The synthesis of time optimal supervisors by using heaps-of-pieces

Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://se.wtb.tue.nl/

SE Report: Nr. 2009-08

The Synthesis of Time Optimal Supervisors by Using Heaps-of-Pieces

Rong Su, Jacobus E. Rooda and Jan H. van Schuppen

ISSN: 1872-1567

SE Report: Nr. 2009-08
Eindhoven, November 2009
SE Reports are available via http://se.wtb.tue.nl/sereports

Abstract

In many practical applications we are asked to compute a nonblocking supervisor that not only complies with some prescribed safety and liveness requirements but also achieve a certain time optimal performance such as throughput. In this paper we first introduce the concept of supremal minimum-time controllable sublanguage and define a minimum-time supervisory control problem, where the plant is modeled as a finite collection of finite-state automata, whose events are associated with weights, which represent their respective execution time. Then we show that the supremal minimum-time controllable sublanguage can be obtained by a terminable algorithm, where the execution time of each string is computed by using a technique extended from the theory of heaps-of-pieces.

1 Introduction

Since the Ramadge-Wonham supervisory control paradigm [22] [30] was invented, a large volume of research has been done on how to synthesize a nonblocking supervisor that complies with the safety and liveness requirements. But in practical applications we are also frequently asked to achieve a certain optimal performance, in particular, the time optimal performance such as throughput [31]. In this paper we discuss time optimal supervisory control. The system under our consideration consists of a finite collection of components modeled as deterministic finite-state automata, whose events are associated with weights, representing their firing durations. The requirement is modeled by an un-weighted deterministic finite-state automaton, which specifies the safety and liveness properties. Such a requirement model carries certain timing information manifested by the ordering of events specified in the requirement. In practical applications, it is possible that a requirement may carry more constraints on timing, e.g. it may explicitly specify the duration between two consecutive event firings, which forces a supervisor to take appropriate delay actions. This type of explicitly timed requirements is not considered in this paper. Since events have durations, event firings in different components may overlap. By initiating event firings at appropriate moments in different components, a supervisor may drive the system from the initial state to a desirable state within the minimum duration that takes account the possible elongation caused by the firings of uncontrollable events. For the time being we call such a minimum duration the *makespan* of the supervisor, whose precise definition will be given later in this paper. The control problem is to find the least restrictive nonblocking supervisor whose makespan is minimum among those of all possible supervisors that comply with the prescribed requirement.

To solve the aforementioned control problem, we make three contributions in this paper. First, we introduce the concept of supremal minimum-time controllable sublanguage and provide a precise formulation of the supremal minimum-time nonblocking supervisory control problem. Second, we present a novel timed supervisory control law that can achieve the time optimal performance specified by the supremal minimum-time controllable sublanguage. Finally, we present an algorithm that computes the supremal minimum-time controllable sublanguage. The algorithm utilizes a novel algorithm to determine the execution time of each string. The basic idea of our approach is to first solve a standard nonblocking supervisory control problem without time, which results in a supermal controllable sublanguage satisfying the prescribed requirement. Then we bring time information back in the obtained supremal controllable sublanguage, from which we compute the supremal minimum-time controllable sublanguage. To determine the execution time of each string, we first use the theory of heaps-of-pieces [27] [10] to build an appropriate heap model, then apply the (max,+) automaton technique to determine the hight of the heap, which is equal to the shortest possible execution time of that string.

A similar setting of timed discrete-event systems has been discussed in the literature about performance evaluation. For example, in [23] [8] [28] [9], time information is described by durations of events, and in [8] [9] the theory of heaps-of-pieces is used to analyze execution time of specific schedules. In comparison with our work, the above mentioned references are about analysis and not about synthesis. More explicitly, these references do not tell how to modify a system's behavior by using control in order to achieve certain performance. In our case, we need to find a supervisor that can achieve time optimal performance by simply using appropriate event disabling. Therefore, the problems in the above mentioned references are different from ours. Furthermore, in their settings no uncontrollable events are considered. Therefore, the concepts of controllability and least restrictive supervisory control are not present in the mentioned references.

By using an appropriate model conversion, the aforementioned time optimal control problem can be solved in the framework of timed automaton theory [1]. More explicitly, in each component we split every event, say a, into two events: $a_$start and $a_$end, then associating a clock with them such that the clock is reset when $a_$start is fired and $a_$end can be fired only when the clock value is equal to the prescribed duration. Such a model conversion has been discussed in, e.g. [11]. After the system is converted into a set of timed automata, we can apply an appropriate supervisor synthesis approach described in, e.g. [16] [3] [2] [26] [15], to compute a minimum-time supervisor. Nevertheless, such a conversion has the following major shortcoming. Too many clocks may be introduced during the conversion. As a result, the parallel composition of a large number of converted timed automata contains a large number of clocks, which incurs high complexity when a region automaton of the composition is constructed for subsequent analysis. This is because the complexity of constructing a region automaton is exponential with respect to the number of clocks. The concern of complexity is our main motivation to present a new approach based on the theory of heaps-of-pieces in this paper. More explicitly, in our approach the composition is only applied to untimed automata, and the execution time of each string of the composed system can be computed later based on algebraic operations. The advantage of this technique is that, the complexity caused by embedding the time information can be postponed to the last stage of analysis, where some appropriate greedy algorithms can be used so that the high complexity may be reduced or never appear. As a contrast, in the timed automaton framework, time information is explicitly embedded in each component model. As a result, the composition can be prohibitively large for subsequent analysis before we can take any complexity reduction procedure. Besides the difference on synthesis complexity, the supervisor synthesis techniques in the aforementioned papers are different from ours. More explicitly, they use game theoretical approach to deal with uncontrollable behaviors, and we simply adopt the standard definition of controllability in the Ramadge-Wonham paradigm to handle uncontrollable behaviors. By separating the time information from the system model in our framework, we can derive a control law, which is robust in the following sense. When the system does not act as fast as the supervisor expects, the supervisor still functions and the performance of the supervised system simply degrades accordingly. As a contrast, in the timed automaton framework a delay of the system's response to a supervisory control command may result in some behavior not specified by the corresponding time optimal supervisor, making the subsequent supervisory control infeasible. Therefore, it is a common assumption that every issued control command must be executed by the system immediately to avoid any potential timing error. In our opinion, this assumption is too strong to hold for many practical applications, which is the second motivation for us not to use the timed automaton framework in this paper.

In [5] the authors also describe least restrictive supervisory control of timed discrete-event systems in the Ramadge-Wonham paradigm. They adopt the Ostroff's semantics for timed transition models [19] for the plant and the controller. Time elapse is explicitly modeled by *ticks*. The semantics can be treated as a special case of the timed automaton theory, which contains only one universal clock. Besides the well known disadvantage associated with the discrete-time semantics, that is the limited modeling accuracy for time owing to discreteness of time, the supervisor synthesis approach proposed in [5] cannot be used to solve the problem described in this paper. This is because, explicitly enumerating time instances as ticks as used in [5] for computation cannot effectively handle the situation where event firings can be indefinitely delayed.

Supervisory control for time performance has also been discussed in the time/timed Petri

nets. In [6] [7] the authors define supervisory controllers for enforcing deadlines on transition firings in time Petri nets. Their goal is to find a supervisor, which can fire a designated transition within a prescribed deadline. Because of the existence of such a deadline, they can unfold a net to enumerate all possible firing sequences within the deadline. This makes their problem and approach significantly different from ours. In our case, we do not have such a deadline. Instead, we need to first decide whether there exists a controllable sublanguage, whose makespan is finite. Furthermore, the events in their setting are associated with firing intervals instead of durations and no uncontrollable transitions are present. In [25] the authors talk about maximally permissive control of time Petri nets. The time information is described by intervals instead of durations, and their control problem is about synthesizing a maximally permissive state-based feedback controller such that some prescribed state requirements are satisfied. It is different from ours because no time optimal performance is considered in their paper. In [12] [13] the authors discuss supervisory control of hybrid systems by using timed Petri nets, where time information is presented as transition holding time. But their control problem is different from ours in the sense that no time optimal performance and least restrictive supervision are under consideration. Furthermore, no uncontrollable transitions are present.

Our approach to find the supremal supervisors bears some similarity to optimal supervisory control, e.g. [20] [4] [14] [24] [17] [21]. These approaches are aimed to find a supervisor that can drive a deterministic plant from the initial state to a state within a target set with the minimum cost, (part of) which is defined as a sum weight. Nevertheless, the sum weight is different from the time weight used in this paper. This can be briefly explained as follows. We can use dynamic programming to determine an optimal supervisor [4] based on the fact that, a local path, which is optimal in terms of the sum weight, is guaranteed to be part of a globally optimal path that traverses the state associated with the locally optimal path. But this is typically not true for the timed case, where a locally time optimal path need not be part of any globally time optimal path. Thus, dynamic programming is in general not sufficient to be used for computing a time optimal supervisor. Because of the different natures of sum weights and time weights, their problem formulations are different from ours. As a result, their supervisor synthesis techniques and control strategies are different from ours as well.

This paper is organized as follows. In Section II we first provide all relevant necessary concepts about languages and time-weighted automata, then introduce a minimum-time supervisory control problem. After that we present a terminable algorithm in Section III, which computes the supremal minimum-time controllable sublanguages. Conclusions are drawn in Section IV.

2 Minimum-Time Supervisory Control Problem

In this section we first review basic concepts of languages and time-weighted systems. Then we present a minimum-time supervisory control problem and show that its solution, if exists, can be implemented by a special type of timed supervisory control map.

2.1 Concept of time-weighted systems

The notations for languages and relevant operations in this paper follow those in [29]. Let Σ be a finite alphabet, we use Σ^+ to denote the collection of all finite sequences of

events taken from Σ, and use Σ^* for the Kleene closure of Σ, i.e. $\Sigma^* := \Sigma^+ \cup \{\epsilon\}$, where ϵ is the empty string. Given two strings $s, t \in \Sigma^*$, s is called a *prefix substring* of t, written as $s \leq t$, if there exists $s' \in \Sigma^*$ such that $ss' = t$, where ss' denotes the concatenation of s and s'. For all string $s \in \Sigma^*$, $\epsilon s = s\epsilon = s$. A subset $L \subseteq \Sigma^*$ is called a *language*. $\overline{L} = \{s \in \Sigma^* | (\exists t \in L)\, s \leq t\} \subseteq \Sigma^*$ is called the *prefix closure* of L. We call L *prefix closed* if $L = \overline{L}$. Given two languages $L, L' \subseteq \Sigma^*$, let $LL' := \{ss' \in \Sigma^* | s \in L \wedge s' \in L'\}$ be the concatenation of L and L', which contains every string obtainable by concatenating one string from L and one string from L'.

Let $\Sigma' \subseteq \Sigma$. A map $P : \Sigma^* \to \Sigma'^*$ is called the *natural projection* with respect to (Σ, Σ'), if

1. $P(\epsilon) = \epsilon$
2. $(\forall \sigma \in \Sigma)\, P(\sigma) := \begin{cases} \sigma & \text{if } \sigma \in \Sigma' \\ \epsilon & \text{otherwise} \end{cases}$
3. $(\forall s\sigma \in \Sigma^*)\, P(s\sigma) = P(s)P(\sigma)$

Given a language $L \subseteq \Sigma^*$, $P(L) := \{P(s) \in \Sigma'^* | s \in L\}$. The inverse image map of P is

$$P^{-1} : 2^{\Sigma'^*} \to 2^{\Sigma^*} : L \mapsto P^{-1}(L) := \{s \in \Sigma^* | P(s) \in L\}$$

Given $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, the *synchronous product* of L_1 and L_2 is defined as:

$$L_1 || L_2 := P_1^{-1}(L_1) \cap P_2^{-1}(L_2) = \{s \in (\Sigma_1 \cup \Sigma_2)^* | P_1(s) \in L_1 \wedge P_2(s) \in L_2\}$$

where $P_1 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_1^*$ and $P_2 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_2^*$ are natural projections. Clearly, $||$ is commutative and associative.

A *finite-state automaton* is a 5-tuple $G = (X, \Sigma, \xi, x_0, X_m)$, where X stands for the state set, Σ for the alphabet, x_0 for the initial state, $X_m \subseteq X$ for the marker state set, and $\xi : X \times \Sigma \to X$ for the (partial) transition function, which is extended to $X \times \Sigma^*$. For all $x \in X$ and $\sigma \in \Sigma$, we use $\xi(x, \sigma)!$ to denote that the transition $\xi(x, \sigma)$ is defined. Let $L(G) := \{s \in \Sigma^* | \xi(x_0, s)!\}$ be the *closed* behavior of G and $L_m(G) := \{s \in L(G) | \xi(x_0, s) \in X_m\}$ for the *marked* behavior of G. We say G is *non-blocking* if $L(G) = \overline{L_m(G)}$. Let $\phi(\Sigma)$ denote the set of all finite-state automata, whose alphabets are Σ. Given a language $K \subseteq \Sigma^*$, suppose K is recognized by a finite-state automaton G, i.e. $L_m(G) = K$ and $L(G) = \overline{L_m(G)}$. Then we use $\kappa(K)$ to denote the canonical recognizer of K.

Let $\mathbb{R}^+$ be the set of positive reals. We treat $+\infty$ as a number, where

1. $+\infty = +\infty$
2. $(\forall a \in \mathbb{R}^+)\, a < +\infty \wedge +\infty + a = +\infty$

A *time-weighted system* is a 3-tuple $(\mathcal{G} = \{G_i \in \phi(\Sigma_i) | i \in I\}, f, h)$, where I is a finite index set, $\mathcal{G}$ is a collection of finite-state automata, $f : \cup_{i \in I} \Sigma_i \to \mathbb{R}^+$ is the time-weighted function on events, and $h : (\cup_{i \in I} \Sigma_i) \times (\cup_{i \in I} \Sigma_i) \to \{0, 1\}$ is the mutual exclusion function, where $h(\sigma, \sigma) = 1$ for all $\sigma \in \cup_{i \in I} \Sigma_i$. For each event $\sigma \in \cup_{i \in I} \Sigma_i$, $f(\sigma)$ denotes the duration required for σ to be completed. For each $(\sigma, \sigma') \in (\cup_{i \in I} \Sigma_i) \times (\cup_{i \in I} \Sigma_i)$,

$h(\sigma, \sigma') = 1$ if the firings of σ and σ' are mutually exclusive, i.e. if one event is under execution, the other event cannot be fired; otherwise, $h(\sigma, \sigma') = 0$. Since mutual exclusion is symmetric, we have $h(\sigma, \sigma') = h(\sigma', \sigma)$. For notation simplicity, we write $h(\sigma, \sigma')$ to denote both $h(\sigma, \sigma')$ and $h(\sigma', \sigma)$. Let $L(\mathcal{G}) := ||_{i \in I} L(G_i)$ and $L_m(\mathcal{G}) := ||_{i \in I} L_m(G_i)$. We call $\{\Sigma_i | i \in I\}$ the *alphabet* of $\mathcal{G}$, and use $\Phi(\{\Sigma_i | i \in I\})$ to denote the collection of all time-weighted systems, whose alphabets are $\{\Sigma_i | i \in I\}$. We use $\tilde{\Phi}(\{\Sigma_i | i \in I\})$ to denote the collection of such $\mathcal{G}$'s without the time-weighted function f.

Definition 2.1. Given a time-weighted system $(\mathcal{G} = \{G_i \in \phi(\Sigma_i) | i \in I\}, f, h)$, let $s \in L(\mathcal{G})$. Suppose $s = \sigma_1 \cdots \sigma_n$ for some $n \in \mathbb{N}$. A *time-stamp* of s w.r.t. $(\mathcal{G}, f, h)$ is a nondecreasing list,

$$w = (t_k^s \in \mathbb{R}^+ | k = 1, \cdots, n)$$

where

$$(\forall q, v \in \{1, \cdots, n\})\, q < v \wedge h(\sigma_q, \sigma_v) = 1 \Rightarrow t_q + f(\sigma_q) \leq t_v$$

Let $T_{\mathcal{G},f,h}(w) := \max\{t_1 + f(\sigma_1), \cdots, t_n + f(\sigma_n)\}$. Let $\Theta_{\mathcal{G},f,h}(s)$ be the collection of all time-stamps for s. We call $\upsilon_{\mathcal{G},f,h}(s) := \min_{w \in \Theta_{\mathcal{G},f,h}(s)} T_{\mathcal{G},f,h}(w)$ the *execution time* (or *makespan*) of s in $(\mathcal{G}, f, h)$. For all $W \subseteq L_m(\mathcal{G})$, let $\omega(\mathcal{G}, f, h, W) := \sup_{s \in W} \upsilon_{\mathcal{G},f,h}(s)$ be the *makespan* of W with respect to $(\mathcal{G}, f, h)$. As a convention, let $\omega(\mathcal{G}, f, h, \varnothing) := +\infty$. □

Each t_k in a time-stamp is interpreted as the starting moment of event σ_k being executed, and $t_k + f(\sigma_k)$ is the ending moment for the execution of σ_k. If $h(\sigma_q, \sigma_v) = 1$ and $q < v$, then we know that, to start executing σ_v, the execution of σ_q must have been finished because of the firing mutual exclusion between σ_q and σ_v. Thus, we have $t_q + f(\sigma_q) \leq t_v$. The execution time $\upsilon_{\mathcal{G},f,h}(s)$ is interpreted as the minimum time required to finish the execution of s. For example, suppose the time-weighted system is $\mathcal{G} = \{G_1, G_2\}$ with $\Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{b, c\}$. Suppose $f(a) = 2$, $f(b) = 3$, $f(c) = 1$, $h(a, b) = 1$, $h(b, c) = 1$ and $h(a, c) = 0$. Let $s = acb$. Then the list $w_1 = (t_a, t_c, t_b) = (0, 2, 3)$ is a time-stamp for s because $a, b \in \Sigma_1$, $b, c \in \Sigma_2$ and $t_a + f(a) = 2 < 3 = t_b$, $t_c + f(c) = 3 \leq 3 = t_b$. The list $w_2 = (t_a, t_c, t_b) = (0, 0, 2)$ is also a time-stamp for s because $t_a + f(a) = 2 \leq 2 = t_b$ and $t_c + f(c) = 1 < 2 = t_b$. We can check that, there is no other time-stamp w such that $T_{\mathcal{G},f,h}(w) < T_{\mathcal{G},f,h}(w_2)$. Thus, the execution time of s is $\upsilon_{\mathcal{G},f,h}(s) = t_b + f(b) = 2 + 3 = 5$.

Sometimes we can encode the mutual exclusion function h in the following simple way. We call $\mathcal{G}$ *asynchronous* if for every string $s = \sigma_1 \cdots \sigma_n \in L(\mathcal{G})$ and every time stamp $w = (t_1, \cdots, t_n)$ of s with respect to $(\mathcal{G}, f, h)$, we have

$$(\forall q, v \in \{1, \cdots, n\})\, q < v \wedge (\exists i \in I)\, \sigma_q \in \Sigma_i \wedge \sigma_v \in \Sigma_i \Rightarrow t_q + f(\sigma_q) \leq t_v$$

which means, in each G_i $(i \in I)$ at every time instant no more than one event is under execution. The function h is called *derivable* from $\mathcal{G}$ if

$$(\forall \sigma, \sigma' \in \cup_{i \in I} \Sigma_i)\, h(\sigma, \sigma') = 1 \iff (\exists j \in I)\, \sigma \in \Sigma_j \wedge \sigma' \in \Sigma_j$$

which means two events σ and σ' are mutually exclusive if and only if there exists one alphabet containing both events. In the above example, we can check that h is derivable from $\mathcal{G}$.

2.2 Formulation of minimum-time supervisory control problem

Given $\{\Sigma_i | i \in I\}$, for each Σ_i let $\Sigma_i = \Sigma_{i,c} \cup \Sigma_{i,uc}$, where disjoint subsets $\Sigma_{i,c}$ and $\Sigma_{i,uc}$ denote respectively the set of *controllable* events and the set of *uncontrollable* events. For

notation simplicity, from now on let $\Sigma := \cup_{i\in I}\Sigma_i$, $\Sigma_c := \cup_{i\in I}\Sigma_{i,c}$ and $\Sigma_{uc} := \Sigma - \Sigma_c$.

Definition 2.2. Given $\mathcal{G} \in \tilde{\Phi}(\{\Sigma_i | i \in I\})$ and $K \subseteq L_m(\mathcal{G})$, we say K is *controllable* with respect to $\mathcal{G}$ if $\overline{K}\Sigma_{uc} \cap L(\mathcal{G}) \subseteq \overline{K}$. When $\mathcal{G}$ is a singleton, say $\mathcal{G} = \{G \in \phi(\Sigma)\}$, then we simply say K is controllable with respect to G. □

The concept of controllability can be extended to time-weighted systems. For a time-weighted system $(\mathcal{G}, f, h) \in \Phi(\{\Sigma_i | i \in I\})$ and $K \subseteq L_m(\mathcal{G})$, we say K is *controllable* with respect to $(\mathcal{G}, f, h)$ if K is controllable with respect to $\mathcal{G}$. Recall that, in the standard Ramadge-Wonham control paradigm there are two basic assumptions: (1) the duration of firing each event is zero; (2) the firings of different events must be sequentially ordered, namely no more than one event can be fired at each time instance. When these two assumptions are satisfied, each requirement $E \in \phi(\Delta)$ with $\Delta \subseteq \cup_{i\in I}\Sigma_i$ can be interpreted as specifying the sequential orders of event firings. When each event has a nonzero firing duration, the firings of two different events may overlap with each other. Thus, none of those basic assumptions holds, which suggests that we should provide a new interpretation of a requirement $E \in \phi(\Delta)$ before we can talk about supervisory control. Given a string, say $ab \in L_m(E)$, we can interpret it in two ways: (1) the *firing moment* of a, which is defined as the moment that a starts to be fired, must precede the firing moment of b; or (2) the firing moment of b is after the moment that a finishes its firing. Fortunately, with the help of the mutual exclusion function h, we do not need to distinguish these two different scenarios. We always interpret E in the first way, but set $h(a, b) = 0$ for scenario (1), and $h(a, b) = 1$ for scenario (2).

Given a time-weighted plant $(\mathcal{G}, f, h) \in \Phi(\{\Sigma_i | i \in I\})$ and a requirement $\mathcal{E} := \{E_j \in \phi(\Delta_j) | \Delta_j \subseteq \cup_{i\in I}\Sigma_i \wedge j \in J\} \in \tilde{\Phi}(\{\Delta_j | j \in J\})$, let

$$\mathcal{C}(\mathcal{G}, \mathcal{E}) := \{K \subseteq L_m(\mathcal{G}) || L_m(\mathcal{E}) | K \text{ is controllable with respect to } \mathcal{G}\}$$

be the collection of all sublanguages of $L_m(\mathcal{G})||L_m(\mathcal{E})$ which are controllable with respect to $\mathcal{G}$. Sometimes we call K a controllable sublanguage of $\mathcal{G}$ under $\mathcal{E}$. Let

$$\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) := \{K \in \mathcal{C}(\mathcal{G}, \mathcal{E}) |\, \omega(\mathcal{G}, f, h, K) < \infty\}$$

be the collection of all controllable sublanguages of $\mathcal{G}$ under the requirement $\mathcal{E}$ such that their makespans are finite. We call each $K \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$ a *finite-makespan controllable sublanguage* of $(\mathcal{G}, f, h)$ under $\mathcal{E}$. It is possible that, $\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) = \varnothing$. Because

$$\min_{\sigma\in\Sigma} f(\sigma) > 0,$$

we can derive that, for all $K \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$ the set

$$\{K' \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) | \omega(\mathcal{G}, f, h, K') \leq \omega(\mathcal{G}, f, h, K)\}$$

is finite. Thus, there exists $K^* \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$ such that

$$(\forall K \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}))\, \omega(\mathcal{G}, f, h, K^*) \leq \omega(\mathcal{G}, f, h, K)$$

Since controllability is closed under language union, we can check that, there exists $\hat{K}^* \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$ such that, for all $K \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$ the following hold,

1. $\omega(\mathcal{G}, f, h, \hat{K}^*) \leq \omega(\mathcal{G}, f, h, K)$
2. $\omega(\mathcal{G}, f, h, K) = \omega(\mathcal{G}, f, h, \hat{K}^*) \Rightarrow K \subseteq \hat{K}^*$

We call $\hat{K}^*$ the *supremal minimum-time controllable sublanguage of* $(\mathcal{G}, f, h)$ *under* $\mathcal{E}$, denoted as $\sup\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$. For notation simplicity, from now on given a requirement $\mathcal{E} \in \tilde{\Phi}(\{\Delta_j | j \in J\})$, we assume that $\Delta_j \subseteq \Sigma$. The supervisor synthesis problem is stated as follows:

Problem 2.3. Given a time-weighted system $(\mathcal{G}, f, h) \in \Phi(\{\Sigma_i | i \in I\})$ and a requirement $\mathcal{E} \in \tilde{\Phi}(\{\Delta_j | j \in J\})$, how to compute $\sup\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$? □

Before we discuss how to solve the above problem in the next section, we would like to describe how to implement an existing supremal minimum-time controllable sublanguage by using an appropriate timed supervisory control map. This is important for the application purpose.

2.3 Timed supervisory control map

Given a time-weighted plant $(\mathcal{G}, f, h) \in \Phi(\{\Sigma_i | i \in I\})$, a *finite run* of $(\mathcal{G}, f, h)$ is a finite sequence of 2-tuples $\mu = (\sigma_1, t_1) \cdots (\sigma_n, t_n) \in (\Sigma \times \mathbb{R}^+)^*$, where $s = \sigma_1 \cdots \sigma_n \in L(\mathcal{G})$ and $(t_1, \cdots, t_n)$ is a nondecreasing time sequence. From now on we use $\mathcal{S}(\mu)$ to denote $\sigma_1 \cdots \sigma_n$, $\mathcal{S}(\mu)^\downarrow$ for σ_n and $\mathcal{W}(\mu)$ for t_n. When $\mu = \epsilon$, we simply let $\mathcal{S}(\epsilon) := \epsilon$, $\mathcal{S}(\epsilon)^\downarrow := \epsilon$ and $\mathcal{W}(\epsilon) = 0$. We use $\mathcal{P}_\sigma(\mu)$ to denote a finite run μ', which is a prefix subrun of μ, i.e. $\mu' \leq \mu$, and $\mathcal{S}(\mu')^\downarrow = \sigma$, and for all $\mu'' \leq \mu$, if $\mu' \leq \mu''$ and $\mu' \neq \mu''$ then we have $\mathcal{S}(\mu'')^\downarrow \neq \sigma$. In other words, $\mathcal{P}_\sigma(\mu)$ is a prefix subrun of μ, whose last event is σ and cannot be extended into another subrun of μ whose last event is σ. The *timed closed behavior* of $\mathcal{G}$, denoted as $L^t(\mathcal{G})$, is the collection of all possible finite runs, and the *timed marked behavior* of $\mathcal{G}$, denoted as $L^t_m(\mathcal{G})$, is the collection of all finite runs $\mu \in L^t(\mathcal{G})$ such that $\mathcal{S}(\mu) \in L_m(\mathcal{G})$.

Let $\mathcal{E} \in \tilde{\Phi}(\{\Delta_j | j \in J\})$ be a requirement, and suppose K is a controllable sublanguage of $(\mathcal{G}, f, h) \in \Phi(\{\Sigma_i | i \in I\})$ under $\mathcal{E}$. We say h is *control compatible* if

$$(\forall \sigma, \sigma' \in \Sigma)\, h(\sigma, \sigma') = 1 \Rightarrow [(\exists i \in I)\, \sigma, \sigma' \in \Sigma_i] \vee [\sigma \in \Sigma_c \wedge \sigma' \in \Sigma_c]$$

In other words, a control compatible function h imposes mutual exclusion on two events, if either there exists one alphabet contains both events or both events are controllable. If h is not control compatible, then there may not exist timed supervisory control that achieves K and in the mean time respects the event mutual exclusion imposed by h. To see this, suppose we have two components G_1 and G_2, whose controllable alphabets are empty. Suppose the requirement $\mathcal{E}$ allows all possible behavior of $\mathcal{G} = \{G_1, G_2\}$. Clearly, $K = L_m(\mathcal{G})$. But if h forces two uncontrollable events to be mutual exclusive, then clearly, there is no timed supervisory control to achieve this. For each $\mu \in L^t(\mathcal{G})$ let $A(\mu) := \{\sigma \in \Sigma | \mathcal{S}(\mu)\sigma \in \overline{K}\}$. Clearly, $A(\mu)$ is the set of all events that are allowed to be fired after μ. For each time instant $\tau \in \mathbb{R}^+$, let

$$\zeta(\mu, \tau) := \{\mu' \leq \mu | \mu' \neq \epsilon \wedge \mathcal{W}(\mu') \leq \tau \wedge \mathcal{W}(\mu') + f(\mathcal{S}(\mu')^\downarrow) > \tau\}$$

be the collection of all *active* prefixed subruns of μ, whose last events are fired before τ but have not been finished by τ. We define the following specific timed supervisory control map $g : L^t(\mathcal{G}) \times \mathbb{R}^+ \to 2^\Sigma$ with respect to K^*, where for each $\mu \in L^t(\mathcal{G})$ and $\tau \in \mathbb{R}^+$, let

$$g(\mu, \tau) := \begin{cases} \{\sigma \in A(\mu) - \Sigma_{uc} | (\forall \mu' \in \zeta(\mu, \tau))\, h(\mathcal{S}(\mu')^\downarrow, \sigma) = 0\} \cup \Sigma_{uc} & \text{if } \mathcal{S}(\mu) \in \overline{K} \\ \Sigma & \text{otherwise} \end{cases}$$

We can interpret g as follows. For every event $\sigma \in \Sigma$, if either it is uncontrollable, or it is in $A(\mu) - \Sigma_{uc}$ such that there is no event in μ that is mutually exclusive with σ and is still under execution at the instance τ, then σ is allowed by g at τ. The *closed behavior* of $(\mathcal{G}, f, h)$ under g, denoted as $L^t(g/\mathcal{G})$, is defined as follows:

1. $\epsilon \in L^t(g/\mathcal{G})$,
2. $(\forall \mu \in L^t(g/\mathcal{G}))(\forall(\sigma,\tau) \in \Sigma \times \mathbb{R}^+)\, \mu(\sigma,\tau) \in L^t(\mathcal{G}) \wedge \sigma \in g(\mu,\tau) \Rightarrow \mu(\sigma,\tau) \in L^t(g/\mathcal{G})$,
3. All strings of $L^t(g/\mathcal{G})$ are obtained from Steps (1) and (2).

We call $L^t_m(g/\mathcal{G}) := L^t(g/\mathcal{G}) \cap L^t_m(\mathcal{G})$ the *marked behavior* of $(\mathcal{G}, f, h)$ under g. Let

$$L^t_m(g/\mathcal{G}, s) := \{\mu \in L^t_m(g/\mathcal{G}) | \mathcal{S}(\mu) = s\}$$

Clearly, for each $s \in K$ there exists a finite run $\mu \in L^t(g/\mathcal{G})$ such that $\mathcal{S}(\mu) = s$. Let

$$V(g/\mathcal{G}, f, h, K) := \max_{s \in K} \min_{\mu \in L^t_m(g/\mathcal{G}, s)} \max_{\mu' \leq \mu} \mathcal{W}(\mu') + f(\mathcal{S}(\mu')^{\downarrow})$$

For any $\mu \in L^t(g/\mathcal{G})$, in practical situations, computing $g(\mu)$ always takes nonzero time and the firing of $\sigma \in g(\mu)$ also starts later than the moment of $g(\mu)$ being computed. We say g is *ideal* if for all $\mu(\sigma,\tau) \in L^t(g/\mathcal{G})$, we have

$$\tau = \max_{\sigma' \in \Sigma:\, h(\sigma,\sigma')=1} \mathcal{W}(\mathcal{P}_{\sigma'}(\mu)) + f(\sigma')$$

namely computing $g(\mu,\tau)$ takes no time and every event allowed by $g(\mu,\tau)$ starts to fire immediately whenever it is eligible. We have the following result.

Theorem 2.4. Given a time-weighted system $(\mathcal{G}, f, h) \in \Phi(\{\Sigma_i | i \in I\})$ and a requirement $\mathcal{E} \in \tilde{\Phi}(\{\Delta_j | j \in J\})$, let $K \in \mathcal{C}(\mathcal{G}, \mathcal{E})$ and suppose h is control compatible and $\mathcal{G}$ is asynchronous. Then (1) for every finite run $\mu = (\sigma_1, t_1) \cdots (\sigma_n, t_n) \in L^t(g/\mathcal{G})$, the time sequence $(t_1, \cdots, t_n)$ is a time stamp of $s = \sigma_1 \cdots \sigma_n$ with respect to $(\mathcal{G}, f, h)$; (2) $V(g/\mathcal{G}, f, h, K) \geq \omega(\mathcal{G}, f, h, K)$; (3) if g is ideal, then $V(g/\mathcal{G}, f, h, K) = \omega(\mathcal{G}, f, h, K)$. □

Proof: Suppose h is control compatible and $\mathcal{G}$ is asynchronous. (1) We need to show that

$$(\forall q, v \in \{1, \cdots, n\})\, q < v \wedge h(\sigma_q, \sigma_v) = 1 \Rightarrow t_q + f(\sigma_q) \leq t_v$$

Suppose this condition does not hold. Let (q, v) with $q < v$ be the first pair that violates the condition, namely $h(\sigma_q, \sigma_v) = 1$ and $t_q + f(\sigma_q) > t_v$. The term 'first pair' means that, for any other pair (q', v') that violates the condition, we have either $q < q'$ or $q = q'$ and $v < v'$. Since h is control compatible, we have two cases to consider.
Case 1: $\sigma_q, \sigma_v \in \Sigma_c$. Let τ_v be the decision time instant, where $g(\mu_v, \tau_v)$ is computed with $\mu_v = (\sigma_1, t_1) \cdots (\sigma_v, t_v)$ and $t_q \leq \tau_v \leq t_v$. Then by the definition of g we know that, for all $\mu' \leq \mu_v$, if $\mathcal{W}(\mu') \leq \tau_v$ and $\mathcal{W}(\mu') + f(\mathcal{S}(\mu')^{\downarrow}) > \tau_v$, then $h(\mathcal{S}(\mu')^{\downarrow}, \sigma_v) = 0$. Clearly, $\mu_q \leq \mu_v$ and $\mathcal{W}(\mu_q) \leq \tau_v$. Since $t_q + f(\sigma_q) > t_v$, we have $\mathcal{W}(\mu_q) + f(\mathcal{S}(\mu_q)^{\downarrow}) > t_v \geq \tau_v$. But this is a contradiction because $h(\sigma_q, \sigma_v) = 1$ implies that $\mathcal{W}(\mu_q) + f(\mathcal{S}(\mu_q)^{\downarrow}) \leq \tau_v$.
Case 2: there exists Σ_i such that $\sigma_q, \sigma_v \in \Sigma_i$. Since $\mathcal{G}$ is asynchronous, we have that $t_q + f(\sigma_q) \leq t_v$. But this contradicts the assumption that $t_q + f(\sigma_q) > t_v$. Since in either case we obtains a contradiction, the time sequence is a time stamp of s with respect to $(\mathcal{G}, f, h)$.
(2) Since $L^t_m(g/\mathcal{G}, s) \neq \varnothing$ for all $s \in K$, and by (1) each time sequence allowed in $L^t(g/\mathcal{G})$ is a time stamp, we can derive that

$$\upsilon_{\mathcal{G},f,h}(s) \leq \min_{\mu \in L^t_m(g/\mathcal{G}, s)} \max_{\mu' \leq \mu} \mathcal{W}(\mu') + f(\mathcal{S}(\mu')^{\downarrow})$$

which implies $V(g/\mathcal{G}, f, h, K) \geq \omega(\mathcal{G}, f, h, K)$.
(3) Let $s = \sigma_1 \cdots \sigma_n \in K$ with $(t_1, \cdots, t_n)$ being the time-stamp with respect to $\mathcal{G}$ such that

$$\upsilon_{\mathcal{G},f,h}(s) = \max_i(t_i + f(\sigma_i))$$

We want to show that, $\mu = (\sigma_1, t_1) \cdots (\sigma_n, t_n) \in L_m^t(g/\mathcal{G})$.
Clearly, $\sigma_1 \cdots \sigma_n \in L_m(\mathcal{G})$. Suppose $\mu = (\sigma_1, t_1) \cdots (\sigma_n, t_n) \notin L_m^t(g/\mathcal{G})$. Then there exists q such that, $\mu' = (\sigma_1, t_1) \cdots (\sigma_q, t_q) \in L^t(g/\mathcal{G})$ but $\mu'(\sigma_{q+1}, t_{q+1}) \notin L^t(g/\mathcal{G})$. Therefore, $\sigma_{q+1} \notin g(\mu', \tau_{q+1})$ with $t_q \leq \tau_{q+1} \leq t_{q+1}$. Since τ_{q+1} can be any value in that range, we pick $\tau_{q+1} = t_{q+1}$. This means

$$(\exists \mu'' \in \zeta(\mu', t_{q+1}))\, h(\mathcal{S}(\mu''), \sigma_{q+1}) = 1$$

Since $\mu'' \in \zeta(\mu', t_{q+1})$, we have that

$$\mathcal{W}(\mu'') \leq t_{q+1} \wedge \mathcal{W}(\mu'') + f(\mathcal{S}(\mu'')^{\downarrow}) > t_{q+1}$$

Since g is ideal, we have that the firing moment $\hat{t}_{q+1}$ for σ_{q+1} satisfies the following:

$$\hat{t}_{q+1} = \max_{\sigma' \in \Sigma:\, h(\sigma_{q+1}, \sigma') = 1} \mathcal{W}(\mathcal{P}_{\sigma'}(\mu')) + f(\sigma')$$

By the definition of time stamp, we have $t_{q+1} \geq \hat{t}_{q+1}$. Since $h(\mathcal{S}(\mu''), \sigma_{q+1}) = 1$ and $\mu'' \leq \mu'$, we get that

$$\mathcal{W}(\mu'') + f(\mathcal{S}(\mu'')^{\downarrow}) \leq \hat{t}_{q+1} \leq t_{q+1}$$

But this contradicts the assumption that $\mathcal{W}(\mu'') + f(\mathcal{S}(\mu'')^{\downarrow}) > t_{q+1}$. Thus, we have $\mu \in L^t(g/\mathcal{G})$. Since $\mu \in L_m(\mathcal{G})$, we have $\mu \in L_m^t(g/\mathcal{G})$. Since g is ideal, we have

$$\upsilon_{\mathcal{G},f,h}(s) = \max_i(t_q + f(\sigma_q)) \geq \min_{\mu \in L_m^t(g/\mathcal{G}, s)} \max_{\mu' \leq \mu} \mathcal{W}(\mu') + f(\mathcal{S}(\mu')^{\downarrow})$$

which implies $V(g/\mathcal{G}, f, h, K) \leq \omega(\mathcal{G}, f, h, K)$. By (1), we have $V(g/\mathcal{G}, f, h, K) \geq \omega(\mathcal{G}, f, h, K)$. Thus, $V(g/\mathcal{G}, f, h, K) = \omega(\mathcal{G}, f, h, K)$. The theorem follows. ■

Theorem 2.4 indicates that, when h is control compatible and $\mathcal{G}$ is asynchronous, the supervisory control map g respects the event mutual exclusion imposed by h in the sense that, the time sequence of every finite run of the supervised system $g/\mathcal{G}$ is a time stamp. If additionally g is ideal, then the makespan of the minimum-time controllable sublanguage can be achieved by applying the proposed supervisory control map g. Although in practical applications g is rarely ideal, as long as we can speed up computation of $g(\mu, \tau)$ and initiation the firing of $\sigma \in g(\mu, \tau)$, the execution time of a finite run of $\mathcal{G}$ under the supervision of g can always be shortened.

To illustrate the aforementioned control strategy, we present a simple example. Suppose $\mathcal{G} = \{G_1 \in \phi(\Sigma_1), G_2 \in \phi(\Sigma_2)\}$, where $\Sigma_1 = \{a\}$, $\Sigma_2 = \{b, c, d\}$, $L_m(G_1) = \{a\}$, $L(G_1) = \overline{L_m(G_1)}$, $L_m(G_2) = \{bcd\}$ and $L(G_2) = \overline{L_m(G_2)}$. The time-weighted function f is defined as $f(a) = 2$, $f(b) = f(c) = f(d) = 1$, and the mutual exclusion function h is defined as $h(a, a) = h(b, b) = h(c, c) = h(d, d) = h(a, b) = h(c, d) = 1$, $h(b, c) = h(a, c) = 0$. The requirement is $\mathcal{E} = \{E_1 = \{ac\} \in \phi(\{a, c\})\}$. Suppose $\Sigma_c = \{a, b\}$ and $\Sigma_{uc} = \{c, d\}$. Clearly, h is control compatible. Then by using the standard procedure for supervisor synthesis [22] we can get that, the supremal controllable sublanguage of $\mathcal{G}$ under $\mathcal{E}$) is $K = \{abcd\}$. The control sequence is computed as follows. At the initial instant $\tau = 0$, we have

$$\zeta(\epsilon, 0) = \varnothing,\ A(\epsilon) = \{a\}$$

Therefore, $g(\epsilon, 0) = \{a\} \cup \Sigma_{uc} = \{a, c, d\}$, which means a and c are allowed at $\tau = 0$. Suppose a is fired at $t_a = 0.1$. Then at $\tau = 0.5$ we have

$$\zeta((a, 0.1), 0.5) = \{(a, 0.1)\},\ A((a, 0.1)) = \{b\}$$

Since $h(a, b) = 1$, we get $g((a, 0.1), 0.5) = \Sigma_{uc} = \{c, d\}$. We can see that, although $b \in A((a, 0.1))$, it cannot be fired because $(a, 0.1)$ is still active at $\tau = 0.5$, which prevents b to be fired, owing to mutual exclusion. When $\tau = 2.1$, we have

$$\zeta((a, 0.1), 2.1) = \varnothing,\ A((a, 0.1)) = \{b\}$$

from which we have $g((a, 0.1), 2.1) = \{b\} \cup \Sigma_{uc} = \{b, c, d\}$. Suppose b is fired at $t_b = 2.2$. Then for all $\tau \geq t_b$ we have $g((a, 0.1)(b, 2.2), \tau) = \{c, d\}$. Because $h(b, c) = 0$, the firing of c can be at any moment after t_b. Suppose c is fired at $t_c = 3$. Then at $\tau = 3.2$ we have $g((a, 0.1)(b, 2.2)(c, 3), 3.2) = \{c, d\}$. If $\mathcal{G}$ is not asynchronous, then d can be fired at any moment after $\tau = 3.2$, which clearly violates the mutual exclusion $h(c, d) = 1$. This shows that, without the condition of $\mathcal{G}$ being asynchronous, Theorem 2.4 may not hold. Suppose $\mathcal{G}$ is asynchronous, then d can only be fired after c is done. Suppose $t_d = 4$. Then we have a time sequence $(t_a = 0.1, t_b = 2.2, t_c = 3, t_d = 4)$ for $s = abcd$. Clearly, the time sequence is a time stamp of s with respect to $(\mathcal{G}, f, h)$. The final execution time of s is

$$\nu_{\mathcal{G},f,h}(s) = \max\{t_a + f(a), t_b + f(b), t_c + f(c), t_d + f(d)\} = 5$$

If g is ideal, then the smallest value for t_a is 0, for t_b is 2, for t_c is 2 and for t_d is 3. The corresponding execution time of s is 4.

3 Computing Supremal Minimum-Time Nonblocking Supervisors

In this section we first briefly introduce the concepts of heaps-of-pieces. Then we present a heaps-of-pieces-based algorithm to compute the supremal minimum-time nonblocking supervisor.

3.1 Concepts of heaps-of-pieces

Recall that, in the previous section we define a map $\upsilon_{\mathcal{G},f,h}(s) := \min_{w \in \Theta_{\mathcal{G},f,h}(s)} T_{\mathcal{G},f,h}(w)$, whose value is the execution time of s in a time-weighted automaton $(\mathcal{G}, f, h)$. There are several ways to compute $\upsilon_{\mathcal{G},f,h}(s)$, one of which is by using the theory of heaps-of-pieces [27].

The name "heaps-of-pieces" comes from the following informal interpretation [9]. Imagine that there is a horizontal axis with a finite number of slots. We have a finite number of geometric objects called *pieces*, each of which is a solid (possibly not connected) "block" occupying some of the slots, with staircase-shaped upper and lower contours. With an ordered sequence of pieces, we associate a heap by piling up the pieces, starting from a horizontal ground. This piling occurs in the intuitive way: a piece is only subject to vertical translations and occupies the lowest possible position, provided it is above the

ground and the pieces previously piled up. A formal definition is provided below.

Definition 3.1. A *heap model* is a 5-tuple $\mathcal{H} = (\mathcal{T}, \mathcal{R}, R, l, u)$, where

1. $\mathcal{T}$ is a finite set whose elements are called *pieces*;
2. $\mathcal{R}$ is a finite set whose elements are called *slots*;
3. $R : \mathcal{T} \to 2^{\mathcal{R}}$ associates a piece with a subset of slots. We assume $R(a) \neq \varnothing$ for all $a \in \mathcal{T}$;
4. $l : \mathcal{T} \times \mathcal{R} \to \mathbb{R}^+ \cup \{0, -\infty\}$ gives the height of the lower contour of a piece;
5. $u : \mathcal{T} \times \mathcal{R} \to \mathbb{R}^+ \cup \{0, -\infty\}$ gives the height of the upper contour of a piece.

By convention, $l(a, r) = u(a, r) = -\infty$ if $r \notin R(a)$, and $\min_{r \in R(a)} l(a, r) = 0$. □

Each slot $r \in \mathcal{R}$ is interpreted as a resource. For a piece $a \in \mathcal{T}$ and a slot $r \in R(a)$, we interpret the difference $u(a, r) - l(a, r)$ as the *occupation time* of a over r. In the simplest case, we have $l(a, r) = 0$ for all $a \in \mathcal{T}$ and $r \in R(a)$, namely, each piece is a rectangular bar, not necessarily connected. Given a string $s = a_1 \cdots a_k \in \mathcal{T}^*$ with $k \in \mathbb{N}$, we associate with each a_q $(q = 1, \cdots, k)$ a nonnegative real t_q. We say s is a *heap* with respect to $w = (t_1, \cdots, t_k)$, if

$$(\forall q, v \in \{1, \cdots, k\})\, q < v \Rightarrow (\forall r \in R(a_q) \cap R(a_v))\, t_q + u(a_q, r) \leq t_v$$

In other words, the piece a_v, which appears after a_q, should pile upon a_q. We call w a *hight-stamp* of s. Suppose $\mathcal{R} = \{r_1, \cdots, r_n\}$. The *upper contour* of s with respect to w is a row vector $x_{\mathcal{H}}(s, w) = (x_1, \cdots, x_n)$, where

$$(\forall q \in \{1, \cdots, n\})\, x_q = \max_{v \in \{1, \cdots, k\}} t_v + u(a_v, r_q)$$

The *height* of s with respect to w is

$$y_{\mathcal{H}}(s, w) := \max_{q \in \{1, \cdots, n\}} x_q$$

Suppose Ξ_s is the collection of all hight-stamps of s. Then the *height* of s is

$$y_{\mathcal{H}}(s) := \min_{w \in \Xi_s} y_{\mathcal{H}}(s, w)$$

If we interpret t_i in a hight-stamp w as the firing moment of the piece a_i, then the height $y_{\mathcal{H}}(s)$ corresponds to the minimum time that is required to complete s, which requires a piece to be executed as soon as all relevant resources are available.

Definition 3.2. The (max,+) semiring $\mathbb{R}_{max}$ is the set $\mathbb{R} \cup \{-\infty\}$, equipped with the operation max, written additively (i.e. $a \oplus b = \max\{a, b\}$), and the usual sum, written multiplicatively (i.e. $a \otimes b = a + b$). In this semiring, the zero element $\mathbf{0}$ is $-\infty$, and the unit element $\mathbf{1}$ is 0. □

The matrix operations are induced by the semiring structure as follows. For matrixes A and B of appropriate dimentions,

$$(A \oplus B)_{qv} := A_{qv} \oplus B_{qvj} = \max\{A_{qv}, B_{qv}\}$$

and

$$(A \otimes B)_{qv} := \oplus_k (A_{qk} \otimes B_{kv}) = \max_k (A_{qk} + B_{kv})$$

For a scalar $a \in \mathbb{R} \cup \{-\infty\}$, $(a \otimes A)_{ij} := a \otimes A_{ij} = a + A_{ij}$. From now on we omit the $\otimes$ sign, and directly use AB to denote $A \otimes B$. By the definition of semiring we get that, the matrix multiplication is associative, i.e. $(AB)C = A(BC)$. Let $\mathbb{R}_{max}^{\mathcal{R},\mathcal{R}}$ be the collection of all matrices, whose dimensions are $|\mathcal{R}| \times |\mathcal{R}|$. Let $\mathcal{M} : \mathcal{T}^* \to \mathbb{R}_{max}^{\mathcal{R},\mathcal{R}}$, where

$$(\forall a \in \mathcal{T})\, \mathcal{M}(a)_{qv} := \begin{cases} \mathbf{1} & \text{if } q = v \text{ and } q \notin R(a) \\ u(a,v) - l(a,q) & \text{if } q, v \in R(a) \\ \mathbf{0} & \text{otherwise} \end{cases}$$

and for all $s = s's'' \in \mathcal{T}^*$, $\mathcal{M}(s) := \mathcal{M}(s')\mathcal{M}(s'')$ with $\mathcal{M}(\epsilon)$ being the unit matrix $\mathbf{I}$ (i.e. all diagonal entries are $\mathbf{1}$ and all other entries are $\mathbf{0}$). Thus, if $s = a_1 \cdots a_k$, then $\mathcal{M}(s) = \mathcal{M}(a_1) \cdots \mathcal{M}(a_k)$. Clearly, $\mathcal{M}$ is a morphism between $\mathcal{T}^*$ and $\mathbb{R}_{max}^{\mathcal{R},\mathcal{R}}$. We say $\mathcal{M}$ is *induced* from $\mathcal{H}$. Let $\mathbf{1}_{\mathcal{R}}$ be the $1 \times |\mathcal{R}|$ dimension column vector, whose entries are all equal to $\mathbf{1}$. We use $\mathbf{1}_{\mathcal{R}}^t$ to denote the transpose of $\mathbf{1}_{\mathcal{R}}$, which is a row vector. From [9] we have the following,

$$(\forall s \in \mathcal{T}^*)\, y_{\mathcal{H}}(s) \quad = \quad \mathbf{1}_{\mathcal{R}}^t \mathcal{M}(s) \mathbf{1}_{\mathcal{R}} \tag{1}$$

Once a heap model $\mathcal{H}$ is given, the morphism $\mathcal{M}$ is uniquely determined. Thus, the height of each string $s \in \mathcal{T}^*$ can be computed. We call $x_{\mathcal{H}}(s) := \mathbf{1}_{\mathcal{R}}^t \mathcal{M}(s)$ the *upper contour* of s.

Given a time-weighted plant $(\mathcal{G}, f, h) \in \Phi(\{\Sigma_i | i \in I\})$, we first build an undirected graph $\mathrm{Gr}(\mathcal{G}, f, h) = (\mathrm{Ver}, \mathrm{Edg})$, where $\mathrm{Ver} := \Sigma$ is the vertex set and Edg is the edge set such that,

$$(\forall \sigma, \sigma' \in \Sigma)\, (\sigma, \sigma') \in \mathrm{Edg} \iff h(\sigma, \sigma') = 1 \wedge \sigma \neq \sigma'$$

which means we only use an edge to connect two different events, which are mutually exclusive. Although $h(\sigma, \sigma) = 1$ for all $\sigma \in \Sigma$, we do not want to add selfloops in the graph. A subgraph of $\mathrm{Gr}(\mathcal{G}, f, h)$ is *complete* if every pair of vertices in the subgraph are connected by an edge. A complete subgraph is maximal if it is not contained in any larger complete subgraph. A *clique cover* of $\mathrm{Gr}(\mathcal{G}, f, h)$ is a collection of (maximal) complete subgraphs such that every edge of $\mathrm{Gr}(\mathcal{G}, f, h)$ is contained in at least one (maximal) complete subgraph. Such a clique cover need not be unique. It has been shown in [18] that, finding a clique cover whose size is no more than a given value is NP-hard, which implies that, finding a clique cover with the minimum size is also NP-hard. Let $\Lambda(\mathcal{G}, f, h)$ be a clique cover of $\mathrm{Gr}(\mathcal{G}, f, h)$. For each (maximal) complete subgraph $\lambda \in \Lambda(\mathcal{G}, f, h)$, we use Ver_λ to denote its vertex set. We build the following heap model $\mathcal{H} = (\mathcal{T}, \mathcal{R}, R, l, u)$ associated with $(\mathcal{G}, f, h)$, where

1. $\mathcal{T} := \Sigma$ and $\mathcal{R} := \Lambda(\mathcal{G}, f, h)$
2. $R : \mathcal{T} \to 2^{\mathcal{R}} : \sigma \mapsto R(\sigma) := \{\lambda \in \mathcal{R} | \sigma \in \mathrm{Ver}_\lambda\}$
3. $l(\sigma, r) = 0$, and $u(\sigma, r) = \begin{cases} f(\sigma) & \text{if } r \in R(\sigma) \\ 0 & \text{otherwise} \end{cases}$

In this heap model, each resource $r \in \mathcal{R}$ is a maximal complete subgraph of $\mathrm{Gr}(\mathcal{G}, f, h)$, meaning that any pair of events in the vertex set of r are mutually exclusive.

The aforementioned way of computing $y_{\mathcal{H}}(s)$ cannot be directly used to compute the execution time of s because the latter requires that the time stamp should be a nondecreasing list, which may not hold in the formulation of $y_{\mathcal{H}}(s)$. We will see this shortly

in Example 1. To prevent such an undesirable situation from happening, we propose the following algorithm to compute the execution time of s:

1. Input: a heap model $\mathcal{H}_q$ based on $\mathcal{G}$ and $\mathcal{E}$, and $s = \sigma_1 \cdots \sigma_n \in \Sigma^*$.
2. Initialization:
   (a) $C_1 := \mathbf{1}_{\mathcal{R}}^t \mathcal{M}(\sigma_1)$
   (b) For each σ_q ($q = 1, \cdots, n$), define two $|\mathcal{R}|$-dimensional row vectors $\tilde{Q}_q$ and $\check{Q}_q$, where
   $$(\forall r \in \mathcal{R})\, \tilde{Q}_{q,r} := \begin{cases} f(\sigma_q) & \text{if } r \in R(\sigma_q) \\ \mathbf{1} & \text{otherwise} \end{cases}$$
   and
   $$(\forall r \in \mathcal{R})\, \check{Q}_{q,r} := \begin{cases} -f(\sigma_q) & \text{if } r \in R(\sigma_q) \\ \mathbf{0} & \text{otherwise} \end{cases}$$
3. For each $k = 2, 3, \cdots, n-1$
   (a) $\hat{C}_k := C_{k-1}\mathcal{M}(\sigma_k)$
   (b) $C_k := \hat{C}_k \oplus ((\hat{C}_k \check{Q}_k^t)\tilde{Q}_k)$

   or equivalently, $C_k := C_{k-1}\mathcal{M}(\sigma_k)[\mathbf{I} \oplus (\check{Q}_k^t \tilde{Q}_k)]$.
4. $\hat{y}_{\mathcal{H}}(s) := C_{n-1}\mathcal{M}(\sigma_n)[\mathbf{I} \oplus (\check{Q}_k^t \tilde{Q}_k)]\mathbf{1}_{\mathcal{R}}$ and the upper contour is $\hat{x}_{\mathcal{H}}(s) = C_{n-1}\mathcal{M}(\sigma_n)$ □

In Step (3.b) the term $(\hat{C}_k \check{Q}_k^t)$ is used to determine the height of the bottom edge of the piece associated with σ_k on the heap, which is interpreted as the earliest possible firing moment of σ_k. Then the term $\hat{C}_k \oplus ((\hat{C}_k \check{Q}_k^t)\tilde{Q}_k)$ is used to set the upper contour before the piece associated with σ_{k+1} can be piled on. We can check that, the minimum height of such a contour is at least the same as the height of the bottom edge of the piece associated with σ_k, namely the bottom edge of the piece associated with σ_{k+1} will not be lower than the bottom edge of the piece associated with σ_k - in other words, the firing moment of σ_{k+1} cannot be earlier than the firing moment of σ_k. This will guarantee that, the firing moments of those events in the heap can form an order specified by the original string $s = \sigma_1 \cdots \sigma_n$. A formal argument is provided as follow. Let $t_{k+1} := \hat{C}_{k+1}\check{Q}_{k+1}^t$, which is interpreted as the firing moment of event σ_{k+1}. Then we have

$$\begin{aligned} t_{k+1} &= \hat{C}_{k+1}\check{Q}_{k+1}^t \\ &= C_k \mathcal{M}(\sigma_{k+1})\check{Q}_{k+1}^t \\ &= (\hat{C}_k \oplus ((\hat{C}_k \check{Q}_k^t)\tilde{Q}_k))\mathcal{M}(\sigma_{k+1})\check{Q}_{k+1}^t \\ &= \hat{C}_k \mathcal{M}(\sigma_{k+1})\check{Q}_{k+1}^t \oplus (\hat{C}_k \check{Q}_k^t)\tilde{Q}_k \mathcal{M}(\sigma_{k+1})\check{Q}_{k+1}^t \\ &\geq \hat{C}_k \check{Q}_k^t \text{ because } \mathcal{M}(\sigma_{k+1})\check{Q}_{k+1}^t \text{ is a nonnegative column vector} \\ &= t_k \end{aligned}$$

which means the sequence of firing moments $t_1, \cdots, t_n$ is a nondecreasing sequence.

Given a nondecreasing height stamp $w = (t_1, \cdots, t_n)$, by the theory of heaps-of-pieces, we get that

$$(\forall q, v \in \{1, \cdots, n\})\, q < v \Rightarrow (\forall r \in R(\sigma_q) \cap R(\sigma_v))\, t_q + u(\sigma_q, r) \leq t_v$$

By the definition of the heap model $\mathcal{H}$, we know that, if $r \in R(\sigma_q) \cap R(\sigma_v)$, then $h(\sigma_q, \sigma_v) = 1$. Since $u(\sigma_q, r) = f(\sigma_q)$ if $r \in R(\Sigma_q)$, we have

$$(\forall q, v \in \{1, \cdots, n\})\, q < v \Rightarrow [h(\sigma_q, \sigma_v) = 1 \Rightarrow t_q + f(\sigma_q) \leq t_v]$$

or equivalently,

$$(\forall q, v \in \{1, \cdots, n\})\, q < v \wedge h(\sigma_q, \sigma_v) = 1 \Rightarrow t_q + f(\sigma_q) \leq t_v$$

which means the height stamp $(t_1, \cdots, t_n)$ is actually a time stamp of s with respect to $(\mathcal{G}, f, h)$. By using a similar argument, we can show that a time stamp of s with respect to $(\mathcal{G}, f, h)$ is actually a height stamp of s in the associated heap model $\mathcal{H}$. From this fact we can derive that

$$\upsilon_{\mathcal{G},f,h}(s) = \hat{y}_{\mathcal{H}}(s)$$

and for all $W \subseteq L_m(\mathcal{G})$ we have

$$\omega(\mathcal{G}, f, h, W) = \max_{s \in W} \hat{y}_{\mathcal{H}}(s)$$

Thus, by using the heap model we can compute the execution time of each string and the makespan of a sublanguage. This fact will be used in the next section to compute supremal minimum-time nonblocking supervisors.

We use a simple example to illustrate the aforementioned concepts and results. Suppose we have a time-weighted plant $(\mathcal{G} = \{G_1, G_2, G_3\}, f, h)$, where

$$\Sigma_1 = \{a, b\},\ \Sigma_2 = \{c\},\ \Sigma_3 = \{c\},\ f(a) = 2,\ f(b) = 3,\ f(c) = 1$$

and h is derivable from $\mathcal{G}$. Suppose $L_m(G_1) = (ab)^*$, $L_m(G_2) = c^*$ and $L_m(G_3) = c^*$. Since $h(a, b) = 1$ and $h(a, c) = h(b, c) = 0$, we can easily check that $\Lambda(\mathcal{G}, f, h)$ contains two maximal complete subgraphs r_1 and r_2 of $\mathrm{Gr}(\mathcal{G}, f, h)$, where the vertex set of r_1 is $\{a, b\}$, and the vertex set of r_2 is $\{c\}$. We build the heap model $\mathcal{H}$, where

1. $\mathcal{T} := \{a, b, c\}$ and $\mathcal{R} := \{r_1, r_2\}$
2. $R(a) = \{r_1\}$, $R(b) = \{r_1\}$, $R(c) = \{r_2\}$
3. $u(a, r_1) = f(a) = 2$, $u(a, r_2) = 0$, $l(a, r_1) = l(a, r_2) = 0$
   $u(b, r_1) = f(b) = 3$, $u(b, r_2) = 0$, $l(b, r_1) = l(b, r_2) = 0$
   $u(c, r_1) = 0$, $u(c, r_2) = f(c) = 1$, $l(c, r_1) = l(c, r_2) = 0$

Figure 1 depicts the relevant pieces. The associated morphism $\mathcal{M}$ is described as follows:

$$\mathcal{M}(a) = \begin{bmatrix} 2 & -\infty \\ -\infty & 0 \end{bmatrix},\ \mathcal{M}(b) = \begin{bmatrix} 3 & -\infty \\ -\infty & 0 \end{bmatrix},\ \mathcal{M}(c) = \begin{bmatrix} 0 & -\infty \\ -\infty & 1 \end{bmatrix}$$

From the model of $\mathcal{G}$, we know that $s = abc \in L_m(\mathcal{G}) = L_m(G_1)||L_m(G_2)||L_m(G_3)$. The execution time of s is $\hat{y}_{\mathcal{H}}(s)$, which can be computed as follows:

1. Initialization: $C_1 = \mathbf{1}^t \mathcal{M}(a) = [2\ \ 0]$, $\tilde{Q}_a = [2\ 0]$, $\check{Q}_a = [-2\ \ -\infty]$, $\tilde{Q}_b = [3\ 0]$, $\check{Q}_b = [-3\ \ -\infty]$, $\tilde{Q}_c = [0\ 1]$, $\check{Q}_c = [-\infty\ \ -1]$
2. Iterate on $k = 2$
   (a) $\hat{C}_2 = C_1 \mathcal{M}(b) = [5\ \ 0]$
   (b) $C_2 = [5\ 0] \oplus (([5\ 0] \otimes [-3\ \ -\infty]^t) \otimes [3\ \ 0]) = [5\ 2]$
3. $\hat{y}_{\mathcal{H}}(s) = C_2 \mathcal{M}(c) \mathbf{1}_{\mathcal{R}} = [5\ \ 3][0\ \ 0]^t = 5$ and $\hat{x}_{\mathcal{H}}(s) = [5\ \ 3]$

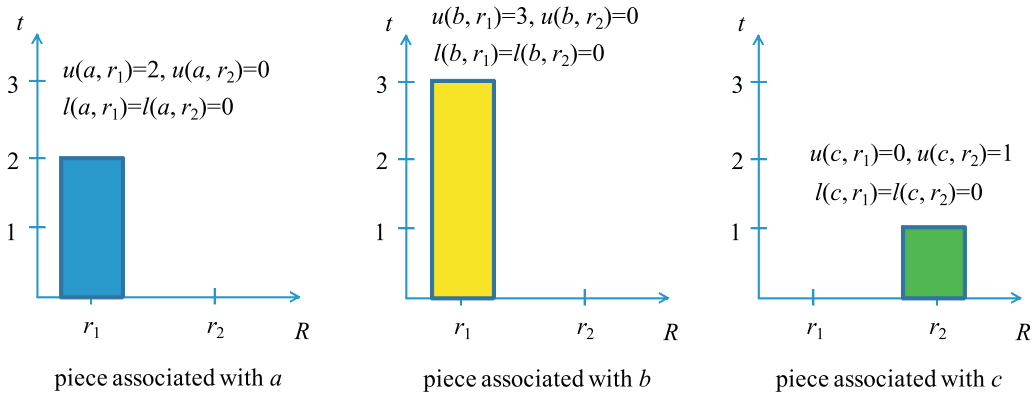

Figure 1: Example 1: Pieces for a, b and c

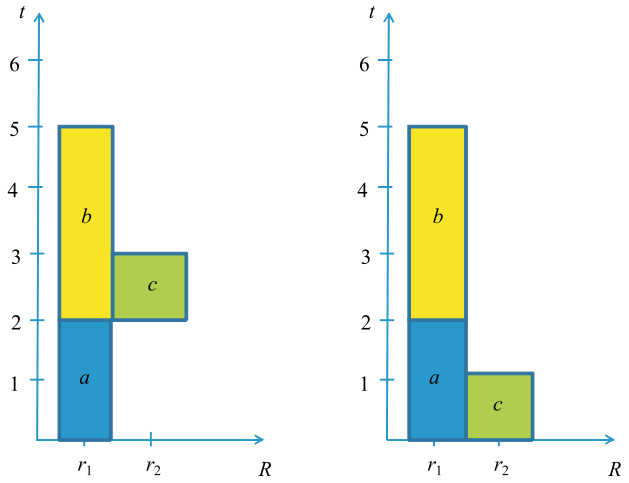

Figure 2: Example 1: Heap of $\hat{y}_{\mathcal{H}}(abc)$ (left) and heap of $y_{\mathcal{H}}(abc)$ (right)

The heap of $\hat{y}_{\mathcal{H}}(abc)$ is depicted in the left picture of Figure 2. As a comparison, we apply the standard heaps-of-pieces theory to compute $y_{\mathcal{H}}(s)$, which is shown as follows:

$$\begin{aligned} y_{\mathcal{H}}(abc) &= \mathbf{1}^t \mathcal{M}(a)\mathcal{M}(b)\mathcal{M}(c)\mathbf{1} \\ &= [0\ 0] \begin{bmatrix} 2 & -\infty \\ -\infty & 0 \end{bmatrix} \begin{bmatrix} 3 & -\infty \\ -\infty & 0 \end{bmatrix} \begin{bmatrix} 0 & -\infty \\ -\infty & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ &= [5\ 1] \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 5 \end{aligned}$$

The upper contour is $x_{\mathcal{H}}(abc) = [5\ 1]$. The heap of $y_{\mathcal{H}}(abc)$ is depicted in the right picture of Figure 2. In this example we can see the difference between $\hat{y}_{\mathcal{H}}(s)$ and $y_{\mathcal{H}}(s)$, where the former one takes into account the fact that, the starting moments of a, b and c form a nondecreasing order, but in the latter c starts firing before b, which does not reflect the actual firing order.

3.2 Computation of supremal minimum-time controllable sublanguages

We provide the following procedure to compute the supremal minimum-time controllable sublanguage $\sup\mathcal{NS}(\mathcal{G}, f, \mathcal{E})$.

Procedure for Supremal Minimum-Time Controllable Sublanguage (SMT):

1. Input:
   (a) a time-weighted plant $(\mathcal{G}, f, h) \in \Phi(\{\Sigma_i | i \in I\})$
   (b) a requirement $\mathcal{E} \in \tilde{\Phi}(\{\Delta_j | j \in J\})$
2. Initialization:
   (a) Compute $K = \sup\mathcal{C}(\mathcal{G}, \mathcal{E})$. If $K = \varnothing$ then set $K^* := \varnothing$ and go to step (6).
   (b) Let $G = \kappa(K) = (Z, \Sigma = \cup_{i\in I}\Sigma_i, \delta, z_0, Z_m)$ with $\Sigma_{uc} := \cup_{i\in I}\Sigma_{i,uc}$, $\Sigma_c = \Sigma - \Sigma_{uc}$
   (c) For each $z \in Z_m$, set
   $$\mathcal{W}_0(z) = \begin{cases} 0 & \text{if } \mu(z) \subseteq \Sigma_c \\ +\infty & \text{otherwise} \end{cases}$$
   and for each $z \in Z - Z_m$, define $\mathcal{Q}_0(z) := +\infty$
3. Iterate on $k = 1, 2, \cdots$, as follows:
   (a) For each $z \in Z$ we have
   $$\mathcal{Q}_k(z) := \begin{cases} \max_{\sigma\in\Sigma_{uc}}(f(\sigma) + \mathcal{Q}_{k-1}(\delta(z,\sigma))) & \text{if } \mu_G(z) \cap \Sigma_{uc} \neq \varnothing \\ \min\{\min_{\sigma\in\mu_G(z)}(f(\sigma) + \mathcal{Q}_{k-1}(\delta(z,\sigma))),\ \mathcal{Q}_{k-1}(z)\} & \text{if } \varnothing \neq \mu_G(z) \subseteq \Sigma_c \\ \mathcal{Q}_{k-1}(z) & \text{otherwise} \end{cases}$$
   (b) Termination when: $(\exists r \in \mathbb{N})(\forall z \in Z)\ \mathcal{Q}_{r-1}(z) = \mathcal{Q}_r(z)$
4. If $\mathcal{Q}_r(z_0) = +\infty$, $K^* := \varnothing$ and go to (6). Otherwise, let
   $$K' := \{s \in L_m(G) | \upsilon_{\mathcal{G},f,h}(s) \leq \mathcal{Q}_r(z_0)\}$$
   and $\hat{K} := \sup\mathcal{C}(G, \kappa(K'))$. Since $\hat{K}$ is finite, we construct a tree automaton S, which recognizes $L_m(\hat{K})$, i.e. $L_m(S) = L_m(\hat{K})$ and $L(S) = \overline{L_m(S)}$. Suppose $S = (Z', \Sigma, \delta', z_0', Z_m')$.
5. We perform the following iteration on S:
   (a) Initialization: for each $z \in Z'$, if $z \in Z_m'$ and $\mu_S(z) \subseteq \Sigma_c$, then set $\eta_0(z) := \upsilon_{\mathcal{G},f}(s)$, where $\delta'(z_0', s) = z$; otherwise, set $\eta_0(z) := +\infty$
   (b) Iterate on $k = 1, 2, \cdots, l$, where l is the length of the longest path in S,
   $$(\forall z \in Z')\ \eta_k(z) := \begin{cases} \max_{\sigma\in\Sigma_{uc}} \eta_{k-1}(\delta'(z,\sigma)) & \text{if } \mu_S(z) \cap \Sigma_{uc} \neq \varnothing \\ \min\{\min_{\sigma\in\mu_G(z)} \eta_{k-1}(\delta'(z,\sigma)),\ \eta_{k-1}(z)\} & \text{if } \varnothing \neq \mu_S(z) \subseteq \Sigma_c \\ \eta_{k-1}(z) & \text{otherwise} \end{cases}$$
   (c) Let $S' = (Z', \Sigma, \delta'', z_0', Z_m')$ with $\delta'' : Z' \times \Sigma \to Z'$, where for all $(z, \sigma) \in Z' \times \Sigma$,
   $$\delta''(z,\sigma) := \begin{cases} \delta'(z,\sigma) & \text{if } \delta(z,\sigma)! \text{ and } \eta_l(z) \leq \eta_l(z_0') \text{ and } \eta_l(\delta'(z,\sigma)) \leq \eta_l(z_0') \\ \text{undefined} & \text{otherwise} \end{cases}$$
   Let $K^* := L_m(S')$.

6. Output: K^* □

We first briefly explain what SMT does. It first computes the supremal controllable sublanguage $K \in \mathcal{C}(\mathcal{G}, \mathcal{E})$ in Step (2). If $K = \varnothing$, then clearly $\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) = \varnothing$. Suppose $K \neq \varnothing$. Then the computation in Step (3) is used to determine whether $\min_{\tilde{K} \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})} \omega(\mathcal{G}, f, h, \tilde{K})$ is finite, whose validity is shown in Lemma 3.3 below, which also indicates that the makespan of the supremal minimum-time controllable sublanguage is no more than the weight of the initial state $\mathcal{Q}_r(z_0)$. Thus, if $\mathcal{Q}_r(z_0) < +\infty$ then the sublanguage K' in Step (4) must contain the supremal minimum-time controllable sublanguage. Since $\hat{K}$ is the supremal controllable sublanguage of K', it must contain the supremal minimum-time controllable sublanguage. Step (5) is used to find such a supremal language. The idea is that, each marker state of S is associated with the execution time of the string that leads to the marker state. The string is unique because S is a tree automaton. Then we apply an algorithm similar to the one in [4], except that we do not count edge weights anymore. By doing this we can guarantee that, the resulting sub-automaton S' recognizes a controllable sublanguage of $\hat{K}$ (the proof of controllability is similar to the one in [4]) such that its makespan is minimum among all controllable sublanguages. This result will be shown shortly in Theorem 3.4. We can easily check that SMT terminates for each pair of $\mathcal{G}$ and $\mathcal{E}$, because, from [4] we know that the iteration at Step (3) always terminates, and clearly $\hat{K}$ is finite. The complexity of Step (3) has been shown in [4] to be polynomial. Similarly, in Step (5) the complexity of iteration is polynomial with respect to the number of states and transitions of S. But constructing S requires enumerating all possible strings, whose execution times are smaller than $\mathcal{Q}_r(z_0)$. It can be shown that searching the supremal minimum-time controllable sublanguage is NP-hard. Owing to the limited space, we will not provide the proof in this paper. It will be presented in another paper, which addresses the computational issues. To show that K^* is the supremal minimum-time controllable sublanguage, we need the following lemma.

Given a time-weighted plant $(\mathcal{G} = \{G_i \in \phi(\Sigma_i) | i \in I\}, f, h)$ and a requirement $\mathcal{E}$, let $(\mathcal{G}, f, h')$ be another time-weighted plant, where for every $\sigma, \sigma \in \Sigma$, $h'(\sigma, \sigma') = 1$. In other words, every pair of events in $(\mathcal{G}, f, h')$ is mutually exclusive. We can easily check that, the corresponding graph $\mathrm{Gr}(\mathcal{G}, f, h')$ is complete, which means in the resulting heap model $\mathcal{H}'$, the resource set $\mathcal{R}'$ is a singleton. For each $s = \sigma_1 \cdots \sigma_n \in L_m(\mathcal{G}) || L_m(\mathcal{E})$, we can derive that, $y_{\mathcal{H}'}(s) = \sum_{i=1}^{n} f(\sigma_i)$, which means the time-weights associated with $(\mathcal{G}, f, h')$ become ordinary weights, as described in, e.g. [4]. We call $(\mathcal{G}, f, h')$ *induced* from $(\mathcal{G}, f, h)$.

Lemma 3.3. Given a time-weighted plant $(\mathcal{G}, f, h) \in \Phi(\{\Sigma_i | i \in I\})$, let $(\mathcal{G}, f, h')$ be induced from $(\mathcal{G}, f, h)$. Let $\mathcal{E} \in \tilde{\Phi}(\{\Delta_j | j \in J\})$ be a requirement. Then (1) $\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) \neq \varnothing$ iff $\mathcal{NS}(\mathcal{G}, f, h', \mathcal{E}) \neq \varnothing$; (2) If $\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) \neq \varnothing$, then

$$\min_{K \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})} \omega(\mathcal{G}, f, h, K) \leq \min_{K \in \mathcal{NS}(\mathcal{G}, f, h', \mathcal{E})} \omega(\mathcal{G}, f, h', K)$$

□

Proof: $\omega(\mathcal{G}, f, h, K) < +\infty$ implies that K is finite, which means $\omega(\mathcal{G}, f, h', K) < +\infty$. Similarly, $\omega(\mathcal{G}, f, h', K) < +\infty$ implies $\omega(\mathcal{G}, f, h, K) < +\infty$. Thus, we have $K \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$ iff $K \in \mathcal{NS}(\mathcal{G}, f, h', \mathcal{E})$, which means $\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) \neq \varnothing$ iff $\mathcal{NS}(\mathcal{G}, f, h', \mathcal{E}) \neq \varnothing$.
(2) Suppose $\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) \neq \varnothing$. By (1) we have $\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) = \mathcal{NS}(\mathcal{G}, f, h', \mathcal{E})$. From

the definition of heap model, we can derive that,

$$(\forall K \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}))\, \omega(\mathcal{G}, f, h, K) \leq \omega(\mathcal{G}, f, h', K)$$

Apply minimization on both sides and we get

$$\min_{K \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})} \omega(\mathcal{G}, f, h, K) \leq \min_{K \in \mathcal{NS}(\mathcal{G}, f, h', \mathcal{E})} \omega(\mathcal{G}, f, h', K)$$

Thus, the lemma is true. ■

Lemma 3.3 says that, we can decide the emptiness of $\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$ by checking the emptiness of $\mathcal{NS}(\mathcal{G}, f, h', \mathcal{E})$. Since the latter is equivalent to decide whether there exists a controllable sublanguage, whose weight in terms of the maximum sum weight of a string in that sublanguage is finite, it can be effectively checked by using the algorithm presented in [4] (see Step (3) in SMT). It is interesting to point it out that, the algorithm presented in [4], which is a type of dynamic programming, can not be directly used to decide whether $\mathcal{NS}(\mathcal{G}, f, \mathcal{E})$ is empty because a time optimal path from a state, say x, to a marker state, need not be time optimal with respect to all pathes that traverse x, in other words, optimality of a global solution cannot be found by extending an existent local optimal solution, which is the key requirement of dynamic programming. Lemma 3.3 also says that, the smallest makespan of controllable sublanguages of $(\mathcal{G}, f, h)$ under $\mathcal{E}$ is no more than the smallest makespan of controllable sublanguages of $(\mathcal{G}, f, h')$ under $\mathcal{E}$. Since the latter can be effectively computed, Lemma 3.3 allows us to construct a finite language, that guarantees to contain the supremal minimum-time controllable sublanguage of $(\mathcal{G}, f, h)$ under $\mathcal{E}$. Since the language is finite, we can represent it by a finite-state tree automaton and compute the supremal minimum-time controllable sublanguage by a polynomial algorithm. This is described in Step (4) and Step (5). Next, we present our main result.

Theorem 3.4. Given a time-weighted plant $(\mathcal{G}, f, h) \in \Phi(\{\Sigma_i | i \in I\})$ and a requirement $\mathcal{E} \in \tilde{\Phi}(\{\Delta_j | j \in J\})$, suppose K^* is computed in SMT. Then we have the following results: (1) If $K^* \neq \varnothing$ then $K^* = \sup\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$; (2) if $K^* = \varnothing$ then $\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) = \varnothing$. □

Proof: (1) Suppose $K^* \neq \varnothing$. We first show that $K^* \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$, namely K^* is controllable with respect to $L(\mathcal{G})$, and $\omega(\mathcal{G}, f, h, K^*) < +\infty$. The latter is clearly true. In Step (5) we can check that, at each state z all uncontrollable events are allowed. Thus, K^* is controllable with respect to $\hat{K} = L_m(S)$. Since $\hat{K}$ is controllable with respect to $L(G)$, which is controllable with respect to $L(\mathcal{G})$, we get that K^* is controllable with respect to $L(\mathcal{G})$. Therefore, $K^* \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$, meaning that $\sup\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$ exists. Let $\tilde{K} := \sup\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})$. By Lemma 3.3 we have

$$\omega(\mathcal{G}, f, h, \tilde{K}) = \min_{K'' \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})} \omega(\mathcal{G}, f, h, K'') \leq \min_{K'' \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})} \omega(\mathcal{G}, f, h', K'')$$

By [4] we have

$$\min_{K'' \in \mathcal{NS}(\mathcal{G}, f, h, \mathcal{E})} \omega(\mathcal{G}, f, h', K'') = \mathcal{Q}_r(z_0)$$

Thus, $\tilde{K} \subseteq K'$. Since $\hat{K} := \sup\mathcal{C}(G, \kappa(K'))$, we get $\tilde{K} \subseteq \hat{K} = L_m(S)$. By using a similar argument as in [4] we can derive that,

$$\omega(\mathcal{G}, f, h, K^*) = \min_{K' \in \mathcal{C}(\mathcal{G}, \{S\})} \omega(\mathcal{G}, f, h, K') = \eta_l(z_0)$$

Thus, $\omega(\mathcal{G}, f, h, \tilde{K}) \geq \omega(\mathcal{G}, f, h, K^*)$. Since $\tilde{K}$ is the supremal minimum-time controllable sublanguage, we have $\omega(\mathcal{G}, f, h, \tilde{K}) = \omega(\mathcal{G}, f, h, K^*)$. Since S is a tree automaton, we can derive that K^* contains every controllable sublanguage $K' \in \mathcal{C}(\mathcal{G}, \{S\})$ with $\omega(\mathcal{G}, f, h, K') \leq \omega(\mathcal{G}, f, h, K^*)$. In particular, $\tilde{K} \subseteq K^*$. But since $\tilde{K}$ is supremal, we have

$\tilde{K} = K^*$.
(2) If $\mathcal{Q}_r(z_0) < +\infty$, by [4] we know that, there exists a controllable sublanguage of $(\mathcal{G}, f, h')$ under $\mathcal{E}$, whose weight is finite. Thus, by Lemma 3.3, there must exist a controllable sublanguage of $\mathcal{G}$ under $\mathcal{E}$, whose makespan is finite and no more than $\mathcal{Q}_r(z_0)$. Thus, $\hat{K} \neq \varnothing$, which means $K^* \neq \varnothing$. Therefore, when $K^* = \varnothing$, we only need to consider two cases. Case 1: $K = \sup\mathcal{C}(\mathcal{G}, \mathcal{E}) = \varnothing$. by the convention rule, we have $\omega(\mathcal{G}, f, h, \varnothing) = +\infty$. Thus, $\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) = \varnothing$. Case 2: $\mathcal{Q}_r(z_0) = +\infty$. This means $\mathcal{NS}(\mathcal{G}, f, h', \mathcal{E}) = \varnothing$, where $(\mathcal{G}, f, h')$ is induced from $(\mathcal{G}, f, h)$. Then, by Lemma 3.3, we have $\mathcal{NS}(\mathcal{G}, f, h, \mathcal{E}) = \varnothing$. ■

As an illustration, we apply the aforementioned technique to a simplified cluster tool example depicted in Figure 3, which consists of one load/exit lock (LEL) for feeding unprocessed wafers into the system and pulling processed wafers out of the system, two

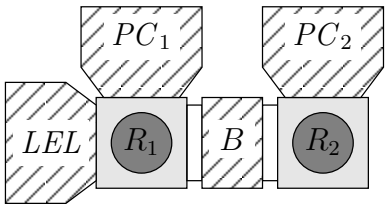

Figure 3: Example 4: The simplified cluster tool

processing chambers (PC1 and PC2) for processing wafers, two robots (R1 and R2) for transporting wafers inside the system, and one buffer (B) for swapping wafers between two robots. We assume that B has one slot. Figure 4 depicts the time-weighted plant model, where the time weight of each event is 1, except for Process1 and Process2, whose

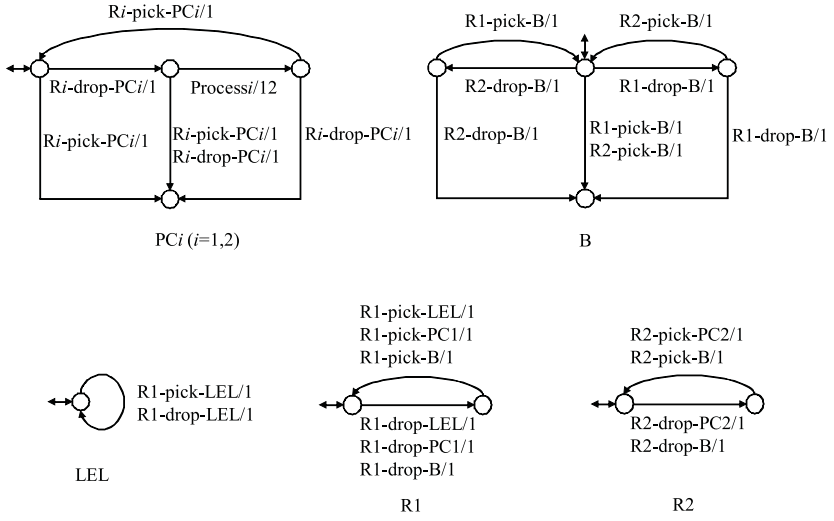

Figure 4: Example 4: Time-weighted component models of LEL, R1, R2, PC1, PC2, B

weights are 12 (because processing usually takes more time). All events are controllable, except for Process1 and Process2. The requirements are depicted in Figure 5, where

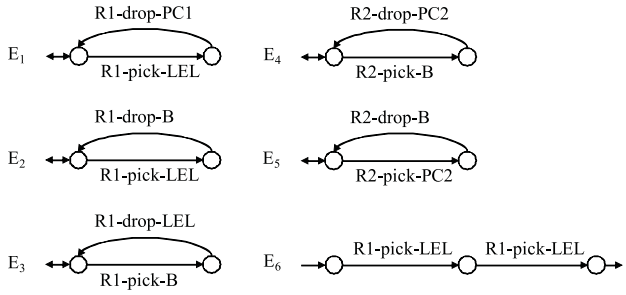

Figure 5: Example 4: Requirement models

requirements E_1-E_5 specify that, each wafer needs to go through the following routine sequence: LEL $\rightarrow$ PC1 $\rightarrow$ B $\rightarrow$ PC2 $\rightarrow$ B $\rightarrow$ LEL. The requirement E_6 specifies that there are only two wafers per each batch, which is for the purpose of illustration. We can certainly add more wafers in each batch, as shown in Table 1. We assume that the mutual exclusion function h is derivable from $\mathcal{G}$, which is asynchronous. The rationality of such an assumption can be explained as follows. Clearly, both robot models are asynchronous because each robot can only perform one action at each time instant. The LEL model contains only R1's events. Thus, it is also asynchronous. Although PCi ($i = 1, 2$) and B contain events from both R1 and R2, their legal behaviors (i.e. the nonblocking part) are strictly asynchronous. For example, in B only after R1 drops, R2 can pick; and after R2 drops, R1 can pick – in other words, pick and drop actions cannot be executed at the same time instant. Thus, all component models are asynchronous. From the requirement models we can see that, each requirement contains events from only one component automaton, thus, it does not introduce any new mutual exclusion pairs.

We apply SMT on the system to compute the supremal minimum-time controllable sublanguage of $(\mathcal{G}, f, h)$. By standard supervisor synthesis we can check that, $K = \sup\mathcal{C}(\mathcal{G}, \mathcal{E}) \neq \varnothing$. Step (3) terminates with $\mathcal{Q}_r(z_0) = 68$. We then create a tree automaton S according to SMT. Finally, the result of Step (5) is obtained, whose makespan is 54. This number matches our expectation based on manual calculation. A recognizer of K^* is depicted in Figure 6. Since h is control compatible and $\mathcal{G}$ is asynchronous, by Theorem 2.4 we know that, a timed supervisory control map g exists that can achieve K^* and respects the mutual exclusion imposed by h. From Figure 6 we can see that, the key to the minimum-time supervision is to process the second wafer in PC1 along with the first wafer being handled by R2, namely R1 and R2 handle two wafers in parallel. To test the effectiveness of SMT, we increase the batch size to different values and the results are summarized in Table 1. As we have expected, the unfolding part of SMT (i.e. the construction of the tree automaton S) is computationally intense. A possible solution to avoid constructing S is to use a greedy algorithm to compute a suboptimal timed supervisory control map. Owing to limited space we leave such a greedy algorithm to another paper that addresses the computational aspect of time optimal supervisory control. From the data in Table 1 we can derive the following formula $T : \mathbb{N} \rightarrow \mathbb{R}^+$,

Table 1

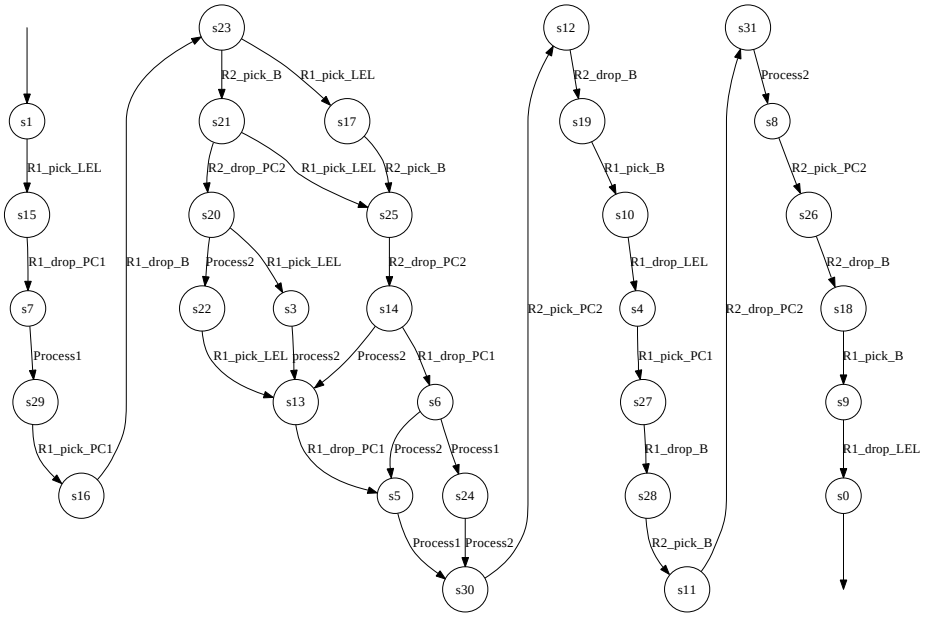

Figure 6: Example 4: Automaton $\kappa(K^*)$

Batch Size (BS)	Minimum Makespan (MM)	Throughput (BS/MM)
2	54	0.0370
3	74	0.0405
4	94	0.0426
5	114	0.0439
6	134	0.0448
7	154	0.0455
8	174	0.0460

which maps the batch size $n \in \mathbb{N}$ to the corresponding minimum makespan $T(n) \in \mathbb{R}^+$,

$$T(n) = 54 + 20(n-2), \text{ where } n \geq 2$$

Owing to limited space, a formal analysis of the correctness of this formula for an arbitrary value $n \geq 2$ is skipped here. An informal explanation is that, the system can hold at most 2 wafers at each time instant. Thus, when the batch size is equal to or more than 2, the system has reached its steady state. Thus, the inter arrival time of wafers becomes a constant, which is 20. As a byproduct, from this formula we can derive the steady-state throughput, which is

$$\lim_{n \to +\infty} \frac{n}{T(n)} = \lim_{n \to +\infty} \frac{n}{54 + 20(n-2)} = 0.05$$

4 Conclusions

In this paper we first present a minimum-time supervisory control problem, where the plant is described by a time-weighted system and the requirement is un-weighted. After that, we provide a terminable algorithm SMT to compute the supremal minimum-time controllable sublanguage, whose overall computational complexity is determined by the

complexity of creating a tree automaton because the rest of steps are polynomial. We have also shown that, the computed supremal minimum-time controllable sublanguage is guaranteed to be implementable by a timed supervisory control map if the mutual exclusion function is control compatible and the system model is asynchronous. In an ideal situation, where the computation of the control law takes no time and each eligible event fires immediately without any delay, the timed supervisory control map can achieve the minimum-time supervision. It is an open question whether a similar treatment can be applied to the case, where partial observation may be present. The supervisory control problem in this paper is formulated in a centralized manner, namely we have one product plant and one product requirement. In reality, we may encounter high computational complexity during synthesis. Thus, it is of our primary interest to investigate whether there is a similar minimum-time supervisory control framework applicable to a hierarchical and distributed setting, which will be addressed in our future papers.

Acknowledgement:

We would like to thank Dr. Albert T. Hofkamp of the Systems Engineering Group at Eindhoven University of Technology for coding all algorithms presented in this paper.

Bibliography

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183-235, 1994.

[2] E. Asarin and O. Maler. As soon as possible: time optimal control for timed automata. In *Proc. 2nd International Workshop on Hybrid Systems: Computation and Control (HSCC99), volume 1569 of LNCS*, pages 19-30, 1999.

[3] E. Asarin, O. Maler, A. Pnueli and J. Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469-474, 1998.

[4] Y. Brave and M. Heymann. On optimal attraction of discrete-event processes. *International Journal of Information Sciences*, 67(3):245-276, 1993.

[5] B.A. Brandin and W.M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Trans. Autom. Control*, 39(2):329-342, 1994.

[6] U. Buy, H. Darabi, M. Lehene and V. Venepally. Supervisory control of time Petri nets using Nnet unfolding. In *Proc. 29th Annual International Computer Software and Applications Conference (COMPSAC05)*, pages 97-100, 2005

[7] U. Buy, M. Lehene and H. Darabi. Latency-based supervisors for enforcing deadlines in time Petri nets. In *proc. 29th Annual IEEE/NASA Software Engineering Workshop*, pages 211-218, 2005.

[8] S. Gaubert. Performance evaluation of (max,+) automata. *IEEE Trans. Autom. Control*, 40(12):2014-2025, 1995.

[9] S. Gaubert and J. Mairesse. Modeling and analysis of timed Petri nets using heaps of pieces. *IEEE Trans. Autom. Control*, 44(4):683-697, 1999.

[10] B. Heidergott, G. J. Olsder and J. van der Woude. Max Plus at Work. *Princeton University Press*, 2006.

[11] C. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666-677, 1978.

[12] X.D. Koutsoukos and P.J. Antsaklis. Hybrid control systems using timed Petri nets: supervisory control design based on invariant properties. *Hybrid Systems V, Lecture Notes in Computer Science*, pages 142-162, 1999.

[13] X.D. Koutsoukos, K.X. He, M.D. Lemmon and P.J. Antsaklis. Timed Petri nets in hybrid systems: stability and supervisory control. *Discrete Event Dynamic Systems: Theory and Applications*, 8(2):137-173, 1998.

[14] R. Kumar and V. Garg. Optimal supervisory control of discrete event dynamical systems. *SIAM J. Control and Optimization*, 33(2):419-439, 1995.

[15] S. La Torre, S. Mukhopadhyay and A. Murano. Optimal-reachability and control for acyclic weighted timed automata. In *Proc. 2nd IFIP Conference on Theoretical Computer Science (TCS02)*, pages 485-497, 2002.

[16] O. Maler, A. Pnueli and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Proc. 12th Symposium on Theoretical Aspects of Computer Science (STACS95), volume 900 of LNCS*, pages 229-242, 1995.

[17] H. Marchand, O. Boivineau and S. Lafortune. Optimal control of discrete event systems under partial observation. In *Proc. 40th IEEE Conference on Decision and Control (CDC01)*, pages 2335-2340, 2001.

[18] J.B. Orlin. Contentment in graph theory: covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80:406-424, 1977.

[19] J.S. Ostroff. Temporal Logic for Real-Time Systems. *Advanced Software Development Series, ed. J. Kramer. Research Studies Press Limited (distributed by John Wiley and Sons)*, England, 1989.

[20] K. Passino and P. Antsaklis. On the optimal control of discrete event systems. In *Proc. 28th IEEE Decision and Control Conference (CDC89)*, pages 2713-2718, 1989.

[21] J. Pu, CM. Lagoa, and A. Ray. Robust optimal control of regular languages with event cost uncertainties. In *Proc. 42th IEEE Conference on Decision and Control (CDC03)*, pages 3209-3214, 2003.

[22] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event systems. *SIAM J. Control and Optimization*, 25(1):206–230, 1987.

[23] C.V. Ramamoorthy and G.S. Ho. Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Transactions on Software Engineering*, 6(5):440-449, 1980.

[24] R. Sengupta and S. Lafortune. An optimal control theory for discrete event systems. *SIAM J. Control and Optimization*, 36(2):488-541, 1998.

[25] A. Takae, S. Takai, T. Ushio, S. Kumagai and S. Kodama. Maximally permissive controllers for controlled time Petri nets. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 79(5): 1-8, 1996.

[26] S. Tripakis and K. Altisen. On-the-fly controller synthesis for discrete and dense-time systems. In *Proc. of World Congress on Formal Methods (FM99), volume 1708 of LNCS*, pages 233-252, 1999.

[27] G.X. Viennot. Heaps of pieces, I: Basic definitions and combinatorial lemmas. *Combinatoire Énumérative*, Labelle and Leroux, Eds., no. 1234 in Lecture Notes in mathematics, pages 321–350, 1997.

[28] J. Wang. Timed Petri Nets: Theory and Application. *Kluwer Academic Publishers*, 1998.

[29] W. M. Wonham. *Supervisory Control of Discrete-Event Systems*. Systems Control Group, Dept. of ECE, University of Toronto. URL: www.control.utoronto.ca/DES, 2007.

[30] W.M. Wonham and P.J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control and Optimization*, 25(3):637–659, 1987.

[31] J. Yi, S. Ding, M.T. Zhang, M.T. and P. van der Meulen. Throughput analysis of linear cluster tools. In *proc. 3rd IEEE International Conference on Automation Science and Engineering (CASE07)*, pages 1063-1068, 2007.