2019 IEEE 58th Conference on Decision and Control (CDC)
Palais des Congrès et des Expositions Nice Acropolis
Nice, France, December 11-13, 2019

Supervisory Control under Local Mean Payoff Constraints

Yiding Ji, Xiang Yin and Stéphane Lafortune

Abstract— **This work investigates quantitative supervisory control with a local mean payoff objective for weighted discrete event systems. Weight flows are generated by the system and a supervisor must be designed to ensure that the mean payoff of weights over a fixed number of transitions never drops below a given threshold while the system is operating. The local mean payoff may be viewed as a stability measure of weight flows. We formulate the supervisory control problem and transform it to a two-player game between the supervisor and the environment. Next, window payoff functions are defined to characterize the objective for the supervisor in the game. Then we analyze the game and develop a method to synthesize game-winning supervisors, which solve the proposed problem.**

I. Introduction

In the context of discrete event systems (DES), supervisory control is a widely-studied topic. The plant under control is usually modeled as a finite discrete structure and a (qualitative) specification is given to model the desired behavior of the plant. The supervisor restricts the behavior of the plant so that the specification is achieved [4].

Since the system dynamics may not be perfectly captured by the monitoring sensors, the problem of supervisory control under partial observation naturally arises; for recent references, see, e.g., [1], [3], [7], [11], [12], [16], [17], [19]. In particular, a uniform supervisory control approach was proposed in [20] to enforce a series of properties on partially-observed DES. Another important topic is quantitative supervisory control of DES, where the performance of the system is evaluated by some quantitative measures in addition to the qualitative specification, see, e.g., [13], [14], [18]. Specifically, some recent works investigate supervisory control with limit average objectives like mean payoff functions, under either full observation [15] or partial observation [9].

In many applications, the system generates some quantitative flows associated with event occurrences during its operation. Supervisors may be designed to regulate the flows and ensure that the long-run average rate of the flow lies in an acceptable interval. The limit mean payoff provides a suitable metric for the long-run quantitative performance of the system, however, it suffers from the drawback of ignoring transient fluctuations of the flows. Suppose that there are two infinite weight sequences generated by the system: one is $6,0,0,0,0,0,6,0,0,0,0,0\cdots$, (one 6 every 6 transitions), while the other is $1,1,1,\cdots$ (the same weight 1 all the time). Obviously, both sequences have the same limit mean payoff 1, but the first one shows more fluctuations since the weights alternate between 0 and 6. To be more specific, if we evaluate mean payoff every three transitions, then it is either 0 or 2 for the first sequence while it is always 1 for the second. Thus, we may say that the weight flow in the second sequence is more *stable* since the flow rate fluctuates less. A stable flow is usually preferred since it is easier to regulate and may cause less damage to the system.

Motivated by the above considerations, we explore supervisory control under local mean payoff constraints in this paper. The supervisor ensures safety and liveness, i.e., no undesired state is reached and the system never terminates. It also guarantees that the mean payoff over a fixed number of transitions is within some bounds. The quantitative requirement is termed as a *stable-flow constraint*. Intuitively, we may imagine that there exists a "window", which is sliding with the occurrence of new events, and the supervisor regulates the mean payoff within that window. We formulate this problem as *stable-flow supervisory control problem*. Then we transform it to a two-player game between the supervisor and the environment on a suitable information structure called *weighted bipartite transition system (WBTS)*. We give the generic definition of WBTS, then construct the largest WBTS which contains all control strategies that are safe and live. Next, we introduce window payoff functions and properly define an objective for the supervisor on the game. We illustrate that the supervisor solves the proposed problem by achieving the objective. Our method is inspired by conventional mean payoff games [6], [10] and mean payoff games with window objectives [5]. However, to the best of our knowledge, this paper is the first one to consider a local mean payoff supervisory control problem in DES.

The following sections are organized as follows. Section II describes the system model and formulates the stable-flow supervisory control problem. Section III introduces the weighted bipartite transition system, transforms the proposed problem in Section II to a two-player game and partially solves the problem. Then in Section IV, we analyze the game and synthesize control strategies that completely solve the problem. Finally, Section V concludes the paper.

II. System Model and Problem Formulation

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 to each event. The weight represents the *payoff* of the event, which may be positive

Research supported in part by the US National Science Foundation under grant CNS-1738103 and by the National Natural Science Foundation of China (61803259, 61833012).

Y. Ji and S. Lafortune are with the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, Michigan, USA. {jiyiding;stephane}@umich.edu

X. Yin is with the Department of Automation, Shanghai Jiao Tong University, Shanghai, China. {yinxiang}@sjtu.edu.cn

or nonpositive. The domain of f can be extended to $X \times E^*$ in the standard manner [4]. The language generated by G is $\mathscr{L}(G) = \{s \in E^* : f(x_0, s)!\}$ where ! means "is defined". ω is additive and its domain can be extended to E^* by letting $\omega(\varepsilon) = 0$, $\omega(se_o) = \omega(s) + \omega(e_o)$ for $s \in E^*$ and $e \in E$.

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* in G 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 its initial state is the initial state of the system. We denote by $Run(G)$ the set of runs in system G. Given r, for some $1 \le j < m < n$, we call $x_j \xrightarrow{e_j} x_{j+1} \xrightarrow{e_{j+1}} \cdots \xrightarrow{e_m} x_{m+1}$ a *run fragment*, which is a run by itself. Furthermore, we denote by $r(j,m) = x_j \xrightarrow{e_j} \cdots \xrightarrow{e_m} x_{m+1}$ the run fragment of r starting from x_j and ending in x_{m+1}. We also call $x_j \xrightarrow{e_j} \cdots \xrightarrow{e_{n-1}} x_n$ a *suffix* of run r and $x_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{j-1}} x_j$ a *prefix* of r, for some $1 \le j \le n$.

Conventionally, the *safety* property in DES is expressed in a language inclusion manner [4]. Using the refinement process in [8], we are always able to equivalently discuss it in a state-based manner. In this work, we assume the refinement has been done *a priori* and consider safety in terms of states. We let X_{us} be the set of unsafe states and G is *safe* if no state in X_{us} is reached. We also consider the (weak) *liveness* property: G is live if $\forall s \in \mathscr{L}(G)$, $\exists u \in E$, s.t. $su \in \mathscr{L}(G)$, i.e., there is a transition defined out of any state in G. Every finite run may be extended to be infinite when G is live.

The system is controlled by a *supervisor* $S : \mathscr{L}(G) \to \Gamma$, which dynamically enables and disables events so that some specification is achieved [4]. 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 Γ the set of all admissible control decisions. In the context of this work, all events are *observable*. We use S/G to represent the controlled system under S. Accordingly, we denote by $\mathscr{L}(S/G)$ the language generated in S/G and $Run(S/G)$ the set of runs in S/G, respectively. We call a supervisor *safe* and *live* if its supervised system is both safe and live.

Given $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_n} x_{n+1}$, its *(accumulative) payoff* is $\sum_{i=1}^{n} \omega(e_i)$ while its *mean payoff* is $\frac{1}{n}\sum_{i=1}^{n} \omega(e_i)$. As illustrated in the introduction section, the mean payoff within a local "window" provides a measure of fluctuation of the weight flow generated with the system's operation. If the mean payoff changes abruptly with an event occurrence, then the weight flows are not "stable". For reliability and stability of the system, weight flows should remain relatively smooth without severe fluctuations. So we evaluate the *stability of flows* by the mean payoff within a fixed number of events, i.e., $\frac{1}{n}\sum_{i=1}^{n} \omega(e_i)$ should be within some interval $[v_1, v_2]$. As $\frac{1}{n}\sum_{i=1}^{n} \omega(e_i) \le v$ is equivalent with $\frac{1}{n}\sum_{i=1}^{n}(-\omega(e_i)) \ge -v$, we will only focus on one-side inequality $\ge$ in the rest of the work. To proceed, we have the following two definitions.

Definition 1 (Stable-Flow Window): Given system G, window size $N \in \mathbb{N}^+$ and mean payoff threshold $v \in \mathbb{Z}$, a finite run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_N} x_{N+1}$ in G forms a stable-flow window if $\exists \ell \le N$ such that $\frac{1}{\ell}\sum_{i=1}^{\ell} \omega(e_i) \ge v$.

Definition 2 (Stable-Flow Run): Given system G, window size $N \in \mathbb{N}^+$ and mean payoff threshold $v \in \mathbb{Z}$, an infinite run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots$ is a stable-flow run if $\exists i \ge 1$ such that $\forall j \ge i$, $r(j, j+N)$ forms a stable-flow window.

A stable-flow window states that the mean payoff of a run should stay no less than a given threshold after at most N event occurrences. It is a local objective defined over a finite range, in contrast to the limit mean payoff objective discussed in [9], which may be viewed as a global objective. We may imagine that the next N events form a window, in which the mean payoff is evaluated. Since the system is live and never terminates, it is more reasonable to emphasize the local mean payoff after the system has been operating for a while. That is why we require that stable-flow windows be repeatedly achieved from certain position x_i (not necessarily the initial state) in Definition 2. In other words, a stable-flow run is *independent* of finite prefixes. The inequality in Definition 1 is the same as $\frac{1}{\ell}\sum_{i=1}^{i=\ell}(\omega(e_i) - v) \ge 0$, 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.

Intuitively, the windows are "sliding" with the occurrence of new events. If there is a deviation violating the bound v, it should be compensated within at most the next N events.

Example 1: Consider the weighted automaton G in Figure 1, 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 $E_{uc} = \{u_1, u_2, u_3, u_4, u_5\}$. The weight of each event is shown in the figure. If the window size is $N = 3$, then a run traversing between x_1 and x_2 infinitely often is not a stable-flow run since $\omega(a) + \omega(d) < 0$, while the run traversing among x_1, x_6 and x_7 infinitely often is a stable-flow run.

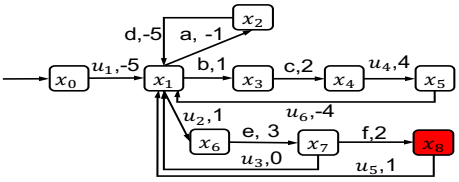

Fig. 1. The weighted automaton G in Example 1

When there exist non stable-flow runs in the system, a supervisory may be designed to "stabilize" the flows. We formulate Problem 1 to achieve both qualitative specifications (safety and liveness) and quantitative ones (stable flows).

Problem 1 (Stable-Flow Supervisory Control Problem): Given system G, window size $N \in \mathbb{N}^+$, set of unsafe states X_{us}, design a supervisor $S \in \mathbb{S}$ such that: (i) S/G is both safe and live; (ii) any infinite run in S/G is a stable-flow run.

Given a run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots$, suppose x_j with $j \le N$ is the first position where a stable-flow window is formed, i.e., $\sum_{i=1}^{j} \omega(e_i) \ge 0$ and $\sum_{i=1}^{j'} \omega(e_i) < 0$ for all $j' < j$. By some simple derivation, we know that $\sum_{i=j'}^{j} \omega(e_i) \ge 0 > \sum_{i=1}^{j'-1} \omega(e_i)$ holds for any $j' < j$, otherwise it contradicts with x_j being the first place where a stable-flow window is formed. So any run fragment $r(j', j)$ also forms a stable-flow window. This fact is called *inductive property* of stable-flow windows, which is leveraged later on to solve Problem 1.

III. Weighted Bipartite Transition System

In this section, we transform the automaton model to a two-player game structure and partially solve Problem 1. For this purpose, we first define the *weighted bipartite transition system (WBTS)* to characterize the game between the supervisor and the system (environment). Then a special WBTS is introduced to include all safe and live supervisors.

Definition 3 (Weighted Bipartite Transition System): A weighted bipartite transition system w.r.t. system G is a tuple $T = (Q_Y, Q_Z, E, \Gamma, f_{yz}, f_{zy}, \omega, y_0)$. Here $Q_Y \subseteq X$ is the set of states where the supervisor plays. $Q_Z \subseteq X \times \Gamma$ is the set of states where the environment plays. Let $I(z)$ and $\Gamma(z)$ be the state component and control decision component of $z \in Q_Z$, so that $z = (I(z), \Gamma(z))$. E is the set of events and 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 $f_{yz}(y,\gamma) = z \Rightarrow [z = (y,\gamma)]$. $f_{zy} : Q_Z \times E \to Q_Y$ 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 $f_{zy}(z,e) = y' \Leftrightarrow [e \in \gamma] \wedge [y' = f(y,e)]$. ω is the event weight function inherited from G and labeling f_{zy} transitions. Finally $y_0 \in Q_Y$ is the initial state and $y_0 = x_0$.

In essence, 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 $y \notin X_{us}$. 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. At $y \in Q^Y$, we use $C_T(y)$ to stand for the set of control decisions defined at y. f_{zy} transitions are defined from Z-states to Y-states, which are the reachable states under the executed events. Since the supervisor is unable to choose which event will occur, all enabled events should be defined at a Z-state and we put "$\Leftrightarrow$" in the last third line of Definition 3 to define f_{zy}. Similarly with G, ω is the weight function associated with f_{zy} transitions. A run in T is of the form $r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{\gamma_n} z_n \xrightarrow{e_n} y_{n+1}$ and we denote by $Run_y(T)$ (respectively $Run_z(T)$) the set of initial runs whose last states are Y-states (respectively Z-states).

Generally, both players make decisions based on the past history of control decisions and event occurrences, i.e., runs. In a WBTS T, we define the *supervisor's strategy (control strategy)* as $\pi_s : Run_y(T) \to \Gamma$ and the *environment's strategy* as $\pi_e : Run_z(T) \to E_o$. We denote the set of all supervisor's and environment's) strategies by Π_s and Π_e, respectively.

The f_{yz} transitions in a WBTS reflect the events enabled under control decisions, while f_{zy} transitions reflect the executions of the enabled events. As is seen, a control strategy in the WBTS just works in the same way as a standard supervisor. In the following discussion, we will use the terms "supervisor" and "supervisor's strategy (control strategy)" interchangeably. Intuitively, a strategy has *memory* if the player may make different decisions when the same state is reached again, otherwise, it is *memoryless*. We refer the reader to [2] for more detailed discussion of memory.

In a WBTS T, we define the *attractor* following the standard definition in [2]. Suppose Q is a set of states in T, then the supervisor's attractor $Attr_s^T(Q)$ is defined recursively as: $Q_0 = Q$, $Q_{i+1} = Q_i \cup \{y \in Q_Y : \exists z \in Q_Z, \gamma \in \Gamma \ s.t. \ f_{yz}(y,\gamma) = z\} \cup \{z \in Q_Z : \forall y \in Q_Y \ [(\exists e \in E, \text{ s.t. } f_{zy}(z,e) = y) \Rightarrow y \in Q_i]\}$ and $Attr_s^T(Q) = \bigcup_{i \geq 0} Q_i$. So the supervisor reaches Q_i from Q_{i+1} within one transition *for sure* regardless of the environment's strategies. Therefore, $Attr_s^T(Q)$ 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 $Attr_s^T(Q)$, otherwise it contradicts the definition of an attractor for the supervisor. The attractor for the environment is defined analogously.

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 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 $f_{zy}^1(z,e) = y \Rightarrow f_{zy}^2(z,e) = y$. Given a WBTS T and a set of states $Q \subseteq Q_Y \cup Q_Z$, we denote by $T' = T \downharpoonright Q$ if $T' \sqsubseteq T$ and Q is the state space of T', i.e., the game on T is restricted to a subgame with states in Q.

In a WBTS, a Y-state is called a *terminal state* if it has no successor states. Then T is called *complete* if $\forall y \in Q_Y$, y has successors. A Z-state z is *deadlocking* if $\nexists e \in E$, s.t. $f_{zy}(z,e)!$. Deadlocking states should be avoided to solve Problem 1.

In Algorithm 1, we construct the maximum complete WBTS without deadlocking 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 deadlocking Z-states or unsafe Y-states, we have $T \sqsubseteq T_m$.

Algorithm 1: Build T_m

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

$Q_Y^m = \{y_0\}$, $Q_Z^m = \emptyset$;
DoDFS(y_0, G);
while *there exist Y-states that have no successor* **do**
    Remove all such Y-states and their predecessor Z-states, take the accessible part;
return T_m;
Procedure: *DoDFS*(y, G)
for $\gamma \in \Gamma$ **do**
    $z = f_{yz}(y,\gamma)$;
    if $I(z) \notin X_{us}$ *and z is not deadlocking* **then**
        add transition $y \xrightarrow{\gamma} z$ to f_{yz}^m;
        if $z \notin Q_Z^m$ **then**
            $Q_Z^m = Q_Z^m \cup \{z\}$;
            for $e \in \gamma$ **do**
                $y' = f_{zy}(z,e)$;
                add $z \xrightarrow{e} y'$ to f_{zy}^m, its weight is $\omega(e)$;
                if $y' \notin Q_Y^m$ **then**
                    $Q_Y^m = Q_Y^m \cup \{y'\}$;
                    DoDFS(y', G);

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. Notice that we only include non-deadlocking Z-states without unsafe state components, as 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 V.I in [20], we show the correctness of Algorithm 1 as follows, while the detailed proof is omitted due to space limitations.

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

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 2. 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 deadlocking. 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, x_8 is removed by the while loop in Algorithm 1, so is (x_7,γ'_{10}). If we ignore all dashed/shaded states, (x_7,γ'_{10}) and transitions leading to them, the remaining structure is T_m.

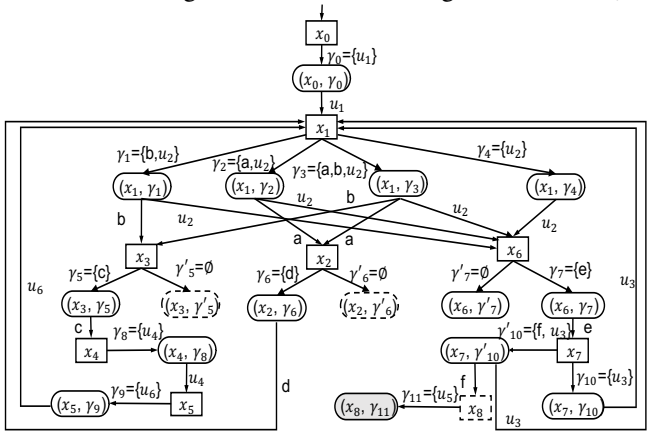

Fig. 2. The resulting structure after *DoDFS* (without dashed/shaded states)

IV. Supervisor Synthesis under Local Mean Payoff Constraints

In the previous section, we have obtained safe and live supervisors. Based on those partial solutions to Problem 1, we further investigate how to synthesize supervisors satisfying the local mean payoff requirement. For this purpose, we first transform this requirement to a proper objective on the game structure T_m. Then we analyze the game and completely solve Problem 1 by finding winning strategies for the supervisor.

A. Find the Supervisor's Winning Region

On the game graph T_m, given the window size $N \in \mathbb{N}^+$, we define the *stable-flow window objective* as $W^{sf}_{mp}(T_m,N) = \{r \in Run(T_m) : r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_N} y_{N+1}, \exists \ell \leq N \text{ s.t. } \frac{1}{\ell}\sum_{i=1}^{\ell} \omega(e_i) \geq 0\}$ and the *window mean payoff objective* as $W_{mp}(T_m,N) = \{r \in Run(T_m) : r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots, \exists i \geq 1 \text{ s.t. } \forall j \geq 0, \exists \ell \leq N, \frac{1}{\ell}\sum_{p=0}^{\ell-1} \omega(e_{j+i+p}) \geq 0\}$. Further observation shows that if the supervisor achieves $W_{mp}(T_m,N)$, then it perpetually achieves $W^{sf}_{mp}(T_m,N)$ from a certain point on. Also $W_{mp}(T_m,N)$ is equivalent to the local mean payoff requirement in Problem 1, so if a control strategy achieves $W_{mp}(T_m,N)$ on T_m, then it solves Problem 1. Next, we define the *window payoff function* in T_m for the sake of further characterizing $W^{sf}_{mp}(T_m,N)$ and $W_{mp}(T_m,N)$.

Definition 4 (Window Payoff Function): 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 $\forall q \in Q^m_Y \cup Q^m_Z : h_0(q) = 0$; $\forall q \in Q^m_Y, \forall 1 \leq i \leq N$: $h_i(q) = \max_{z \in Q^m_Z, \gamma \in \Gamma} \{h_i(z) : f^m_{yz}(q,\gamma) = z\}$ and $\forall q \in Q^m_Z, \forall 1 \leq i \leq N$: $h_i(q) = \min_{y \in Q^m_Y, e \in E} \{\omega(e) + h_{i-1}(y) : f^m_{zy}(q,e) = y\}$.

The window payoff function tracks the best worst-case accumulative weights that the supervisor may achieve from a state in T_m within at most N event occurrences. From the value of $h_i(q)$, we may backtrack a run from q, whose control decisions and event occurrences lead to $h_i(q)$. The supervisor aims to obtain a nonnegative accumulative payoff (also mean payoff) within the next N enabled events, while the environment aims to spoil that goal by enforcing negative payoffs. 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^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 accumulative payoff to-go so as to calculate $h_i(q)$, where we also 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 $W_{mp}(T_m,N)$.

The supervisor should repeatedly reach states q where $h_i(q) \geq 0$ for some $1 \leq i \leq N$, after the game has been played for a while. Next, we introduce the supervisor's *winning region*, denoted by $\mathscr{W}_s$, as the set of states where the supervisor has a strategy to achieve $W_{mp}(T_m,N)$. While the environment's winning region is the set of states from which it prevents the supervisor from achieving $W_{mp}(T_m,N)$. A state in $\mathscr{W}_s$ is *winning* while a state not in $\mathscr{W}_s$ is *losing* for the supervisor. Due to the determinacy of Büchi/co-Büchi games [2], a state is either winning or losing for a player.

Algorithm 2 recursively computes the supervisor's winning region for $W_{mp}(T_m,N)$. It generalizes the standard divide-and-conquer algorithm of solving Büchi/co-Büchi games [2]. Initially at line 1, each state in T_m is viewed as potentially losing for the supervisor. In line 3, we call procedure WinLocal to compute the set of states $\mathscr{W}^n_p$ from which the supervisor achieves $W_{mp}(T_m,N)$. Then in line 4, we add new winning states to the supervisor's winning region $\mathscr{W}_s$. Since whether the supervisor achieves $W_{mp}(T_m,N)$ does not depend on the finite prefixes of runs, if the supervisor is winning from $\mathscr{W}^n_p$, it also wins from the attractor of $\mathscr{W}^n_p$. Hence, the environment must avoid entering $\mathscr{W}^n_{attr}$ and remain in the subgame described by line 5 to preserve the chance of winning the game. After that, we iterate on the remaining subgame and recall procedure WinLocal to find more winning states for the supervisor; note that T_m gets updated in

Algorithm 2: Obtain the supervisor's winning region

Input : T_m, N
Output : supervisor's winning region $\mathscr{W}_s$

1 $\mathscr{W}_s = \emptyset$, $n = 1$, $W_p^0 = Q_Y^m \cup Q_Z^m$;
2 **while** $\mathscr{W}_s \neq Q_Y^m \cup Q_Z^m$ *and* $\mathscr{W}_p^{n-1} \neq \emptyset$ **do**
3 $\quad \mathscr{W}_p^n = WinLocal(T_m, N)$;
4 $\quad \mathscr{W}_{attr}^n = Attr_s^{T_m}(\mathscr{W}_p^n)$, $\mathscr{W}_s \leftarrow \mathscr{W}_s \cup \mathscr{W}_{attr}^n$;
5 $\quad T_m \leftarrow T_m \downharpoonright [(Q_Y^m \cup Q_Z^m) \setminus \mathscr{W}_s]$, $n = n+1$;
6 Return $\mathscr{W}_s$;
Procedure: $WinLocal(T_m, N)$
7 $\mathscr{W}_g = StableWindow(T_m, N)$;
8 **if** $\mathscr{W}_g = Q_Y^m \cup Q_Z^m$ *or* $\mathscr{W}_g = \emptyset$ **then**
9 $\quad \mathscr{W}_p = \mathscr{W}_g$;
10 **else**
11 $\quad T_m \leftarrow T_m \downharpoonright \mathscr{W}_g$, $\mathscr{W}_p = WinLocal(T_m, N)$;
12 **return** $\mathscr{W}_p$;
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 $\quad\quad h_i(q) = \min_{y \in Q_Y^m, e \in E} \{\omega(e) + h_{i-1}(y) : f_{zy}^m(q,e) = y\}$;
18 $\quad$ **for** $q \in Q_Y^m$ **do**
19 $\quad\quad h_i(q) = \max_{z \in Q_Z^m, \gamma \in \Gamma} \{h_i(z) : f_{yz}^m(q,\gamma) = z\}$;
20 **return** $\mathscr{W}_g = \{q \in Q_Y^m \cup Q_Z^m : \exists 1 \leq i \leq N \text{ s.t. } h_i(q) \geq 0\}$;

lines 5 and 11. In this manner, if the supervisor has strategies to win the game from a state in $\mathscr{W}_s$, then it also has strategies to win the game from its successor states, which are also contained in $\mathscr{W}_s$. The whole algorithm essentially computes the greatest fixed point, and when it terminates, the states not in $\mathscr{W}_s$ are where the environment can falsify the window mean payoff objective, so the algorithm is correct.

The values of window payoff functions are computed in StableWindow, which returns states q with $h_i(q) \geq 0$ for $1 \leq i \leq N$ in line 20. If the supervisor repeatedly achieves a nonnegative h_i from a state and its successors, then it repeatedly achieves $W_{mp}^{sf}(T_m, N)$ and $W_{mp}(T_m, N)$ from them. Since we need to ensure that $W_{mp}^{sf}(T_m, N)$ is always satisfied, we recursively call line 11 and WinLocal actually computes a least fixed point. Then the supervisor may always play 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 certain state q each time it makes a decision, while it suffices to record at most N states.

Theorem 1: Algorithm 2 correctly computes the supervisor's winning region for $W_{mp}(T_m, N)$.

Proof: Proof omitted here due to space limitations. ■

By Theorem 1, we further claim that if the initial state of T_m is contained in $Attr_s^{T_m}(\mathscr{W}_s)$, then the supervisor has strategies to reach states in $\mathscr{W}_s$ and win the game from y_0. That is, there exist solutions to Problem 1. If this is the case, we denote by $T_{win} = Ac(T_m \downharpoonright Attr_s^{T_m}(\mathscr{W}_s))$ where Ac stands for taking the accessible part from the initial state, otherwise, we let T_{win} be empty. We will discuss supervisor synthesis on T_{win} in the next subsection if T_{win} is not empty.

Remark 1: We briefly discuss the complexity of Algorithm 2. Here we denote by $|T_m|$ the number of states in T_m and n_e the number of edges in T_m. For StableWindow, each edge is visited at most N times in computing window payoff functions, so its complexity is $O(n_e \cdot N)$. Then in 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 WinLocal for at most $|T_m|$ times in Algorithm 2 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 3: We continue Example 2 and follow Algorithm 2 to find the winning region of the supervisor for $W_{mp}(T_m, N)$, where the window size is $N = 3$. 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 $q \in Q_Y^m \cup Q_Z^m$ and the values 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 StableWindow, states (x_1,γ_2), (x_1,γ_3), x_2 and (x_2,γ_6) are not in $\mathscr{W}_g$ since the values of their window payoff functions are negative for all $i \geq 1$. All states reachable from x_1 in Figure 3 are returned by *StableWindow*, thus included in $\mathscr{W}_p$ after *WinLocal*. Although both x_0 and (x_0,γ_0) have negative h_i for all $i \geq 1$, they are still included in $\mathscr{W}_s$ since they are in the supervisor's attractor of x_1. Finally T_{win} is shown in Figure 3. Here, $\mathscr{W}_s = Attr_s^{T_m}(\mathscr{W}_s)$.

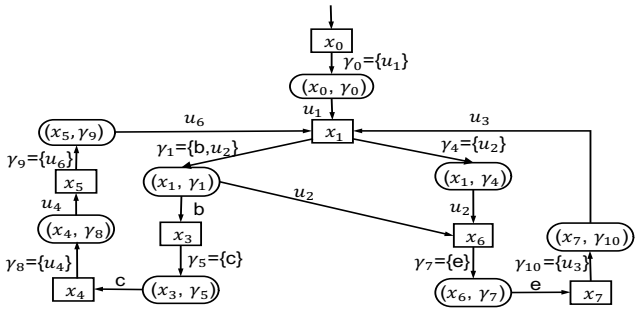

Fig. 3. T_{win} with the supervisor's winning region in Example 3

B. Synthesize Winning Supervisors

For supervisor synthesis, we define *first stable-flow decision sequences* to characterize how the supervisor achieves a nonnegative sum of weights within N event occurrences. Here we denote by $\mathscr{W}_{local} = \cup_{n \geq 0} \mathscr{W}_p^n$ which is the union of each $\mathscr{W}_p^n$ obtained from line 3 of Algorithm 2.

Definition 5 (First Stable-Flow Decision Sequences): At a Y-state $y \in \mathscr{W}_{local}$, a sequence of control decisions $\gamma_1 \gamma_2 \cdots \gamma_j$ with $j \leq N$ forms a a stable-flow 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 $j = min\{1 \le i \le N : h_i(y) \ge 0\}$.

A supervisor achieves the window mean payoff objective in two steps. First it plays strategies to reach some state in $\mathscr{W}_{local}$. Then it may repeatedly play strategies prescribed by *StableWindow* in Algorithm 2 to perpetually ensure a nonnegative sum within N event occurrences. Due to the inductive property mentioned at the end of Section II, the supervisor may play another first stable-flow decision sequence from y_j in Definition 5. So the supervisor keeps a memory bounded by N at each state to select successors (control decisions) and achieve $h_j(y) \ge 0$. The memory may be released immediately after a nonnegative sum of weights 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: $T_E = (Q_Y^E, Q_Z^E, E, \Gamma, f_{yz}^e, f_{zy}^e, \lambda, \omega, y_0^e)$. Here $Q_Y^E = Q_Y \times \mathbb{N}$ and $Q_Z^E = Q_Z \times \mathbb{N}$ are the (extended) Y-states and Z-states, respectively. $f_{yz}^e : Q_Y^E \times \Gamma \to Q_Z^E$ is the state transition from extend Y-states to extended Z-states and $f_{zy}^e : Q_Z^E \times E \to Q_Y^E$ is the state transition from Z-states to Y-states. Specifically, $f_{yz}^e((y,n),\gamma)$ (respectively $f_{zy}^e((z,n),e)$) is of the form $(f_{yz}(y,\gamma), \lambda(y,n,\gamma))$ (respectively $((f_{zy}(z,e), \lambda(z,n,e))$, where $\lambda : (Q_Y^E \cup Q_Z^E) \times \mathbb{N} \times (\Gamma \cup E) \to \mathbb{N}$ is some function that updates the integer component of the states. The exact form of λ 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. 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 f_{yz}^e and f_{zy}^e transitions to domains Q_Y and Q_Z, respectively, then they are consistent with f_{yz} and f_{zy} transitions in a WBTS. However, function λ has not been defined yet and it is left to count the number of times a state in the WBTS is revisited in unfolding the game graph. 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 $Pre_Y^{T_E}(q^e)$ and $Pre_Z^{T_E}(q^e)$ denote, respectively, the set of Y-states and the set of Z-states that may reach q^e, excluding q^e itself. We also let $|\cdot|$ be the number of elements in a set.

Definition 6: An unfolded weighted bipartite transition System (UWBTS) is an EWBTS of a complete WBTS T. It is a tuple $U = (Q_Y^U, Q_Z^U, E, \Gamma, f_{yz}^u, f_{zy}^u, \lambda_u, \omega, y_0^u)$ where (i) $\forall y^n \in Q_Y^U : |C_U(y^n)| = 1$; (ii) $\forall z^n \in Q_Z^U$, $\forall e \in E$: $f_{zy}(z,e)! \Rightarrow f_{zy}^u(z^n,e)!$; (iii) $\forall y^n \in Q_Y^U$: $n = |\{y^{\tilde{n}} \in Pre_Y^{T_U}(y^n) : \tilde{n} \in \mathbb{N}\}|$ and $\forall z^{n'} \in Q_Z^U$: $n' = |\{z^{\tilde{n}} \in Pre_Z^{T_U}(z^n) : \tilde{n} \in \mathbb{N}\}|$.

Given a UWBTS U, item 1 in Definition 6 states that there is a unique control decision defined at each Y-state y^n in U and we use $c_U(y^n)$ to stand for it. Item 2 illustrates that if an f_{zy} transition 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 λ_u is updated with state 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.

As is seen, there is a unique control strategy (supervisor) in a UWBTS U. We denote the supervisor by S_U, which is *realized* by an automaton $G_U = (Q_Y^U, E, \xi, y_0^n)$. Here y_0^0 is the initial Y-state of U; $\xi : Q_Y^U \times E \to Q_Y^U$ is the transition function such that $\forall y^n \in Q_Y^U$, $\forall e \in E$: $\xi(y^n, e) = f_{zy}^u(f_{yz}^u(y^n, c_U(y^n)), e)$ if $e \in c_U(y)$. The language of the supervised system is $\mathscr{L}(S_U/G) = \mathscr{L}(G_U \times G)$ where $\times$ is the product operation.

Algorithm 3: Synthesize a supervisor solving Problem 1

Input : T_{win}, $\mathscr{W}_{local}$, N
Output : a supervisor S_U solving Problem 1

$Q_Y^U = \{y_0^0\}$;
$U \leftarrow Unfold(T_{win}, N)$;
Return S_U;
Procedure: $Unfold(T_{win}, N)$
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* $\exists e \in \Gamma(z) : f_{zy}^m(z,e)!$ *in* T_{win} *but* $f_{zy}^u(z^{n'},e) \neg !$ *in* $U]$ **do**
  for $y^n \in Q_Y^U$ *s.t.* $C_U(y^n) = \emptyset$ **do**
    if $y \notin \mathscr{W}_{local}$ **then**
      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 $y_{m+1} \in \mathscr{W}_{local}$, and for $1 \le i \le m+1$, we have $n_i' = |\{\tilde{z}_i^{\tilde{n}} \in Pre_Z^U(y_i^{n_i}) : \tilde{z}_i = z_i,\ \tilde{n} \ge 0\}|$, $n_i = |\{\tilde{y}_i^n \in Pre_Y^U(z_i^{n_i'}) : \tilde{y}_i = y_i,\ \tilde{n} \ge 0\}|$;
    if $y \in \mathscr{W}_{local}$ **then**
      Find a first stable-flow decision sequence $\gamma_1 \cdots \gamma_j$ from y, let $n_1 = n$;
      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**
        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 $1 \le i \le j$, we have $n_i' = |\{\tilde{z}_i^{\tilde{n}} \in Pre_Z^U(y_i^{n_i}) : \tilde{z}_i = z_i,\ \tilde{n} \ge 0\}|$, $n_i = |\{\tilde{y}_i^n \in Pre_Y^U(z_i^{n_i'}) : \tilde{y}_i = y_i,\ \tilde{n} \ge 0\}|$;
      else
        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;
        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 $1 \le i \le \ell$, we have $n_i' = |\{\tilde{z}_i^{\tilde{n}} \in Pre_Z^U(y_i^{n_i}) : \tilde{z}_i = z_i,\ \tilde{n} \ge 0\}|$, $n_i = |\{\tilde{y}_i^n \in Pre_Y^U(z_i^{n_i'}) : \tilde{y}_i = y_i,\ \tilde{n} \ge 0\}|$ (the augmented part is subsumed into the existing structure at $y_\ell^{n_\ell}$) ;
  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**
    for $e \in \Gamma(z)$ *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**
      Agument the current U with $z^{n'} \xrightarrow{e} y^n$ where $y = f_{zy}^m(z,e)$ and $n = |\tilde{y}^{\tilde{n}} \in Pre_Y^U(z^{n'}) : \tilde{y} = y,\ \tilde{n} \ge 0|$;

Algorithm 3 constructs a UWBTS U from T_{win} and returns the supervisor S_U. The procedure $Unfold$ constructs

a UWBTS recursively by adding new states and transitions from the initial state y_0^0. As discussed earlier, a supervisor achieves $W_{mp}(T_m,N)$ by first entering $\mathscr{W}_{local}$ and then tries to repeatedly achieving $W_{mp}^{sf}(T_m,N)$. So if a Y-state y^n has no successors and it is not in $\mathscr{W}_{local}$, we augment the current U in line 7 to make the supervisor reach $\mathscr{W}_{local}$. Otherwise, we find a first stable-flow decision string $\gamma_1\gamma_2\cdots\gamma_j$ from $h_j(y)\geq 0$ in line 9. Since all states in $\mathscr{W}_{local}$ are winning for the supervisor, such a sequence always exists.

Afterwards, we continue to augment U and there are two cases. First, if the whole sequence $\gamma_1\gamma_2\cdots\gamma_j$ is not in U, 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, we augment U in line 14 so that the augmented part is finally subsumed into U. 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 reflects the supervisor's memory, bounded by $|T_{win}|\cdot N$.

Theorem 2: If a supervisor is synthesized by Algorithm 3, then it solves Problem 1.

Proof: Proof omitted here due to space limitations. ■

Example 4: We continue Example 3 and synthesize a winning supervisor from T_{win} following Algorithm 3. First, the supervisor plays γ_0 from the initial state x_0. By the occurrence of u_1, we reach Y-state x_1 in $\mathscr{W}_{local}$. Next we choose a first stable-flow 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 ($h_1(x_7)=0$). We also augment U with the corresponding Y-states and Z-states.

Notice that at Y-state x_5, the only first stable-flow 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. Finally, a UWBTS U is in Figure 4 and the supervisor S_U solving Problem 1 is in Figure 5. As is seen, S_U has memory since it alternates between enabling b and disabling b at x_1.

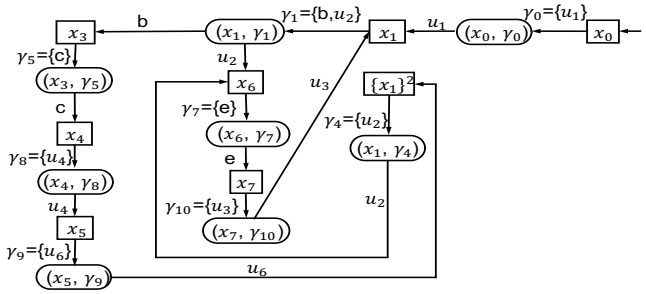

Fig. 4. One U after procedure Unfold

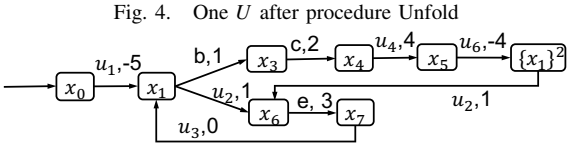

Fig. 5. A supervisor S_U solving Problem 1

V. Conclusion

We investigated, for the first time, a quantitative supervisory control problem requiring that the mean payoff within a fixed number of events satisfy a given threshold. After the problem formulation, we introduced the weighted bipartite transition system (WBTS) and transformed the problem to a two-player game on a special WBTS with the window mean payoff objective. Then we analyzed the game and proposed a supervisor synthesis algorithm to solve the problem. In future work, we will explore local mean payoff objectives where the window length is bounded, but not necessarily fixed.

References

[1] M. V. S. Alves, J. C. Basilio, A. E. C. da Cunha, L. K. Carvalho, and M. V. Moreira. Robust supervisory control against intermittent loss of observations. In *Proceedings of the 12th IFAC International Workshop on Discrete Event Systems*, pages 294–299, 2014.

[2] K. R. Apt and E. Grädel. *Lectures in game theory for computer scientists*. Cambridge University Press, 2011.

[3] K. Cai, R. Zhang, and W. M. Wonham. Relative observability of discrete-event systems and its supremal sublanguages. *IEEE Transactions on Automatic Control*, 60(3):659–670, 2015.

[4] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems – 2nd Edition*. Springer, 2008.

[5] K. Chatterjee, L. Doyen, M. Randour, and J.-F. Raskin. Looking at mean-payoff and total-payoff through windows. *Information and Computation*, 242:25–52, 2015.

[6] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

[7] C. Gu, X. Wang, Z. Li, and N. Wu. Supervisory control of state-tree structures with partial observation. *Info. Sciences*, 465:523–544, 2018.

[8] 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 Dynamic Systems:Theory and Applications*, 6(4):379–427, 1996.

[9] Y. Ji, X. Yin, and S. Lafortune. Mean payoff supervisory control under partial observation. In *Proceedings of the 57th IEEE Conference on Decision and Control*, pages 3981–3987, 2018.

[10] Y. Ji, X. Yin, and S. Lafortune. Enforcing opacity by insertion functions under multiple energy constraints. *Automatica*, DOI: 10.1016/j.automatica.2019.06.028, in press, 2019.

[11] J. Komenda and T. Masopust. Computation of controllable and coobservable sublanguages in decentralized supervisory control via communication. *Discrete Event Dynamic Systems: Theory and Applications*, 27(4):585–608, 2017.

[12] L. Lin, T. Masopust, W. M. Wonham, and R. Su. Automatic generation of optimal reductions of distributions. *IEEE Transactions on Automatic Control*, 64(3):896–911, 2019.

[13] H. Marchand, O. Boivineau, and S. Lafortune. On optimal control of a class of partially observed discrete event systems. *Automatica*, 38(11):1935–1943, 2002.

[14] V. Pantelic and M. Lawford. Optimal supervisory control of probabilistic discrete event systems. *IEEE Transactions on Automatic Control*, 57(5):1110–1124, 2012.

[15] S. Pruekprasert, T. Ushio, and T. Kanazawa. Quantitative supervisory control game for discrete event systems. *IEEE Transactions on Automatic Control*, 61(10):2987–3000, 2016.

[16] S. Shu and F. Lin. Supervisor synthesis for networked discrete event systems with communication delays. *IEEE Transactions on Automatic Control*, 60(8):2183–2188, 2015.

[17] T. Ushio and S. Takai. Nonblocking supervisory control of discrete event systems modeled by Mealy automata with nondeterministic output functions. *IEEE Trans. on Auto. Control*, 61(3):799–804, 2016.

[18] B. Wu, X. Zhang, and H. Lin. Permissive supervisor synthesis for Markov decision processes through learning. *IEEE Transactions on Automatic Control*, 64(8):3332 – 3338, 2019.

[19] X. Yin and S. Lafortune. Synthesis of maximally permissive supervisors for partially-observed discrete-event systems. *IEEE Transactions on Automatic Control*, 61(5):1239–1254, 2016.

[20] X. Yin and S. Lafortune. A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems. *IEEE Trans. on Automatic Control*, 61(8):2140–2154, 2016.