Local Mean Payoff Supervisory Control for Discrete Event Systems

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

Abstract—This work 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 the mean weight over a finite number of events, which serves as a measure to reflect the stability or robustness of the weight flows. The range of events to evaluate the mean payoff is termed the 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 of the system 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 then 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—discrete event systems, automata, supervisory control, mean payoff, algorithmic game theory

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].

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], [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], [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], [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 how many bits are contained in each packet. The information flow is generated when packets are transmitted through the network. At each stage, the sender transmits 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. Inspired by the flow control mechanism in the communication network, we may imagine that there is a sliding “window” over the network, where the window size indicates the 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 paper. Each event is associated a weight which represents 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 pre-specified bounds within a
finite number of events. For both scenarios, qualitative re-
quirements like safety and liveness are also required, 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. Then 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 every fixed time units.

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 one solution is not applicable to the other problem.

Under the framework of local mean payoff supervisory
control in this work, the supervisor issues the current com-
mand 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
work are also inspired by the literature in algorithmic graph
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], [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 paper 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 investigated 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 weighted bipartite transition
system, 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 paper.

A preliminary version of this paper with partial results
appears in [18], which only considers the second problem
discussed in this work. The improvements of the current work
are two-fold: 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, \( x_0 \in X \)
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 \( L(G) = \{ s \in E^+ : f(x_0, s) \} \) where \( ! \) means “is defined”.
The function \( \omega \) is additive and its domain can be extended to
\( E^+ \) by letting \( \omega(e) = 0, \omega(se) = \omega(s) + \omega(e) \) for all \( s \in E^+ \)
and \( e \in E \). In this work, we denote by \( W \) the maximum absolute
case of value 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: \( r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_n} x_n \). A run is initial if it starts from the initial
state of G. We denote by \( Run(G) \) and \( Run_{\omega,p}(G) \) 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 \( x_1 \xrightarrow{e_1} \cdots \xrightarrow{e_i} \) a prefix
of \( r \). In addition, for indexes \( j \) and \( m \) such that \( 1 \leq j < m \leq n \),
we call \( x_j \xrightarrow{e_j} \xrightarrow{e_{j+1}} \cdots \xrightarrow{e_m} x_{m+1} \) 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 $\forall i,j \in \{1, 2, \cdots, n-1\}, i \neq j \Rightarrow x_i \neq x_j$. If $r$ is a cycle, the corresponding string $e_1 e_2 \cdots e_{n-1}$ forms a loop, while the loop is called 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 $X_{sa} \subset X$ 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 nonblockiness, which is not the focus of this work. 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 $S : \mathcal{L}(G) \rightarrow 2^E$ and we denote by $\mathbb{S}$ the set of supervisors. The event set $E$ is partitioned as $E = E_c \cup E_{uc}$, 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 $E_{uc} \subseteq \gamma$, i.e., no uncontrollable event is disabled. Denote by $\Gamma$ the set of all admissible control decisions. In this work, 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 $\text{Run}(S/G)$ the set of runs in $S/G$, respectively. As marked states are not involved in $G$, we do not consider the standard nonblockiness of supervisors [6]. In the remainder of the work, a supervisor is called safe and (weakly) live if its supervised system is both safe and live.

Given $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} x_n$ 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 the introduction section, the mean weight within a sliding “window” provides a measure of stability or robustness of the flows, while the window size reflects the length of the horizon within which we evaluate those properties. In contrast to the conventional limit mean payoff which evaluates the “global” asymptotic performance of the system, we focus on the local mean payoff in this work. 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 work, 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 $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{m+1}} x_m$ 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 $\forall \ell \leq m, \frac{1}{\ell} \sum_{i=1}^{\ell} \omega(e_i) < v$. 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 $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{N+1}} x_{N+1}$ 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 work, 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 $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots$ is a desirable-window infinite run if $\exists \ell \geq 1$ such that $\forall \ell \geq i, \exists m_j \geq 0$, 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 $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \in \text{Run}_{inf}(G)$ is an $N$-step desirable-window infinite run if $\exists \ell \geq 1$ such that $\forall \ell \geq i$, we have that $r(j, j + N)$ forms an $N$-step desirable window.

Both Definition 3 and Definition 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 infinite number of events in [19]. Furthermore, it may be tolerable to accept 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 Definition 3 and Definition 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.

Notice that the inequalities in both Definitions 1 and 2 are the same as $\frac{1}{\ell} \sum_{i=1}^{\ell} (\omega(e_i) - v) \geq 0$, 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
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 $S \in S$ such that: (i) $S/G$ is both safe and live; (ii) for all $r \in \text{Run}_{inf}(S/G)$, $r$ 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 $S \in S$ such that: (i) $S/G$ is both safe and live; (ii) for all $r \in \text{Run}_{inf}(S/G)$, $r$ 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 $\sum_{i=1}^{j-1} \omega(e_i) < 0$ 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, 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 Figure 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 $x_1 \xrightarrow{a} x_2 \xrightarrow{c} x_3 \xrightarrow{a} x_1 \xrightarrow{a} x_2 \xrightarrow{c} x_3 \xrightarrow{a} x_2 \xrightarrow{d} \cdots$ is a 3-step desirable-window infinite run. However, $x_1 \xrightarrow{c} x_3 \xrightarrow{a} x_4 \xrightarrow{u_4} x_5 \xrightarrow{b} x_3 \xrightarrow{c} x_4 \xrightarrow{a} x_5 \xrightarrow{b} x_3 \cdots$ is not a 3-step desirable-window infinite run as its fragment $x_5 \xrightarrow{b} x_1 \xrightarrow{b} x_3 \xrightarrow{c} x_4$ is not a 3-step desirable window due to $\omega(u_6) < 0$, $\omega(u_6b) < 0$ and $\omega(u_6bc) < 0$. Moreover, unsafe state $x_8$ is reached under some strings. Hence supervisory control is necessary to restrict the behaviors of $G$. We will solve Problem 1 and Problem 2 on $G$ in the remaining sections of the paper.
the supervisor and the environment. A QY 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 $y \in QY$ safe if $y \notin X_m$. A QZ 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 $C_T(y) = \{ y \in \Gamma : f_{yZ}(y, y') \}$ to stand for the set of control decisions defined at $y \in QY$. $f_{yZ}$ is defined from Z-states to Y-states which are the reachable states in $G$ under the executed events. 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, $\omega$ is the same weight function inherited from $G$ and labels the events associated with $\omega$. Finally, $T$ is characterized analogously. The readers may refer to [3] for more details concerning memory of game strategies.

Given a WBTS $T$, a run in $T$ is of the form $r = y_1 \xrightarrow{y_2} \cdots \xrightarrow{y_n} y_{n+1}$. 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 $R_m(T)$ (respectively $R(T)$) 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 $R_{inf}(T)$ as the set of infinite runs in $T$.

Consider a run $r$ in a WBTS $T$, we say it generates a run $y_1 \xrightarrow{y_2} \cdots \xrightarrow{y_n} y_{n+1}$ 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 $\pi_s : R_m(T) \rightarrow \Gamma$ and the environment’s strategy as $\pi_e : R(T) \rightarrow E$. We denote the set of all supervisor’s and environment’s strategies by $\Pi_s$ and $\Pi_e$, respectively. A player selects a transition at its position following its strategy.

From a Y-state $y$ in $T$, if the supervisor plays $\pi_s$, and the environment plays $\pi_e$, a unique run is formed. We define $\text{Run}(\pi_e, y, T) = \{ y \xrightarrow{y_2} \cdots \xrightarrow{y_n} y_{n+1} \in G : n \in N^+, \forall i < n, y_i = \pi_e(y_1) \xrightarrow{y_i} y_i \}$ as the set of runs starting from $y$ and consistent with control strategy $\pi_e$, i.e., the control decisions are specified by $\pi_e$. 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 $f_{yZ}$ transitions reflect the executions of the enabled events. By Definition 5, a control strategy in $T$ works in the same mechanism as a standard supervisor in supervisory control theory [6]. In the following discussion, we will use the terms “supervisor” and “supervisor’s strategy (control strategy)” interchangeably. Given a control strategy $\pi_s$, and string $s$, we will use notations like $\pi_s/G$, $\pi_s(s)$ to stand for the supervised system under $\pi_s$ and the control decision made by $\pi_s$, on occurrence of $s$.

Intuitively, a strategy has memory if the player makes different decisions when the same state is visited again, otherwise, it is called memoryless. In a WBTS $T$, a control strategy $\pi_s$ is of finite memory if it can be encoded as a deterministic finite-state Moore automaton $A_M = (M, \delta_m, \delta, m_0)$ where $M$ is the finite set of states representing the memory; $\delta_m : M \times (QY \cup QZ) \rightarrow M$ is the transition function for memory update; $\delta : M \times QY \rightarrow QZ$ reflects the supervisor’s choice of successor states. If the supervisor plays strategy $\pi_s$ at $y \in QY$ with the current memory $m \in M$, then it chooses $z = \delta_m(m, y)$ 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 $\pi_s$ from $A_M$ such that $\pi_s(y_1) \xrightarrow{y_2} \cdots \xrightarrow{y_n} y_{n+1} = \pi_s$, where $\delta_m(m_0, y_1) = y_n$, $f_{yZ}(y, y' \pi_s)$ where the domain of $\delta_m$ can be extended to $(QY \cup QZ)^+$ naturally. Strategy $\pi_s$ is memoryless if $|M| = 1$, i.e., the supervisor’s choice only depends on its current state. 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 $\pi_s$ in a WBTS $T$, let $s = e_1 e_2 \cdots e_n \in \mathcal{L}(\pi_s/G)$ and the occurrence of $s$ induces a run $r(s) = y_0 = y_0 \xrightarrow{e_1} y_2 \xrightarrow{e_2} \cdots \xrightarrow{e_n} y_n \pi_s(e_1 e_2 \cdots e_n) = y_{n+1} \in T$. We denote by $Y(s)$ and $Z(s)$ the last Y-state and Z-state of $r(s)$, respectively. Formally speaking, if a control strategy $\pi_s$ is in a WBTS $T$, 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:

$$\begin{align*}
\text{Attr}_{T,0}^T(Q) &= Q \\
\text{Attr}_{T,i+1}^T(Q) &= \{ y \in QY \setminus \text{Attr}_{T,i}^T(Q) : \exists z \text{ s.t. } z \in \text{Attr}_{T,i}^T(Q) \} \\
&\cup \{ z \in QZ \setminus \text{Attr}_{T,i}^T(Q) : \forall y, y \in \text{Attr}_{T,i}^T(Q) \} \\
\text{Attr}_{T,i}^T(Q) &= \bigcup_{i \geq 0} \text{Attr}_{T,i}^T(Q) 
\end{align*}$$

By definition, the supervisor reaches $Q$ from $\text{Attr}_{T,i}^T(Q)$ by $i$ events for sure regardless of the environment’s strategies. Therefore, $\text{Attr}_{T,i}^T(Q)$ 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 $\text{Attr}_{T,i}^T(Q)$; 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 $O(n(T))$ to compute $\text{Attr}_{T,i}^T(Q)$ where $n(T)$ denotes the number of transitions in $T$. Note that we only add new states that are not in $\text{Attr}_{T,i}^T(Q)$ in each stage of calculating $\text{Attr}_{T,i}^T(Q)$. The environment’s attractor with respect to $Q$ is defined analogously and denoted by $\text{Attr}_{T,i}^E(Q)$.

Given a WBTS $T$ and a set of states $Q$ in $T$, we introduce a rank function $\sigma : QY \cup QZ \rightarrow N$ associating with every state the stage at which it is added to the attractor $\text{Attr}_{T,i}^T(Q)$.

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

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, $\sigma$ is always finite and may be obtained when the attractor

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAC.2021.3075186, IEEE Transactions on Automatic Control.
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 $\sigma(q)$ is, the “closer” $q$ is to $Q$ and $\sigma(q) = 0$ if $q \in Q$. Accordingly, for any $Y$-state $q \in Attrf_1^m(Q) \setminus Q$, 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 $q, q_1, q_2, \ldots \in Attrf_1^m(Q) \setminus Q$ 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 $Attrf_1^m(Q) \setminus Q$, 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 $\exists e \in E$, s.t. $f_2(z, e)$, 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 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 $G_T = (Q_T, E, \xi, y_0)$. Here $\xi : Q_T \times E \to Q_T$ is the transition function such that $\forall y \in Q_T, \forall e \in E : \xi(y, e) = f_2(y, \xi(y, e), e)$ if $y$ is chosen at $y$ and $e \in \gamma$ and $y$ is chosen at $y$. $y_0$ is the initial $Y$-state of $T$. We may compute the language of the supervised system as $L = L(S_T(G)) = L(S_T \times G)$ where $\times$ is the standard product between automata [6].

Given two WBTSs $T_1 = (Q_1, Q_2, E, \Gamma, f_1^1, f_1^2, f_2^1, f_2^2, \omega^1, y_1^1)$ and $T_2 = (Q_3, Q_4, E, \Gamma, f_1^3, f_1^4, f_2^3, f_2^4, \omega^2, y_1^3)$, we say that $T_1$ is a subgame of $T_2$, denoted by $T_1 \subseteq T_2$, if $Q_1 \subseteq Q_2$, $Q_2 \subseteq Q_3$ and for all $y \in Q_1$, $z \in Q_2$, $\gamma \in \Gamma$, $e \in E$, we have $f_2^1(y, \gamma, e) = z \Rightarrow f_2^2(y, \gamma, e) = z$ and $f_2^3(z, e) = y \Rightarrow f_2^4(z, e) = y$. Here the relation of the two weight functions does not really matter. Given a WBTS $T$ and a set of states $Q \subseteq Q_Y$, we denote by $T' = T \mid Q$ if $T' \subseteq T$ 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 $T_m = (Q_m, Q_2^m, E, \Gamma, f_1^m, f_2^m, \omega, y_0)$. 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 $T \subseteq T_m$. For simplicity, we denote by $|T_m|$ the number of states in $T_m$ and by $n_T$ 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 work. 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.

\begin{algorithm}[ht]
    \caption{Build $T_m$ w.r.t. $G$}
    \begin{algorithmic}[1]
        \Require $G$
        \Ensure $T_m = (Q_2^m, Q_2^m, E, \Gamma, f_1^m, f_2^m, \omega, y_0)$ w.r.t. $G$
        \State $Q_2^m = \{y_0\}$, $Q_2^m = \emptyset$;
        \State $DoDFS(y_0, G)$;
        \While{there exist $Y$-states that have no successor}
            \State Remove all such $Y$-states and their predecessor \end{algorithmic}
\end{algorithm}

Notice that we only include non-terminal $Z$-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 complete. Following a similar argument with Theorem VI in [48], we show the correctness of Algorithm 1 as follows.

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

\begin{proof}
We refer the reader to Proposition VI in [48] and we just sketch the idea here. By Definition 5, $Sta(z)$ tracks the reachable states under control decision $C_T(z)$ for $z \in Q_2^m$. Then by Algorithm 1, if for all $z \in Q_2^m$, we have $Sta(z) \neq X_u$, then any 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.
\end{proof}

**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, $|X|2|E| + |X|$ states ($Z$-states plus $Y$-states), where $2|E|$ 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 $O(|X|^2 2|E|)$.

Safety and liveness for Problem 1 and Problem 2 have been enforced. Before proceeding to tackle the quantitative requirements, 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 Figure 3. The rectangular states are $Y$-states while the round rectangular states are $Z$-states. As is seen, dashed $Z$-states $(x_1, y_1^1)$, $(x_2, y_1^2)$ and $(x_6, y_1^1)$ 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 Figure 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 Problem 2, thus further analysis is necessary.

![Fig. 3. The resulting structure after DoDFS (without dashed/shaded states)](image)

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

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 \(n \in \mathbb{N}^+\):

\[
\text{Run}_d(T_m, n) = \{ r \in \text{Run}(T_m) : r = y_1 \overset{γ_1}{\rightarrow} z_1 \overset{e_1}{\rightarrow} y_2 \overset{e_2}{\rightarrow} \cdots \overset{e_n}{\rightarrow} y_{n+1}, \exists \ell \leq n \text{ s.t. } \frac{1}{\ell} \sum_{i=1}^{\ell} \omega(e_i) \geq 0 \} \tag{3}
\]

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

\[
\text{Run}_d(T_m) = \{ r \in \text{Run}(T_m) : r = y_1 \overset{γ_1}{\rightarrow} z_1 \overset{e_1}{\rightarrow} y_2 \overset{e_2}{\rightarrow} \cdots \overset{e_n}{\rightarrow} y_{n+1}, n \in \mathbb{N}^+, \exists \ell \leq n \text{ s.t. } \frac{1}{\ell} \sum_{i=1}^{\ell} \omega(e_i) \geq 0 \} \tag{4}
\]

Then it comes to infinite runs in \(T_m\) and we introduce the desirable-window objective as:

\[
\text{W}_d(T_m) = \{ r \in \text{Run}_{\infty}(T_m) : r = y_1 \overset{γ_1}{\rightarrow} z_1 \overset{e_1}{\rightarrow} y_2 \overset{e_2}{\rightarrow} \cdots, \exists i \geq 1 \text{ s.t. } \forall j \geq i, \exists \ell \geq 1 : \frac{1}{\ell} \sum_{p=0}^{\ell-1} \omega(e_{j+p}) \geq 0 \} \tag{5}
\]

Comparing Equation 3 with Equation 4, we find that if desirable-window finite runs are successively formed on an infinite run, then such infinite run is included in \(\text{W}_d(T_m)\). The supervisor is said to achieve \(\text{W}_d(T_m)\) if it has a strategy \(π_s\) such that any infinite run consistent with \(π_s\) is in \(\text{W}_d(T_m)\). 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 \(\text{W}_d(T_m)\) generate desirable-window infinite runs in the original system \(G\). In what follows, we will study how to achieve \(\text{W}_d(T_m)\) and show that Problem 1 is solved by supervisors achieving \(\text{W}_d(T_m)\).

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 \(r = y_1 \overset{γ_1}{\rightarrow} z_1 \overset{e_1}{\rightarrow} y_2 \overset{e_2}{\rightarrow} \cdots \overset{e_n}{\rightarrow} y_{n+1}\) in \(T_m\), its total payoff is \(\sum_{i=1}^{n} \omega(e_i)\). Notice 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:

\[
\text{Tot}(T_m) = \{ r \in \text{Run}_{\infty}(T_m) : r = y_1 \overset{γ_1}{\rightarrow} z_1 \overset{e_1}{\rightarrow} y_2 \overset{e_2}{\rightarrow} \cdots, \limsup_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} \omega(e_i) \geq 0 \} \tag{6}
\]

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) \(\text{Tot}(T_m)\). It is shown in [4] that memoryless strategies are sufficient to win the total payoff game.

We have formulated a game with objective \(\text{W}_d(T_m)\). Note that our game on \(T_m\) with \(\text{W}_d(T_m)\) 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 (Lemma 10 and Lemma 11 in [7]) 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 $\pi_s$ is a supervisor’s winning strategy for $Tot(T_m)$. The main idea for showing Lemma 1 is to decompose any run consistent with $\pi_s$ 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 $\pi_e$ to ensure that for every run starting from that state and consistent with $\pi_e$, 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 $\pi_e$ 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 $\pi_e$ being a winning strategy for the environment.

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 $\mathcal{W}^{\text{dw}}$. Based on Lemmas 1 and 2, we slightly adapt Algorithm 4 and Algorithm 5 in [7] to present our Algorithm 2 for computing $\mathcal{W}^{\text{dw}}$. Specifically, we introduce an index $\ell$ to label states in $\mathcal{W}^{\text{dw}}$, which reflects when a state is added to $\mathcal{W}^{\text{dw}}$ 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 $\text{NegWindow}$. Since the desirable-window objective does not depend on the prefixes of runs, the environment should repeatedly forms undesirable windows to force the supervisor to lose the game. We denote by $W^{\text{neg}}$ 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 $\text{NegWindow}$. In other words, states not in $\text{NegWindow}(T_m)$ and their attractor states contribute to the supervisor’s winning region $\mathcal{W}^{\text{dw}}$. At the beginning, we assume 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 $W^{\text{neg}}$ may be shrunk each time $\text{NegTotal}(T_m)$ is called. As a result, more states are declared winning for the supervisor by recursively calling $\text{NegTotal}(T_m)$ and calculating the attractor of those states, until no new states are added to the winning region $\mathcal{W}^{\text{dw}}$. 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 $\mathcal{W}^{\text{dw}}$.

Now we take a closer look at the procedure $\text{NegWindow}$. 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 $\text{NegTotal}$ 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.

**Remark 3.** We briefly discuss the complexity of Algorithm 2. Here we denote by $n_e$ the number of edges in $T_m$ and $\mathcal{Q}$ the complexity of procedure $\text{NegTotal}$, i.e., the complexity of solving a total payoff game. First the complexity for procedure $\text{NegWindow}$ is $O(|T_m| \cdot (n_e + |Q|))$ since we take at most $|T_m|$ times of computation and each computation takes $(n_e + |Q|)$. Then the overall complexity of Algorithm 2 is $O(|Q| + |T_m| \cdot (n_e + |Q| + (n_e + |Q|))) = O(|Q|^2 \cdot (n_e + |Q|))$. 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 $\mathcal{W}^{\text{dw}}$ is shown similarly to Algorithm 4 and Algorithm 5 in [7]. 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 $\mathcal{W}^{\text{dw}}$. So the desirable-window objective is never achieved from those states, while the supervisor wins the game from $\mathcal{W}^{\text{dw}}$. The proof is omitted here.
By running Algorithm 2, we collect two sequences of states:

\[ W_{neg} = W_{neg}(0), W_{neg}(1), \ldots, W_{neg}(n) \] (7)

\[ W_{dw} = W_{dw}(0), W_{dw}(1), \ldots, W_{dw}(n) \] (8)

As \( T_m \) is finite and Algorithm 2 always terminates, both sequences are finite. Interestingly, they form two "chains":

\[ W_{dw}(i) \subseteq W_{dw}(j) \quad \text{and} \quad W_{neg}(j) \subseteq W_{neg}(i) \]

for any \( i, j, 0 \leq i < j \leq n \). Also we know that \( W_{neg}(0) = \text{NegWindow}(T_m) \), \( W_{dw}(n) = W_{dw} \) and \( W_{dw}(k) = Attr_{T_m}((Q_0^m \cup Q_1^m) \setminus W_{neg}(k−1)) \) for every \( 1 \leq k \leq n \) by Algorithm 2. When the supervisor's winning region \( W_{dw} \) is not empty and the initial state of \( T_m \) is included in \( W_{dw} \), we denote by \( T_{win} = T_m \mid W_{dw} \), whose state space is \( W_{dw} \). Otherwise we let \( T_{win} \) be empty. When \( T_{win} \) 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

\[ \text{Input} \quad T_m, \{W_{neg}\}, \{W_{dw}\} \]

\[ \text{Output} \quad \text{a supervisor achieving } W_d(T_m) \]

1) \( \text{Syn}(Y_0, T_m) \);

2) Extract the supervisor from the remaining subgame;

3) Procedure: \( \text{Syn}(y, T_m) \)

4) \( y \in W_{dw} \cap Q_y^m \) has not been specified a control decision in the current structure then

5) suppose \( \sigma(y) = k \), so \( y \in W_{dw}(k) \);

6) \( y \notin (Q_0^m \cup Q_1^m) \setminus W_{neg}(k−1) \) and \( y \notin ((Q_0^m \cup Q_1^m) \setminus W_{neg}(k−1)) \)

7) \( y \notin (Q_0^m \cup Q_1^m) \setminus W_{neg}(k−1) \) then

8) \( \text{apply the method developed in [4]} \)

9) \( \text{to solve a total payoff game on the subgame } \)

\[ T_{win} = T_m \mid (Q_0^m \cup Q_1^m) \setminus W_{neg}(k−1) \]

10) remove all control decisions defined at \( y \) except \( \gamma \), and take the accessible part;

11) \( z = f_{y}(y, \gamma) \);

12) for \( y \in T_{win} \) do

13) \( \text{Syn}(f_{y}(z, \varepsilon), T_m) \);

Intuitively, Algorithm 3 specifies a control decision at each \( Y \)-state in \( W_{dw} \) to lead the supervisor to states where it can perpetually achieve desirable-window runs. More specifically, every \( Y \)-state belongs to some \( W_{dw}(k) \) and two cases are categorized. First, if the current state \( y \) is not yet in \( (Q_0^m \cup Q_1^m) \setminus W_{neg}(k−1) \), then in line 6 the supervisor makes a decision to reach a successor that has a lower rank and is in \( Attr_{T_m}((Q_0^m \cup Q_1^m) \setminus W_{neg}(k−1)) \). Note that by the definition of the attractor (Equation 1) and the discussion in Section IV, \( \gamma \) will contribute to leading the supervisor towards \( (Q_0^m \cup Q_1^m) \setminus W_{neg}(k−1) \) ultimately. On the other hand, if the supervisor is already in \( (Q_0^m \cup Q_1^m) \setminus W_{neg}(k−1) \), 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 \( W_{neg}(k−1) \) by Algorithm 2, the supervisor wins the total payoff game from \( (Q_0^m \cup Q_1^m) \setminus W_{neg}(k−1) \). 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 \( \gamma \) specified at line 8 contributes to achieving \( W_d(T_m) \).

Since Procedure \( \text{Syn} \) runs on the attractor of the supervisor, the environment is unable to force the supervisor out of \( W_{dw} \). \( \text{Syn} \) 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 empty.

Proof. \("\text{if}\)\) When \( T_{win}^{dw} \) is empty, we denote by \( S \) a supervisor returned by Algorithm 3. Then for any \( Y \)-state \( y \in W_{dw} \), we know that there exists \( 1 \leq k \leq n \) such that \( y \notin (Q_0^m \cup Q_1^m) \setminus W_{neg}(k−1) \). So for any \( y \), we have that \( y \notin W_d(T_m) \). 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.

\(\text{Only if} \) We show it by contrapositive. When \( T_{win}^{dw} \) is empty, we know that in \( T_m \), for any control strategy \( \pi \), there exists an initial run \( r \) consistent with \( \pi \), such that \( r \notin W_d(T_m) \). This further implies that the run generated by \( r \) in the supervised system under \( \pi \) is not a desirable-window infinite run. That 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 \( T_{win}^{dw} \) is 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 \( x_1 \xrightarrow{y} x_2 \xrightarrow{y} x_3 \xrightarrow{y} \cdots \) in Figure 2 is not a desirable-window infinite run since \( \omega(ad) < 0 \), thus supervisor 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 \( W_{dw} \) exactly the state space of \( T_m \) in Figure 4. In other words, the supervisor achieves the desirable-window objective from every state in Figure 4, thus we are flexible for supervisor synthesis. Next, we run Algorithm 3 and choose control decision \( \gamma \) at \( x_1 \). The resulting supervisor \( S \) is extracted from \( T_m \) following the argument in Section IV and it is shown in Figure 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, 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:

$$W_d(T_m,N) = \{ r \in \text{Run}(T_m) : r = y_1 \overset{n}{\rightarrow} z_1 \overset{e_1}{\rightarrow} y_2 \cdots, \exists i \geq 1 \text{ s.t. } \forall j \geq i, \exists \ell \leq N, \frac{1}{\ell} \sum_{p=0}^{\ell-1} \omega(e_{j+p}) \geq 0 \}$$ (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 $\text{Run}(T_m,N)$, see Equation 3. Notice 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 $h_i : Q_m^o \cup Q_m^z \rightarrow \mathbb{Z}$ where

\[
\begin{align*}
&\forall q \in Q_m^o \cup Q_m^z : h_0(q) = 0 \\
&\forall q \in Q_m^i, \forall 1 \leq i \leq N : h_i(q) = \max_{z \in Q_m^o, y \in \mathbb{N}} \{ h_i(z) : f_{\mathbb{N}}^m(q,y) = z \} \\
&\forall q \in Q_m^z, \forall 1 \leq i \leq N : h_i(q) = \min_{y \in Q_m^i, e \in E} \{ \omega(e) + h_{i-1}(y) : f_{\mathbb{N}}^m(q,e) = y \}
\end{align*}
\]

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. Notice that we do not increase the index $i$ since an $f_{\mathbb{N}}^m$ 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 $f_{\mathbb{N}}^m$ transition indicates another 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 (Equation 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 $W_d^{ndw}$ the set of states where the supervisor wins the game for $W_d(T_m,N)$, termed as winning region. For the supervisor, a state in $W_d^{ndw}$ is called winning while a state not in $W_d^{ndw}$ is called losing. The complement of $W_d^{ndw}$ is the environment’s winning region for preventing the supervisor from achieving $W_d(T_m,N)$.

Algorithm 4: Compute the supervisor’s winning region for the N-step desirable-window objective

\begin{enumerate}
\item \textbf{Input} : $T_m, N$
\item \textbf{Output} : the supervisor’s winning region $W_d^{ndw}$
\item $W_d^{ndw} = \emptyset, n = 1, W_p^0 = Q_m^o \cup Q_m^z$
\item \textbf{while} $W_d^{ndw} \neq Q_m^o \cup Q_m^z$ and $W_p^{n-1} \neq \emptyset$ \textbf{do}
\item \hspace{1cm} $W_d^{ndw} = \text{WinLocal}(T_m,N)$
\item \hspace{1cm} $W_p^{ndw} = \text{Attr}(W_d^{ndw})$, $W_d^{ndw} \leftarrow W_d^{ndw} \cup W_p^{ndw}$
\item $T_m \leftarrow T_m \cup (Q_m^o \cup Q_m^z) \setminus W_d^{ndw}$, $n = n+1$
\item \textbf{return} $W_d^{ndw}$
\end{enumerate}

\begin{enumerate}
\item Procedure: $\text{WinLocal}(T_m,N)$
\item if $W_d^{ndw} = Q_m^o \cup Q_m^z$ or $W_d^{ndw} = \emptyset$ then $W_d^{ndw} = W_d^{ndw}$
\item else $T_m \leftarrow T_m \cup W_d^{ndw}$, $W_p = \text{WinLocal}(T_m,N)$
\item \textbf{return} $W_p$
\end{enumerate}

\begin{enumerate}
\item Procedure: $\text{StableWindow}(T_m,N)$
\item for $q \in Q_m^o \cup Q_m^z$ in the current structure \textbf{do}
\item \hspace{1cm} $h_0(q) = 0$
\item \textbf{for} $i = 1 : N$ \textbf{do}
\item \hspace{1cm} for $q \in Q_m^o \cup Q_m^z$ \textbf{do}
\item \hspace{2cm} calculate $h_i(q)$ by Definition 6
\item \hspace{1cm} for $q \in Q_m^z$ \textbf{do}
\item \hspace{2cm} calculate $h_i(q)$ by Definition 6
\item \hspace{1cm} return $W_d^{ndw} = \{ q \in Q_m^o \cup Q_m^z : \exists 1 \leq i \leq N \text{ s.t. } h_i(q) \geq 0 \}$
\end{enumerate}

Compared with conventional Büchi games [3], we need to ensure that states in $\{ q \in Q_m^o \cup Q_m^z : h_i(q) \geq 0 \text{ for } 1 \leq i \leq N \}$ are not only reached infinitely often but also consecutively. This is due to Equation 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].

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

Fig. 5. A supervisor solving Problem 1
WinLocal to compute state set \( W_s^p \) 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 \( W_s \). 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 \( W_s^p \), it also wins from the attractor of \( W_s^p \), i.e., \( W_{\text{attr}} \), calculated in line 4. Hence, the environment must avoid entering \( W_s^p \) 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 \( W_{\text{ndw}} \), then it also does so from all its successor states, which are contained in \( W_{\text{ndw}} \) as well. Algorithm 4 essentially computes the greatest fixed point. When it terminates, the states not in \( W_{\text{ndw}} \) are where the environment can falsify the window mean payoff objective.

Procedure StableWindow computes the value of window payoff functions for each state in the current game structure and returns \( W_s \) in line 20. Then the supervisor may always play the strategy prescribed by \( h_l(q) \geq 0 \) (following the decisions leading to \( h_l(q) \)) 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 \( h_l(q) \geq 0 \) is achieved from a state 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 \( W_{\text{ndw}} \) 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 \( q \notin W_{\text{ndw}} \). That is, there exists a control strategy \( \pi \in \Pi_s \), such that for all \( \pi \in \Pi_s \), the run from \( q \) and generated under \( (\pi, \pi) \) is in \( W_d(T_m, N) \).

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

“If”: Suppose that \( q \in W_{\text{ndw}} \), we show that \( q \in \cup n \geq 0 W_{\text{attr}}^n \) by contradiction. If \( q \notin \cup n \geq 0 W_{\text{attr}}^n \), then the environment always has a strategy from \( q \) to avoid reaching \( \cup n \geq 0 W_{\text{attr}}^n \), 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 \( \cup n \geq 0 W_{\text{attr}}^n \) 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 \( q \in W_{\text{ndw}} \).

By Theorem 2, if the supervisor’s winning region \( W_{\text{ndw}} \) is not empty and the initial state of \( T_m \) is included in \( W_{\text{ndw}} \), 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 \( T_{\text{win}} = T_m \mid W_{\text{ndw}} \), whose state space is \( W_{\text{ndw}} \). Otherwise we let \( T_{\text{win}} \) be empty.

**Theorem 3.** There exists a supervisor that solves Problem 2 if and only if \( T_{\text{win}} \) is not empty.

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

(“only if”) We show it by contrapositive. When \( T_{\text{win}} \) is empty, then by Theorem 2, we know that for any control strategy \( \pi_s \), there exists an initial run \( r \) consistent with \( \pi_s \) 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 \( T_{\text{ndw}} \) in the next subsection if the winning if \( T_{\text{win}} \) is not empty.

**Remark 4.** We briefly discuss the complexity of Algorithm 4. First for procedure StableWindow, each edge is visited at most \( N \) times to compute window payoff functions, so its complexity is \( O(n_e \cdot N) \). Then in procedure WinLocal, we call StableWindow for at most \( T_m \) times, so its complexity is \( O((T_m \cdot n_e) \cdot N) \). Finally, we call procedure WinLocal 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 \( O((T_m \cdot (n_e + T_m \cdot n_e \cdot N)) = O((T_m^2 \cdot n_e \cdot N)) \).

**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, here we associate a 4-dimensional vector with each state \( q \in Q \). The values are \( h_0(q) \) through \( h_3(q) \).

After one iteration of procedure StableWindow, states \((x_1, y_1), (x_1, y_2), x_2, \) and \((x_2, y_2)\) are not in \( W_s \) since the values of their window payoff functions are negative for all \( i \geq 1 \). All states reachable from \( x_1 \) in Figure 6 are returned by
**StableWindow, thus included in \( WF_p \) after procedure WinLocal. Although both \( x_0 \) and \((x_0, \gamma_0)\) have negative \( h_i \) for all \( i \geq 1 \), they are still included in \( WF_{sdw} \) since they are in the supervisor’s attractor of \( x_1 \). Notice that \( WF_{sdw} = Attr^{sdw}(WF_{ndw}) \) in this example. Finally \( WF_{ndw} \) is shown in Figure 6 whose state space constitutes the supervisor’s winning region in this example.

\[
\begin{array}{c}
0 \xrightarrow{u_1} x_4 \xrightarrow{u_4} x_5 \xrightarrow{u_5} x_6 \xrightarrow{u_6} 0 \\
0 \xrightarrow{\gamma_0 = (u_1)} (x_0, \gamma_0) \xrightarrow{\gamma_1 = (u_2)} x_1 \xrightarrow{\gamma_2 = (u_3)} x_2 \xrightarrow{\gamma_3} x_3 \\
0 \xrightarrow{\gamma_4 = (u_4)} x_4 \xrightarrow{\gamma_5 = (u_5)} x_5 \xrightarrow{\gamma_6 = (u_6)} x_6 \xrightarrow{\gamma_7 = (u_7)} x_7
\end{array}
\]

**Fig. 6.** \( WF_{ndw} \) with the supervisor’s winning region in Example 4

**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 \( \mathcal{WF}_{local} = \cup_{n \geq 0} \mathcal{WF}^{n}_{local} \), the union of each \( \mathcal{WF}^{n}_{local} \) obtained from line 3 of Algorithm 4.

**Definition 7 (First Desirable-Window Decision Sequences).** In \( \mathcal{WF}^{n}_{ndw} \), at \( Y \)-state \( y \in \mathcal{WF}_{local} \), a sequence \( \gamma_0 \gamma_1 \cdots \gamma_j \in \Gamma \) with \( j \leq N \) forms a desirable-window decision sequence if there exists a run \( r = y \overrightarrow{z_1} \overrightarrow{z_2} \cdots \overrightarrow{z_j} \overrightarrow{y_j} \) 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 \( W_d(\text{Term}, N) \) in two steps. First it issues decisions to reach a \( Y \)-state in \( \mathcal{WF}_{local} \). 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 \( y \in \mathcal{WF}_{local} \), there should exist a first desirable-window decision sequence so that the supervisor achieves a nonnegative sum, otherwise it contradicts with \( y \in \mathcal{WF}_{local} \). 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, “unfold” the WBTS and introduce the extended weighted bipartite transition system (EWBTS) w.r.t. a WBTS \( T_e \) as a tuple: \( T_F = (Q^F, \mathcal{Q}^F, E, \Gamma, f^F_{\mathcal{Q}^F}, f^F_{\mathcal{Q}^F}, \delta, \omega, \gamma^0_F) \).

Here we have \( Q^F = Q^Y \times \mathbb{N} \) and \( \mathcal{Q}^F = Q^Z \times \mathbb{N} \). With a slight abuse of notation, we also call \( Q^F \)-states as \( Y \)-states and \( \mathcal{Q}^F \)-states as \( Z \)-states. \( f^F_{\mathcal{Q}^F} : Q^F \times \mathbb{N} \to \mathcal{Q}^F \) and \( f^F_{\mathcal{Q}^F} : \mathcal{Q}^F \times \mathbb{N} \to Q^F \) are the transition functions. Specifically, \( f^F_{\mathcal{Q}^F}((y, n), \gamma) \) (respectively \( f^F_{\mathcal{Q}^F}((z, n), e) \)) is of the form \((f_\gamma(y, \gamma), \delta(y, n, \gamma)) \) (respectively \((f_\delta(z, n), \delta(z, n, e)) \)), where \( \xi : (Q^F \cup \mathcal{Q}^F) \times \mathbb{N} \times (\Gamma \cup E) \to \mathbb{N} \) is some function that updates the integer component of the states. The exact form of \( \delta \) is left unspecified and will be defined when we introduce a special EWBTS. \( \gamma^0_F = (y_0, 0) \) is the initial state. \( T_F \) 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 \( T_F \) is complete if \( \forall (y, n) \in Q^F \), \( C_T(y, n) \neq \emptyset \).

From the definition of the EWBTS, if we restrict the domains of \( f^F_{\mathcal{Q}^F} \) and \( f^F_{\mathcal{Q}^F} \) to \( Q_Y \) and \( Q_Z \), respectively, then they are reduced to \( f_{\mathcal{Q}^F} \) and \( f_{\mathcal{Q}^F} \) in a WBTS, respectively. However, function \( \delta \) 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 \((y, n) \in Q^F \) as \( y^n \) and \((z, n) \in \mathcal{Q}^F \) as \( z^n \). Given a state \( q^n \) in an EWBTS \( T_F \), we let \( \text{Pre}^{\Gamma}_F(q^n) \) and \( \text{Pre}^{\mathcal{Q}^F}_F(q^n) \) denote, respectively, the set of \( Y \)-states and the set of \( Z \)-states that may reach \( q^n \), excluding \( q^n \) itself. Here we also let \(| \cdot |\) be cardinality of a set.

**Definition 8 (Unfolded Weighted Bipartite Transition System).** An unfolded weighted bipartite transition System (UWBTS) is an EWBTS of a complete WBTS \( T_e \). It is a tuple \( U = (Q^U, \mathcal{Q}^U, E, \Gamma, f^U_{\mathcal{Q}^U}, f^U_{\mathcal{Q}^U}, \delta, \omega, \gamma^0_U) \) where \((i) \forall y^n \in \mathcal{Q}^F : C_U(y^n) = 1\); \( (ii) \forall z^n \in \mathcal{Q}^F : \forall e \in E : f^U_{\mathcal{Q}^U}(z^n, e) \in \mathcal{Q}^F \); \( (iii) \forall y^n \in \mathcal{Q}^F : n = |(\gamma^0_F \in \text{Pre}^{\Gamma}_F(y^n) : \gamma \in \mathbb{N})| \) and \( \gamma^0_F \in \mathcal{Q}^F : n' = |(z^n \in \text{Pre}^{\mathcal{Q}^F}_F(z^n) : \gamma \in \mathbb{N})| \); \( (iv) \) 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 (i) in Definition 8 states that there is a unique control decision defined at each \( Y \)-state \( y^n \) in \( U \). Item (ii) illustrates that if \( f^F_{\mathcal{Q}^F} \) is defined at \( z \) in \( Q_Z \) in the complete WBTS \( T_e \), then it should also be defined at \( z^n \in \mathcal{Q}^F \).

Item (iii) specifies how function \( \delta \) 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 (iv) 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 \( \bar{U} \). In other words, \( \bar{U} \) comes from removing states \( \{y^n \in \mathcal{Q}^F : C_U(y^n) = \emptyset\} \) from \( U \), then making any transition that originally reaches \( y^n \) go to the corresponding \( y^0 \) in \( \bar{U} \). Therefore, \( \bar{U} \) is a complete EWBTS. In addition, there is a unique control decision at each \( Y \)-state in \( \bar{U} \), which also indicates a unique control strategy (supervisor) in \( \bar{U} \). We denote this supervisor by \( S_{\bar{U}} \) which is realized by an automaton \( G_{\bar{U}} = (Q^F, E, \xi, \gamma^0_U) \). Here \( \gamma^0_U \) is the initial \( Y \)-state of \( \bar{U} \);
Algorithm 5: Synthesize a supervisor solving Problem 2

Input: \( T_{\text{ndw}} \), \( \mathcal{W}_{\text{local}}, N \)
Output: a supervisor \( S_U \) solving Problem 2

1. \( Q_U = \{0\}; \)
2. \( U \leftarrow \text{Unfold}(T_{\text{ndw}}, N); \)
3. Return \( S_U; \)

Procedure: \( \text{Unfold}(T_{\text{ndw}}, N) \)

4. while \( \exists \gamma \in Q_U \) s.t. \( C_U(\gamma^n) = 0 \) \( \cup \exists \gamma \in Q_U \) such that \( \exists e \in \text{Ctr}(z): f^{\mu}(z^n, e)! \) in \( T_{\text{win}} \) but \( f^{\mu}(z^n, e)\) is not defined in \( U \) do
5. for \( y^n \in Q_U \) s.t. \( C_U(\gamma^n) = 0 \) do
6. if \( y \notin \mathcal{W}_{\text{local}} \) then
7. Let \( n_1 = n \), augment \( U \) with
8. \( y_{n_1} \stackrel{h_1}{\rightarrow} y_{n_1} \rightarrow y_{n_2} \rightarrow \cdots \rightarrow y_{n_m} \rightarrow y_{n_{m+1}} \) where
9. \( y_{m+1} \in \mathcal{W}_{\text{local}} \), and for \( 1 \leq i \leq m+1 \), we have
10. \( n_i = \{y_{n_i} \in \text{Pre}_U(z_{n_i}) : z_i = z_i, \hat{n} \geq 0\}; \)
11. \( n_i = \{y_{n_i} \in \text{Pre}_U(z_{n_i}) : z_i = y_i, \hat{n} \geq 0\}; \)
12. if \( y \in \mathcal{W}_{\text{local}} \) then
13. Find a run \( y^n \stackrel{h_i}{\rightarrow} y_{n_1} \rightarrow y_{n_2} \rightarrow \cdots \rightarrow y_{n_j} \) from \( y \) such that \( y_1 \cdots y_j \) is a first desirable-window decision sequence, let \( n_1 = n \):
14. if \( \hat{n} < j \), s.t. there exists a run
15. \( y_{n_1} \stackrel{h_i}{\rightarrow} y_{n_2} \rightarrow \cdots \rightarrow y_{n_j} \) in \( U \) then
16. Augment the current \( U \) with
17. \( y_{n_1} \stackrel{h_i}{\rightarrow} y_{n_2} \rightarrow \cdots \rightarrow y_{n_j} \) where for \( 1 \leq i \leq \ell \), we have
18. \( n_i = \{y_{n_i} \in \text{Pre}_U(z_{n_i}) : z_i = y_i, \hat{n} \geq 0\}; \)
19. else
20. Find \( \ell \) such that there exists
21. \( y_{n_1} \stackrel{h_i}{\rightarrow} y_{n_2} \rightarrow \cdots \rightarrow y_{n_j} \) in \( U \);
22. Augment the current \( U \) with
23. \( y_{n_1} \stackrel{h_i}{\rightarrow} y_{n_2} \rightarrow \cdots \rightarrow y_{n_j} \) where for \( 1 \leq i \leq \ell \), we have
24. \( n_i = \{y_{n_i} \in \text{Pre}_U(z_{n_i}) : z_i = y_i, \hat{n} \geq 0\}; \)
25. (the augmented part is subsumed into the existing structure at \( y_{n_i} \));
26. for \( z^n \in Q_U \) s.t. \( \exists e \in \text{Ctr}(z): f^{\mu}(z^n, e)! \) in \( T_{\text{win}} \) but \( f^{\mu}(z^n, e)\) is not defined in the current \( U \) do
27. for \( e \in \text{Ctr}(z) \) such that \( f^{\mu}(z^n, e) \) is defined in \( T_{\text{win}} \) but \( f^{\mu}(z^n, e)\) is not defined in \( U \) do
28. Augment the current \( U \) with \( z^n \rightarrow y^n \) where \( y = f^{\mu}(z^n, e) \) and
29. \( n = \{y_{n_i} \in \text{Pre}_U(z_{n_i}) : z_i = y, \hat{n} \geq 0\}; \)

Inspired by the idea of solving the non-blocking supervisory control problem under partial observation in [47], we propose Algorithm 5 which constructs a UWBTS \( U \) from \( T_{\text{ndw}} \), merges the repeated \( Y \)-states and returns a supervisor. The procedure \text{Unfold} recursively adds new states and transitions from the initial state \( y_0 \). As discussed earlier, a supervisor achieves \( W_d(T_m, N) \) by first entering \( \mathcal{W}_{\text{local}} \) and then repeatedly playing first desirable-window decision sequences. Specifically, we distinguish two cases. If the supervisor has not yet reached \( \mathcal{W}_{\text{local}} \) at the current \( Y \)-state \( y^n \), i.e., the corresponding \( y \) in \( T_m \) does not belong to \( \mathcal{W}_{\text{local}} \), then we augment the current \( U \) in line 7 to lead the supervisor to \( \mathcal{W}_{\text{local}} \). Otherwise if the supervisor has entered \( \mathcal{W}_{\text{local}} \), then in line 9 we find a first desirable-window decision sequence \( y_1 \cdots y_j \) from state \( y \) with \( h_j(y) \geq 0 \). Since all states in \( \mathcal{W}_{\text{local}} \) are winning for the supervisor, 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 \( y_1 \cdots y_j \) is not in \( U \), then we augment \( U \) in line 11. Second, if part of the decision string \( y_1 \cdots y_j (\ell < j) \) 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 last paragraph. Due to the inductive property, we ensure that an \( N \)-step desirable window will be formed from any \( Y \)-state in \( y_1 \cdots y_j \) at line 9 so that we may start finding another first desirable-window decision sequence from \( y_1 \). Meanwhile there may be \( Z \)-states whose successors are not fully included in \( U \), 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 \( 2|T_m| \times 2 \) since at most \( 2 \times N \) \( Y \)-states and \( Z \)-states are added from each state in \( T_{\text{ndw}} \) when \( U \) is extended by Definition 7, and the state space of \( T_{\text{ndw}} \) is at most \(|T_m|\). A state in \( T_{\text{ndw}} \) 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 \( T_{\text{ndw}} \) 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 plays first desirable-window decision sequences to achieve \( N \)-step desirable windows. Hence, every infinite run in \( U \) belongs to \( W_d(T_m, N) \) (Equation 9). Since \( S_U \) is extracted from \( U \), every infinite run in \( S_U/G \) is a \( N \)-step desirable-window infinite run. \( S_U \) is already safe and live by Proposition 1, so it solves Problem 2.

Based on Theorems 2, 3 and 4, we have shown the correctness and completeness of the whole procedure from computing the winning region to synthesizing the supervisor. It is always possible to synthesize a supervisor solving Problem 2 following Algorithm 4 and Algorithm 5.

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

**Example 5.** We continue Example 4 and synthesize a winning supervisor from $T_{\text{ndw}}$ following Algorithm 5. First, we unfold $T_{\text{ndw}}$ and let the supervisor play $y_0$ from the initial state $x_0$. By the occurrence of $u_1$, we reach $Y$-state $x_1$ in $W_{\text{local}}$. Next we choose first desirable-window decision sequence $y_1$ at $x_1 \left(y_1(x_1) > 0\right)$, $y_2$ at $x_3 \left(y_2(x_3) > 0\right)$, $y_3$ at $x_4 \left(y_3(x_4) > 0\right)$, $y_4$ at $x_5 \left(y_4(x_5) > 0\right)$ and $y_5$ at $x_7 \left(y_5(x_7) = 0\right)$. Then we follow lines 15 to 17 in Algorithm 5 to augment $U$ by adding successor states for the newly added $Y$-states and $Z$-states.

Notice that at $Y$-state $x_5$, the only first desirable-window decision string is $y_7 y_7$ by which the supervisor achieves $h_5(x_5) = 0$. This further implies that when $x_1$ is visited again, the supervisor has to make a different decision $y_6$. Hence, we augment $U$ with $x_6 \left(y_6, y_7\right)$ and $x_7 \left(y_7, y_7\right)$ since $x_6 \left(y_7, x_6, y_7\right) \rightarrow x_7$ 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 Figure 7. The corresponding supervisor $S_U$ is extracted from $U$ following the earlier argument in this section and it is shown in Figure 8. As is seen, $S_U$ has memory since it alternates between 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 3-step desirable-window infinite run.

**REFERENCES**


Yiding Ji received the Bachelor of Engineering degree of electrical engineering and automation from Tianjin University, China, in 2014. Then he received the Master of Science degree and the Ph.D degree of electrical and computer engineering both from the University of Michigan, Ann Arbor, in 2017 and 2019, respectively. From 2019 to 2020, he worked as a postdoc researcher at Division of Systems Engineering, Boston University, United States.

His research interests include discrete event systems, formal methods, control and automation, algorithmic game theory, machine learning and cyber security and privacy. He is now a member of IEEE and the IEEE Control Systems Society Technical Committee on Discrete Event Systems.

Xiang Yin was born in Anhui, China, in 1991. He received the B.Eng degree from Zhejiang University, China, in 2012, the M.S. degree from the University of Michigan, Ann Arbor, in 2013, and the Ph.D degree from the University of Michigan, Ann Arbor, in 2017, all in electrical engineering.

Since 2017, he has been with the Department of Automation, Shanghai Jiao Tong University, where he is currently an Associate Professor. His research interests include formal methods, control of discrete event systems, model-based fault diagnosis, security of cyber and cyber-physical systems. Dr. Yin received the Outstanding Reviewer Awards from AUTOMATICA, IEEE TRANSACTIONS ON AUTOMATION CONTROL and JOURNAL OF DISCRETE EVENT DYNAMIC SYSTEMS. Dr. Yin also received 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 received the B.Eng degree from Ecole Polytechnique de Montréal in 1980, the M.Eng degree from McGill University, Canada, in 1982, and the Ph.D degree from the University of California at Berkeley, United States, in 1986, all in electrical engineering. Since September 1986, he has been with the University of Michigan, Ann Arbor, where he is a Professor of Electrical Engineering and Computer Science. Lafortune is a Fellow of the IEEE (1999) and of IFAC (2017). He received 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 co-authored with S.-L. Chung and F. Lin) and in 2001 (for a paper co-authored with G. Barrett). Lafortune’s research interests are in discrete event systems and include multiple problem domains: modeling, diagnosis, control, optimization, and applications to computer and software systems. He co-authored with C. Cassandra, the textbook Introduction to Discrete Event Systems - Second Edition (Springer, 2008). Lafortune served as Editor-in-Chief of the JOURNAL OF DISCRETE EVENT DYNAMIC SYSTEMS: THEORY AND APPLICATIONS from 2015 to 2020.