problem 2. ■

Up till now, we have shown the soundness and completeness of Algorithm 4 for computing the supervisor's winning region. We will discuss supervisor synthesis on T_{win}^{ndw} in the next subsection if T_{win}^{ndw} is not empty.

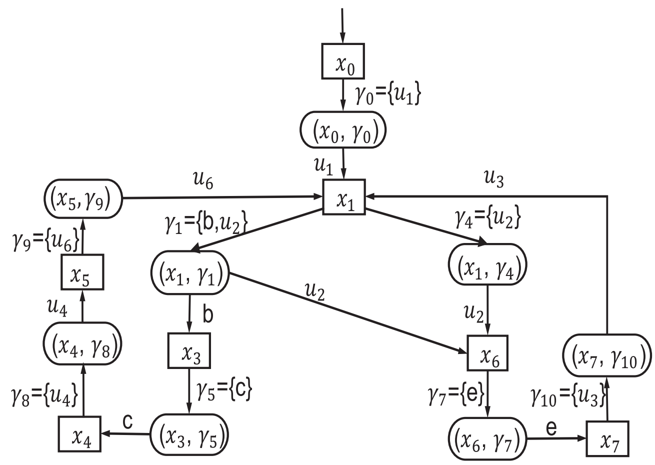


Fig. 6. T_{win}^{ndw} with the supervisor's winning region in Example 4.

Remark 4: We briefly discuss the complexity of Algorithm 4. First, since each edge of the game graph is visited at most N times to compute window payoff functions, the complexity of procedure $StableWindow$ is the procedure $\mathrm{O}(n_e \cdot N)$. Then in procedure W$inLocal$, we call the procedure $StableWindow$ for at most $|T_m|$ times; so its complexity is $\mathrm{O}(|T_m| \cdot n_e \cdot N)$. Finally, we call procedure W$inLocal$ for at most $|T_m|$ times in Algorithm 4 and computing the attractor is linear in n_e. Therefore, the total (worst case) complexity of the algorithm is $\mathrm{O}(|T_m| \cdot (n_e + |T_m| \cdot n_e \cdot N))) = \mathrm{O}(|T_m|^2 \cdot n_e \cdot N)$.

Example 4: We continue Example 2 and solve Problem 2 where we set the window size $N = 3$. Based on the game graph constructed in Example 2, we follow Algorithm 4 to compute the winning region of the supervisor for $W_d(T_m, N)$. First, we calculate the values of window payoff functions for each state in T_m and the results are shown as follows. For simplicity, we associate a four-dimensional vector with each state $q \in Q_Y^m \cup Q_Z^m$ and the elements in the vector are $h_0(q)$ through $h_3(q)$. $x_0 : [0, -5, -4, -1], (x_0, \gamma_0) : [0, -5, -4, -1], x_1 : [0, 1, 4, 4]$, $(x_1, \gamma_1) : [0, 1, 3, 4]$, $(x_1, \gamma_2) : [0, -1, -6, -5]$, $(x_1, \gamma_3) : [0, -1, -6, -5]$, $(x_1, \gamma_4) : [0, 1, 4, 4]$, $x_2 : [0, -5, -4, -1]$, $(x_2, \gamma_6) : [0, -5, -4, -1]$, $x_3 : [0, 2, 6, 2]$, $(x_3, \gamma_5) : [0, 2, 6, 2]$, $x_4 : [0, 4, 0, 1]$, $(x_4, \gamma_8) : [0, 4, 0, 1]$, $x_5 : [0, -4, -3, 0]$, $(x_5, \gamma_9) : [0, -4, -3, 0]$, $x_6 : [0, 3, 3, 4]$, $(x_6, \gamma_7) : [0, 3, 3, 4]$, $x_7 : [0, 0, 1, 4]$, and $(x_7, \gamma_{10}) : [0, 0, 1, 4]$.

After one iteration of procedure $StableWindow$, states (x_1, γ_2), (x_1, γ_3), x_2, and (x_2, γ_6) are not in $\mathcal{W}_g$ since the values of their window payoff functions are negative for all $i \geq 1$. All states reachable from x_1 in Fig. 6 are returned by $StableWindow$ and are, thus, included in $\mathcal{W}_p$ after procedure $WinLocal$. Although both x_0 and (x_0, γ_0) have negative h_i for all $i \geq 1$, they are still included in $\mathcal{W}_s^{ndw}$ since they are in the supervisor's attractor of x_1. Note that $\mathcal{W}_s^{ndw} = Attr_s^{T_m}(\mathcal{W}_s^{ndw})$ in this example. Finally, T_{win}^{ndw} is shown in Fig. 6 whose state space constitutes the supervisor's winning region in this example.

B. Synthesize Winning Supervisors

We proceed to discuss supervisor synthesis in this subsection. The counterpart of Algorithm 3 is proposed, which is more complicated due to the memory of supervisors solving Problem 2. At the beginning, we define *first desirable-window decision sequences* to characterize how the supervisor achieves a nonnegative weight sum within the next N event occurrences from the current Y-state. Here, we denote by $\mathcal{W}_{\text{local}} = \cup_{n \geq 0} \mathcal{W}_p^n$ the union of each $\mathcal{W}_p^n$ obtained from line 3 of Algorithm 4.

Definition 7 (First Desirable-Window Decision Sequences): In T_{win}^{ndw}, at Y-state $y \in \mathcal{W}_{local}$, a sequence of control decisions $\gamma_1\gamma_2 \cdots \gamma_j \in \Gamma^*$ with $j \leq N$ forms a desirable-window decision sequence if there exists a run $r = y \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{\gamma_j} z_j \xrightarrow{e_j} y_j$ such that $\sum_{k=1}^{j} \omega(e_k) = h_j(y)$ where $j = min\{1 \leq i \leq N : h_i(y) \geq 0\}$.

A supervisor achieves the N-step desirable-window objective $W_d(T_m, N)$ in two steps. First, it issues decisions to reach a state in $\mathcal{W}_{\text{local}}$. Then it may repeatedly play strategies prescribed by *StableWindow* in Algorithm 4 to perpetually ensure a nonnegative weight sum within N event occurrences. To be more specific, at some Y-state $y \in \mathcal{W}_{\text{local}}$, there should exist a first desirable-window decision sequence so that the supervisor achieves a nonnegative sum; otherwise, it contradicts with $y \in \mathcal{W}_{\text{local}}$. Then due to the inductive property (Remark 1 in Section III), the supervisor may play another first desirable-window decision sequence from y_j described in Definition 7, and it continues in this manner afterwards. In the above process, the supervisor keeps a memory bounded by N at each Y-state, which reflects how it selects successor states (control decisions) to achieve a nonnegative weight sum. The memory may be reset immediately after a nonnegative weight sum is achieved within the next N event occurrences.

Consequently, we "unfold" the WBTS and introduce the *extended weighted bipartite transition system (EWBTS)* w.r.t. a WBTS T as a tuple: $T_E = (Q_Y^E, Q_Z^E, E, \Gamma, f_{yz}^e, f_{zy}^e, \delta, \omega, y_0^e)$. Here, we have $Q_Y^E = Q_Y \times \mathbb{N}$ and $Q_Z^E = Q_Z \times \mathbb{N}$. With a slight abuse of notation, we also call Q_Y^E-states as Y-states and Q_Z^E-states as Z-states. $f_{yz}^e : Q_Y^E \times \Gamma \to Q_Z^E$ and $f_{zy}^e : Q_Z^E \times E \to Q_Y^E$ are the transition functions. Specifically, $f_{yz}^e((y, n), \gamma)$ (respectively $f_{zy}^e((z, n), e))$ is of the form $(f_{yz}(y, \gamma), \delta(y, n, \gamma))$ (respectively $((f_{zy}(z, e), \delta(z, n, e))$, where $\xi : (Q_Y^E \cup Q_Z^E) \times \mathbb{N} \times (\Gamma \cup E) \to \mathbb{N}$ is some function that updates the integer component of the states. The exact form of δ is left unspecified here and will be defined when we introduce a special EWBTS. $y_0^e = (y_0, 0)$ is the initial state. T_E also describes a game between the supervisor and the environment; thus, the strategies for both players are defined analogously. Similarly with the WBTS, we say that T_E is *complete* if $\forall (y, n) \in Q_Y^E, C_{T_E}((y, n)) \neq \emptyset$.

From the definition of the EWBTS, if we restrict the domains of f_{yz}^e and f_{zy}^e to Q_Y and Q_Z, respectively, then they are reduced to f_{yz} and f_{zy} in a WBTS, respectively. However, function δ has not been defined yet and it is left to count the number of times that a state in the WBTS is revisited when the game graph is unfolded. Then we introduce the *unfolded weighted bipartite transition* *system (UWBTS)* as follows. For simplicity, we write $(y,n) \in Q_Y^E$ as y^n and $(z,n) \in Q_Z^E$ as z^n. Given a state q^e in a EWBTS T_E, we let $\mathrm{Pre}_Y^{T_E}(q^e)$ and $\mathrm{Pre}_Z^{T_E}(q^e)$ denote, respectively, the set of Y-states and the set of Z-states that may reach q^e, excluding q^e itself. Here, we also let $|\cdot|$ be cardinality of a set.

Definition 8 (Unfolded Weighted Bipartite Transition System): A UWBTS is an EWBTS of a complete WBTS T. It is a tuple $U = (Q_Y^U, Q_Z^U, E, \Gamma, f_{yz}^u, f_{zy}^u, \delta_u, \omega, y_0^u)$ where 1) $\forall y^n \in Q_Y^U$:$|C_U(y^n)| = 1$; 2) $\forall z^n \in Q_Z^U, \forall e \in E$: $f_{zy}(z,e)! \Leftrightarrow f_{zy}^u(z^n,e)!$; 3) $\forall y^n \in Q_Y^U$: $n = |\{y^{\tilde{n}} \in Pre_Y^U(y^n) : \tilde{n} \in \mathbb{N}\}|$ and $\forall z^{n'} \in Q_Z^U$: $n' = |\{z^{\tilde{n}} \in Pre_Z^U(z^n) : \tilde{n} \in \mathbb{N}\}|$; 4) the terminal states of U are either terminal Z-states or Y-states of the form y^n with $n \geq 1$.

Given a UWBTS U, item 1) in Definition 8 states that there is a unique control decision defined at each Y-state y^n in U. Item 2) illustrates that if f_{zy} is defined at $z \in Q_Z$ in the complete WBTS T, then it should also be defined at $z^n \in Q_Z^U$. Item 3) specifies how function δ_u is updated with transitions, i.e., the integer component of a state is n if there are n states in its predecessors that have the same Y- or Z-state component. Item 4) implies that any branch of the UWBTS ends a repeated Y-state of a Z-state without outgoing transitions.

Given a UWBTS U, we may also extract a supervisor from it. First, we merge each Y-state y^n with $n \geq 1$ and its predecessor state y^0, which results in a new EWBTS, denoted by $\tilde{U}$. In other words, $\tilde{U}$ comes from removing states $\{y^n \in Q_Y^U : C_U(y^n) = \emptyset\}$ from U, then making any transition that originally reaches y^n go to the corresponding y^0 in $\tilde{U}$. Therefore, $\tilde{U}$ is a complete EWBTS. In addition, there is a unique control decision at each Y-state in $\tilde{U}$, which also indicates a unique control strategy (supervisor) in $\tilde{U}$. We denote this supervisor by S_U which is *realized* by an automaton $G_U = (Q_Y^{\tilde{U}}, E, \xi, y_0^n)$. Here, y_0^0 is the initial Y-state of $\tilde{U}$; $\xi : Q_Y^{\tilde{U}} \times E \to Q_Y^{\tilde{U}}$ is the transition function such that $\forall y^n \in Q_Y^{\tilde{U}}$, $\forall e \in E$: $\xi(y^n, e) = f_{zy}^u(f_{yz}^u(y^n, C_{\tilde{U}}(y^n)), e)$ if $e \in C_{\tilde{U}}(y)$. The language of the supervised system is $\mathcal{L}(S_U/G) = \mathcal{L}(S_U \times G)$.

Inspired by the idea of solving the nonblocking supervisory control problem under partial observation in [47], we propose Algorithm 5 which constructs a UWBTS U from T_{win}^{ndw}, merges the repeated Y-states, and returns a supervisor. The procedure $Unfold$ recursively adds new states and transitions from the initial state y_0^0. As discussed earlier, a supervisor achieves $W_d(T_m, N)$ by first entering $\mathcal{W}_{\mathrm{local}}$ and then repeatedly playing first desirable-window decision sequences. Specifically, we distinguish two cases. If the supervisor has not yet reached $\mathcal{W}_{\mathrm{local}}$ at the current Y-state y^n, i.e., the corresponding y in T_m does not belong to $\mathcal{W}_{\mathrm{local}}$, then we augment the current U in line 7 to lead the supervisor to enter $\mathcal{W}_{\mathrm{local}}$. Otherwise, if the supervisor has already entered $\mathcal{W}_{\mathrm{local}}$, then in line 9, we find a first desirable-window decision sequence $\gamma_1\gamma_2\cdots\gamma_j$ from state y with $h_j(y) \geq 0$. Since the supervisor wins the game on T_{win}^{ndw} from all states in $\mathcal{W}_{\mathrm{local}}$, such a sequence always exists.

For a first desirable-window decision sequence, we consider two cases and augment U correspondingly from line 9. First, if the whole sequence $\gamma_1\gamma_2\cdots\gamma_j$ is not in U, then we augment U in line 11. Second, if part of the decision string $\gamma_\ell\gamma_{\ell+1}\cdots\gamma_j$ $(\ell < j)$ already exists in U, then we augment U in line 14 so that the augmented part is finally subsumed into U. This is essentially the merging process mentioned in the last paragraph. Thanks to the inductive property (Remark 1 in Section III), we ensure that an N-step desirable window will be formed from any Y-state in $y^n \xrightarrow{\gamma_1} z_1^{n'_1} \xrightarrow{e_1} y_2^{n_2} \xrightarrow{\gamma_2} \cdots \xrightarrow{e_j} y_j^{n_j}$ at line 9 so that we may start finding another first desirable-window decision sequence from $y_j^{n_j}$. Meanwhile, there may be Z-states whose successors

---

Algorithm 5: Synthesize a Supervisor Solving Problem 2.

Input : T_{win}^{ndw}, $\mathcal{W}_{local}$, N
Output : a supervisor S_U solving Problem 2

1 $Q_Y^U = \{y_0^0\}$;
2 $U \leftarrow Unfold(T_{win}^{ndw}, N)$;
3 Return S_U;
Procedure: $Unfold(T_{win}^{ndw}, N)$
4 **while** $[\exists y^n \in Q_Y^U$ *s.t.* $C_U(y^n) = \emptyset] \vee [\exists z^{n'} \in Q_Z^U$ *such that* $\exists e \in Ctr(z) : f_{zy}^m(z,e)!$ *in* T_{win} *but* $f_{zy}^u(z^{n'},e)\neg!$ *in* $U]$ **do**
5 **for** $y^n \in Q_Y^U$ *s.t.* $C_U(y^n) = \emptyset$ **do**
6 **if** $y \notin \mathcal{W}_{local}$ **then**
7 Let $n_1 = n$, augment U with $y^{n_1} \xrightarrow{\gamma_1} z_1^{n'_1} \xrightarrow{e_1} y_2^{n_2} \cdots \xrightarrow{\gamma_N} z_m^{n'_m} \xrightarrow{e_m} y_{m+1}^{n_{m+1}}$ where $y_{m+1} \in \mathcal{W}_{local}$, and for $1 \leq i \leq m+1$, we have $n'_i = |\{\tilde{z}_i^{\tilde{n}} \in Pre_Z^U(y_i^{n_i}) : \tilde{z}_i = z_i,\ \tilde{n} \geq 0\}|$, $n_i = |\{\tilde{y}_i^{n} \in Pre_Y^U(z_i^{n'_i}) : \tilde{y}_i = y_i,\ \tilde{n} \geq 0\}|$;
8 **if** $y \in \mathcal{W}_{local}$ **then**
9 Find a run $y^n \xrightarrow{\gamma_1} z_1^{n'_1} \xrightarrow{e_1} y_2^{n_2} \xrightarrow{\gamma_2} \cdots \xrightarrow{e_j} y_j^{n_j}$ from y such that $\gamma_1\cdots\gamma_j$ is a first desirable-window decision sequence, let $n_1 = n$;
10 **if** $\nexists \ell < j$, *s.t. there exists a run* $y_\ell^{n_\ell} \xrightarrow{\gamma_\ell} z_\ell^{n'_\ell} \xrightarrow{e_\ell} y_{\ell+1}^{n_{\ell+1}} \cdots \xrightarrow{e_j} y_j^{n_j}$ *in* U **then**
11 Augment the current U with $y^{n_1} \xrightarrow{\gamma_1} z_1^{n'_1} \xrightarrow{e_1} y_2^{n_2} \cdots \xrightarrow{\gamma_j} z_j^{n'_j} \xrightarrow{e_j} y_{j+1}^{n_{j+1}}$ where for $1 \leq i \leq j$, we have $n'_i = |\{\tilde{z}_i^{\tilde{n}} \in Pre_Z^U(y_i^{n_i}) : \tilde{z}_i = z_i,\ \tilde{n} \geq 0\}|$, $n_i = |\{\tilde{y}_i^{n} \in Pre_Y^U(z_i^{n'_i}) : \tilde{y}_i = y_i,\ \tilde{n} \geq 0\}|$;
12 **else**
13 Find $\ell < j$ such that there exists $y_\ell^{n_\ell} \xrightarrow{\gamma_\ell} z_\ell^{n'_\ell} \xrightarrow{e_\ell} y_{\ell+1}^{n_{\ell+1}} \cdots \xrightarrow{e_j} y_j^{n_j}$ in U;
14 Augment the current U with $y^{n_1} \xrightarrow{\gamma_1} z_1^{n'_1} \xrightarrow{e_1} y_2^{n_2} \cdots \xrightarrow{\gamma_{\ell-1}} z_{\ell-1}^{n'_{\ell-1}} \xrightarrow{e_{\ell-1}} y_\ell^{n_\ell}$ where for $1 \leq i \leq \ell$, we have $n'_i = |\{\tilde{z}_i^{\tilde{n}} \in Pre_Z^U(y_i^{n_i}) : \tilde{z}_i = z_i,\ \tilde{n} \geq 0\}|$, $n_i = |\{\tilde{y}_i^{n} \in Pre_Y^U(z_i^{n'_i}) : \tilde{y}_i = y_i,\ \tilde{n} \geq 0\}|$ (the augmented part is subsumed into the existing structure at $y_\ell^{n_\ell}$) ;
15 **for** $z^{n'} \in Q_Z^U$ *s.t.* $\exists e \in \Gamma(z) : f_{zy}^m(z,e)!$ *in* T_{win} *but* $f_{zy}^u(z^{n'},e)$ *is not defined in the current* U **do**
16 **for** $e \in Ctr(z)$ *such that* $f_{zy}^m(z,e)$ *is defined in* T_{win} *but* $f_{zy}^u(z^{n'},e)$ *is not defined in* U **do**
17 Agument the current U with $z^{n'} \xrightarrow{e} y^n$ where $y = f_{zy}^m(z,e)$ and $n = |\tilde{y}^{\tilde{n}} \in Pre_Y^U(z^{n'}) : \tilde{y} = y,\ \tilde{n} \geq 0|$;

are not fully included in U, and then we augment U in line 17. We also update the index of states in the process, which repeats until no more states are added to U. The number of states in U actually reflects the supervisor's memory, which is bounded by $2 \cdot |T_m| \cdot N$ since at most $2 \cdot N$ Y-states and Z-states are added from each state in T_{win}^{ndw} when U is extended by Definition 7, and the state space of T_{win}^{ndw} is at most $|T_m|$. A state in T_{win}^{ndw} is examined at most once in Algorithm 5 and the unfolding is bounded by the length of a first desirable-window decision sequence; thus, the algorithm terminates after all states in T_{win}^{ndw} are checked and the unfolding is finished.

Theorem 4: If a supervisor is synthesized by Algorithm 5, then it solves Problem 2.

Proof: Suppose S_U is returned by Algorithm 5. We run Algorithm 5 when the supervisor has strategies to achieve $W_d(T_m, N)$. More specifically, the supervisor first make decisions to enter states in $\mathcal{W}_{\text{local}}$ and then always make desirable-window decision sequences to remain in $\mathcal{W}_{\text{local}}$. By construction of U, the supervisor always reaches some state after which it may perpetually play first desirable-window decision sequences to achieve N-step desirable windows. Hence, every infinite run in U belongs to $W_d(T_m, N)$ [see (9)]. Since S_U is extracted from U, every infinite run in S_U/G is an N-step desirable-window infinite run. S_U is already safe and live by Proposition 1; so it solves Problem 2. ■

Based on Theorems 2–4, we have shown the correctness and completeness of the whole procedure of solving Problem 2: from computing the winning region to synthesizing the supervisor. It is always possible to synthesize a supervisor solving Problem 2 following Algorithms 4 and 5.

Remark 5: We briefly discuss the complexity of Algorithm 5. In the procedure $Unfold$, there are at most $|T_{win}^{ndw}|(|T_{win}^{ndw}| <= |T_m|)$ states between a state in T_{win}^{ndw} and states in $\mathcal{W}_{\text{local}}$. Then at most $2 \cdot N$ states are extended from each state in T_{win}^{ndw} when we take first desirable-window decision sequences. Next, at most $|T_{win}^{ndw}|$ states in U are "merged" to extract the supervisor S_U. Therefore, Algorithm 5 is of complexity $\mathrm{O}(N \cdot |T_m|)$.

Example 5: We continue Example 4 and synthesize a winning supervisor from T_{win}^{ndw} following Algorithm 5. First, we unfold T_{win}^{ndw} and let the supervisor play γ_0 from the initial state x_0. By the occurrence of u_1, we reach Y-state x_1 in $\mathcal{W}_{local}$. Next we choose first desirable-window decision sequence γ_1 at x_1 ($h_1(x_1) > 0$), γ_5 at x_3 ($h_1(x_3) > 0$), γ_8 at x_4 ($h_1(x_4) > 0$), γ_7 at x_6 ($h_1(x_6) > 0$), and γ_{10} at x_7 ($h_1(x_7) = 0$). Then we follow lines 15–17 in Algorithm 5 to augment U by adding successor states for the newly added Y-states and Z-states.

Note that at Y-state x_5, the only first desirable-window decision string is $\gamma_9\gamma_4\gamma_7$ by which the supervisor achieves $h_3(x_5) = 0$. This further implies that when x_1 is visited again, the supervisor has to make a different decision γ_4. Hence, we augment U with $x_5 \xrightarrow{\gamma_9} (x_5, \gamma_9) \xrightarrow{u_6} x_1^2 \xrightarrow{\gamma_4} (x_1, \gamma_4) \xrightarrow{u_2} x_6$ since $x_6 \xrightarrow{\gamma_7} (x_6, \gamma_7) \xrightarrow{e} x_7$ already exists after the augmentation from x_6. We continue construction until no more states are added to U. Finally, a UWBTS U is constructed and shown in Fig. 7 . The corresponding supervisor S_U is extracted from U following the earlier argument in this section and it is shown in Fig. 8 . As is seen, S_U has memory since it alternates between

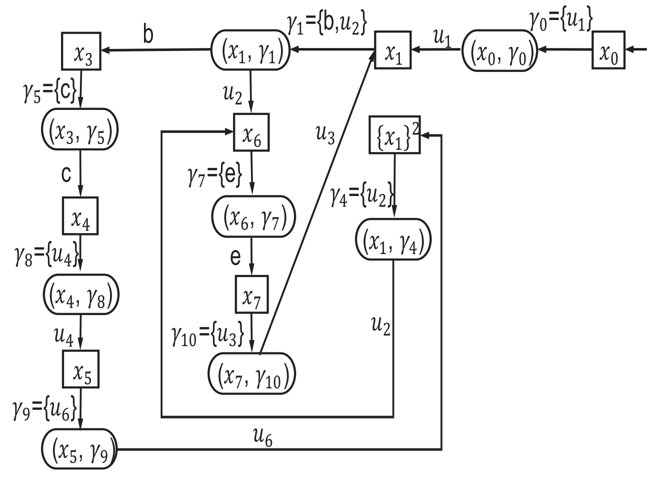


Fig. 7. One U after procedure *Unfold*.

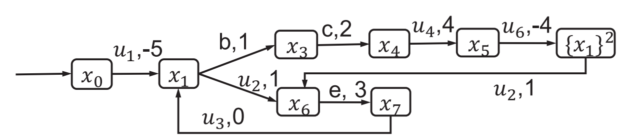


Fig. 8. Supervisor S_U solving Problem 2.

enabling b and disabling b at x_1. We may verify that S_U correctly solves Problem 2 as every infinite run in S_U/G is a three-step desirable-window infinite run.

VII. Conclusion

We developed, for the first time, a supervisory control framework which requires the local mean payoff within a fixed number of events be bounded by given thresholds. Specifically, two problems were formulated, depending on the window length. In order to solve these problems, the WBTS was introduced as a first step to transform the problems to a two-player game between the supervisor and the environment, where the qualitative conditions were resolved. Then we proposed several objectives for the supervisor and formulated two different games on the corresponding WBTS. Both games were analyzed and the algorithms for solving the games were proposed in sequence. We showed that the synthesized winning strategies are provably correct for the original supervisory control problems. Our results can be naturally extended to the multidimensional case where we consider weight vectors, and the details are not included here. For future work, it would be of interest to explore the same set of problems under partial observation.

References

[1] M. V. S. Alves, J. C. Basilio, A. E. C. da Cunha, L. K. Carvalho, and M. V. Moreira, "Robust supervisory control of discrete event systems against intermittent loss of observations," *Int. J. Control*, pp. 1–13, 2019.

[2] M. V. S. Alves, L. K. Carvalho, and J. C. Basilio, "New algorithms for verification of relative observability and computation of supremal relatively observable sublanguage," *IEEE Trans. Autom. Control*, vol. 62, no. 11, pp. 5902–5908, Nov. 2017.

[3] K. R. Apt and E. Grädel, *Lectures in Game Theory for Computer Scientists*. Cambridge, MA, USA: Cambridge University Press, 2011.

[4] T. Brihaye, G. Geeraerts, A. Haddad, and B. Monmege, "Pseudopolynomial iterative algorithm to solve total-payoff games and min-cost reachability games," *Acta Inform.*, vol. 54, no. 1, pp. 85–125, 2017.

[5] K. Cai, R. Zhang, and W. M. Wonham, "Relative observability of discrete-event systems and its supremal sublanguages," *IEEE Trans. Autom. Control*, vol. 60, no. 3, pp. 659–670, Mar. 2015.

[6] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Berlin, Germany: Springer, 2008.

[7] K. Chatterjee, L. Doyen, M. Randour, and J.-F. Raskin, "Looking at mean-payoff and total-payoff through windows," *Inform. Comput.*, vol. 242, pp. 25–52, 2015.

[8] S.-L. Chung, S. Lafortune, and F. Lin, "Limited lookahead policies in supervisory control of discrete event systems," *IEEE Trans. Autom. Control*, vol. 37, 12, pp. 1921–1935, Dec. 1992.

[9] W. Deng, J. Yang, and D. Qiu, "Supervisory control of probabilistic discrete event systems under partial observation," *IEEE Trans. Autom. Control*, vol. 64, no. 12, pp. 5051–5065, Dec. 2019.

[10] A. Farooqui and M. Fabian, "Synthesis of supervisors for unknown plant models using active learning," in *Proc. 15th IEEE Int. Conf. Autom. Sci. Eng.*, pp. 502–508, 2019.

[11] A. Giua and C. Seatzu, "Petri nets for the control of discrete event systems," *Softw. Syst. Model.*, vol. 14, no. 2 pp. 693–701, 2015.

[12] C. Gu, X. Wang, Z. Li, and N. Wu, "Supervisory control of state-tree structures with partial observation," *Inform. Sci.*, vol. 465, pp. 523–544, 2018.

[13] N. B. Hadj-Alouane, S. Lafortune, and F. Lin, "Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 6, no. 4 pp. 379–427, 1996.

[14] F. Hagebring and B. Lennartson, "Time-optimal control of large-scale systems of systems using compositional optimization. discrete event dynamic systems," *Theory Appl.*, vol. 29, no. 3, pp. 411–443, 2019.

[15] X. Han, Z. Chen, and R. Su, "Synthesis of minimally restrictive optimal stability-enforcing supervisors for nondeterministic discrete event systems," *Syst. Control Lett.*, vol. 123, pp. 33–39, 2019.

[16] Y. Hou, W. Wang, Y. Zang, F. Lin, M. Yu, and C. Gong, "Relative network observability and its relation with network observability," *IEEE Trans. Autom. Control*, vol. 65, no. 8, pp. 3584–3591, Aug. 2020.

[17] Y. Ji, X. Yin, and S. Lafortune, "Enforcing opacity by insertion functions under multiple energy constraints," *Automatica*, vol. 108, 2019, Art. no. 108476.

[18] Y. Ji, X. Yin, and S. Lafortune, "Supervisory control under local mean payoff constraints," in *Proc. 58th IEEE Conf. Decis. Control*, pp. 1043–1049, 2019.

[19] Y. Ji, X. Yin, and S. Lafortune, "Optimal supervisory control with mean payoff objectives and under partial observation," *Automatica*, vol. 123, 2021, Art. no. 109359.

[20] Y. Ji, X. Yin, and W. Xiao, "Local mean payoff supervisory control under partial observation," in *Proc. 15th IFAC Int. Workshop Discrete Event Syst.*, 2020, pp. 390–396.

[21] J. Komenda and T. Masopust, "Distributed computation of supremal conditionally controllable sublanguages," *Int. J. Control*, vol. 89, no. 2, pp. 424–436, 2016.

[22] J. Komenda and T. Masopust, "Computation of controllable and coobservable sublanguages in decentralized supervisory control via communication," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 27, no. 4, pp. 585–608, 2017.

[23] M. Kwiatkowska, D. Parker, and C. Wiltsche, "PRISM-games: Verification and strategy synthesis for stochastic multi-player games with multiple objectives," *Int. J. Softw. Tools Technol. Transfer*, vol. 20, no. 2 pp. 195–210, 2018.

[24] L. Lin, T. Masopust, W. M. Wonham, and R. Su, "Automatic generation of optimal reductions of distributions," *IEEE Trans. Autom. Control*, vol. 64, no. 3, pp. 896–911, Mar. 2019.

[25] L. Lin, Y. Zhu, and R. Su, "Synthesis of covert actuator attackers for free," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 30, pp. 561–577, 2020.

[26] Z. Ma, Z. He, and Z. Li, "Supervisory control in partially observable petri nets with sensor reduction," in *Proc. 15th IEEE Int. Conf. Autom. Sci. Eng.*, pp. 189–194, 2019.

[27] Z. Ma, Z. Li, and A. Giua, "Petri net controllers for generalized mutual exclusion constraints with floor operators," *Automatica*, vol. 74, pp. 238–246, 2016.

[28] A. A Malikopoulos, "Supervisory power management control algorithms for hybrid electric vehicles: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 5, pp. 1869–1885, Oct. 2014.

[29] H. Marchand, O. Boivineau, and S. Lafortune, "On optimal control of a class of partially observed discrete event systems," *Automatica*, vol. 38, no. 11, pp. 1935–1943, 2002.

[30] R. Meira-Góes, E. Kang, R. Kwong, and S. Lafortune, "Synthesis of sensor deception attacks at the supervisory layer of cyber-physical systems," *Automatica*, vol. 121, 2020, Art. no. 109172.

[31] S. Mohajerani, R. Malik, and M. Fabian, "Compositional synthesis of supervisors in the form of state machines and state maps," *Automatica*, vol. 76, pp. 277–281, 2017.

[32] V. Pantelic and M. Lawford, "Optimal supervisory control of probabilistic discrete event systems," *IEEE Trans. Autom. Control*, vol. 57, no. 5, pp. 1110–1124, May 2012.

[33] S. Pruekprasert and T. Ushio, "Optimal stabilizing supervisor of quantitative discrete event systems under partial observation," *IEICE Trans. Fundam. Electron.*, *Commun. Comput. Sci.*, vol. 99, no. 2, pp. 475–482, 2016.

[34] S. Pruekprasert and T. Ushio, "Supervisory control of partially observed quantitative discrete event systems for fixed-initial-credit energy problem," *IEICE Trans. Inform. Syst.*, vol. 100, no. 6, pp. 1166–1171, 2017.

[35] S. Pruekprasert, T. Ushio, and T. Kanazawa, "Quantitative supervisory control game for discrete event systems," *IEEE Trans. Autom. Control*, vol. 61, no. 10, pp. 2987–3000, Oct. 2016.

[36] A. Rashidinejad, M. Reniers, and L. Feng, "Supervisory control of timed discrete-event systems subject to communication delays and non-FIFO observations," in *Proc. 14th IFAC Int. Workshop Discrete Event Syst.*, 2018, pp. 456–463.

[37] A. Sakakibara and T. Ushi, "On-line permissive supervisory control of discrete event systems for scLTL specifications," *IEEE Contr. Syst. Lett.*, vol. 4, no. 3, pp. 530–535, Jul. 2020.

[38] K. W. Schmidt and C. Breindl, "A framework for state attraction of discrete event systems under partial observation," *Inform. Sci.*, vol. 281, 265–280, 2014.

[39] S. Shu and F. Lin, "Supervisor synthesis for networked discrete event systems with communication delays," *IEEE Trans. Autom. Control*, vol. 60, no. 8, pp. 2183–2188, 2015.

[40] R. Su, J. H. Van Schuppen, and J. E. Rooda, "The synthesis of time optimal supervisors by using heaps-of-pieces," *IEEE Trans. Autom. Control*, vol. 57, no. 1, pp. 105–118, Jan. 2012.

[41] T. Ushio and S. Takai, "Nonblocking supervisory control of discrete event systems modeled by mealy automata with nondeterministic output functions," *IEEE Trans. Autom. Control*, vol. 61, no. 3, pp. 799–804, Mar. 2016.

[42] B. J. C. van Putten, B. van der Sanden, M. Reniers, J. Voeten, and R. Schiffelers, "Supervisor synthesis and throughput optimization of partially-controllable manufacturing systems," *Discrete Event Dyn. Syst.: Theory Appl.*, vol. 31, pp. 103–135, 2020.

[43] Y. Wang and M. Pajic, "Supervisory control of discrete event systems in the presence of sensor and actuator attacks," in *Proc. 58th IEEE Conf. Decis. Control*, 2019, pp. 5350–5355.

[44] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*. Berlin, Germany: Springer, 2019.

[45] B. Wu, X. Zhang, and H. Lin, "Permissive supervisor synthesis for Markov decision processes through learning," *IEEE Trans. Autom. Control*, vol. 64, no. 8, pp. 3332–3338, Aug. 2019.

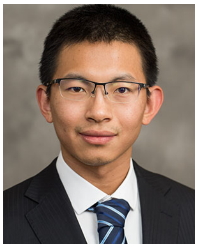
[46] P. Xu, S. Shu, and F. Lin, "Verification of delay co-observability for discrete event systems," *IEEE Control Netw. Syst.*, vol. 7, no. 1, pp. 176–186, Mar. 2020.

[47] X. Yin and S. Lafortune, "Synthesis of maximally permissive supervisors for partially-observed discrete-event systems," *IEEE Trans. Autom. Control*, vol. 61, no. 5, pp. 1239–1254, May 2016.

[48] X. Yin and S. Lafortune, "A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems," *IEEE Trans. Autom. Control*, vol. 61, no. 8, pp. 2140–2154, Aug. 2016.

[49] X. Yin and S. Lafortune, "Synthesis of maximally-permissive supervisors for the range control problem," *IEEE Trans. Autom. Control*, vol. 62, no. 8, pp. 3914–3929, Aug. 2017.

[50] H. Zhang, L. Feng, and Z. Li, "A learning-based synthesis approach to the supremal nonblocking supervisor of discrete-event systems," *IEEE Trans. Autom. Control*, vol. 63, no. 10, pp. 3345–3360, Oct. 2018.



Yiding Ji (Member, IEEE) received the B.Eng. degree in electrical engineering and automaton from Tianjin University, Tianjin, China, in 2014, and the M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 2016 and 2019, respectively.

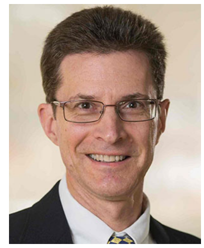
Since 2019 he has worked as a Postdoc Researcher with the Division of Systems Engineering, Boston University, Boston, MA, USA. His research interests include discrete event systems, formal methods, control systems, algorithmic game theory, machine learning, and cybersecurity and privacy.

Dr. Ji is now a member of the IEEE Control Systems Society Technical Committee on Discrete Event Systems.

Xiang Yin (Member, IEEE) was born in Anhui, China, in 1991. He received the B.Eng. degree from Zhejiang University, Hangzhou, China, in 2012 and the M.S. and Ph.D. degrees from the University of Michigan, Ann Arbor, MI, USA, in 2013 and 2017, respectively, all in electrical engineering.

Since 2017, he has been with the Department of Automation, Shanghai Jiao Tong University, Shanghai, China, where he is currently an Associate Professor. His research interests include formal methods, control of discrete event systems, model based fault diagnosis, and security of cyber and cyber–physical systems.

Dr. Yin is the recipient of the Outstanding Reviewer Awards from *Automatica*, IEEE TRANSACTIONS ON AUTOMATIC CONTROL, and *Journal of Discrete Event Dynamic Systems*. He was also the recipient of the IEEE Conference on Decision and Control Best Student Paper Award Finalist in 2016. He is the Co-Chair of the IEEE Control Systems Society Technical Committee on Discrete Event Systems.



Stéphane Lafortune (Fellow, IEEE) received the B.Eng. degree from Ecole Polytechnique de Montréal, Montreal, Canada, in 1980, the M.Eng. degree from McGill University, Montreal, Canada, in 1982, and the Ph.D. degree from the University of California at Berkeley, Berkeley, CA, USA, in 1986, all in electrical engineering.

He has been with the University of Michigan, Ann Arbor, MI, USA, since 1986, where he is a Professor of electrical engineering and computer science. His research interests are in discrete event systems and include multiple problem domains: modeling, diagnosis, control, optimization, and applications to computer and software systems. He coauthored, with C. Cassandras, the textbook *Introduction to Discrete Event Systems - Second Edition* (Springer, 2008).

Dr. Lafortune is a Fellow of IFAC (2017). He was the recipient the Presidential Young Investigator Award from the National Science Foundation in 1990 and the George S. Axelby Outstanding Paper Award from the Control Systems Society of the IEEE in 1994 (for a paper coauthored with S.-L. Chung and F. Lin) and in 2001 (for a paper coauthored with G. Barrett). He served as Editor-in-Chief for the *Journal of Discrete Event Dynamic Systems: Theory and Applications* from 2015 to 2020.