实时任务准入控制器的自动合成

P. S. Thiagarajan[1] 杨绍发[2] 王 义[3]

[1](美国哈佛大学哈佛医学院 系统药理学实验室 波士顿 MA02115)
[2](中国科学院软件研究所 计算机科学国家重点实验室 北京 100190)
[3](瑞典乌普萨拉大学信息技术系 乌普萨拉 75105)

摘 要 在许多实时系统中，同一个计算平台上往往既有硬实时关键计算任务又有软实时非关键计算任务。硬实时任务必须在规定时间内完成，否则将导致系统错乱或崩溃等严重后果。而软实时任务若没有在规定时间内完成，虽会影响系统性能，但不会造成重大后果。为确保每个硬实时任务均在其规定时间内完成，在某些情况下需要拒绝一些软实时任务进入任务队列。文章提出了一种基于控制器自动合成策略的解决方案，通过所设计的准入控制器，对系统产生的每一个新任务自动决定是否准其进入任务队列。准入控制器必须使得所有被准入的任务均在规定时间内完成，并且决策序列满足以线性时态逻辑描述的服务质量要求。文章的主要贡献是提出了判定是否存在准入控制器的算法，该算法能在判定结果为真时构造出一个以有限状态时间自动机表达的准入控制器。

关键词 实时系统；任务调度；控制器合成；自动机理论
中图分类号 TP 3-0 TP 31 **文献标志码** A

Synthesis of Timed Admission Controllers

P. S. Thiagarajan[1] YANG Shaofa[2] WANG Yi[3]

[1](*Laboratory of Systems Pharmacology, Harvard Medical School, Harvard University, Boston* MA 02115, *USA*)
[2](*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing* 100190, *China*)
[3](*Department of Information Technology, Uppsala University, Uppsala* 75105, *Sweden*)

Abstract In many real-time computing environments, there are some tasks that are time-critical while others are not. To ensure that every critical task can be completed before its deadline, it is necessary to reject some non-critical tasks to entry into the ready queue. We address this problem in the framework of controller synthesis. Our goal is to come up with an admission controller which admits or rejects a task request. With such a controller, no admitted tasks will miss their deadline and the admitted patterns of task releases satisfy a quality-of-service constraint in the form of a linear time temporal logic specification. We prove that it is decidable to determine if such an admission controller exists. Furthermore, if the answer is positive, it is possible to effectively construct a controller in the form of a finite timed controller.

Keywords real-time systems; scheduling; controller synthesis; automata theory

收稿日期：2016-12-14 **修回日期**：2017-04-13
基金项目：中国科学院国际人才计划项目(2016VTA024、2017VTB0003)； 国家重点基础研究发展计划(973 计划)项目(2014CB340700)
作者简介：P. S. Thiagarajan, 访问教授，研究方向为系统生物学、模型检测等；杨绍发(通讯作者)，副教授，研究方向为模型检测、计算机器人学等，E-mail：yangsf@ios.ac.cn；王义，教授，研究方向为实时与嵌入式系统、模型检测等。

1 Introduction

In real-time systems, it is important to schedule tasks so that all time-critical tasks are completed before their deadlines. Classical schedulability analysis techniques[1] make strong assumptions about the temporal arrival patterns of the tasks. To overcome this limitation, a model was suggested in Fersman et al's report[2] (hereafter Fersman's report for brevity), where timed automata are used to describe task arrival patterns. It was shown that, in a uniprocessor setting, one can decide whether all tasks can be scheduled to meet their deadlines when the task arrivals are described by a timed automaton. As surveyed in Stigge's report[3], this model is the most expressive among existing graph-based real-time task models. Any existing task model can be represented in the timed automata based task model of Fersman's report[2].

In many real-time settings, the task arrival patterns will be such that some tasks will miss their deadlines. One method, as noted in Liu's report[4], to deal with this is to subject each new task instance to an acceptance test and admit it if the new ready queue resulting from adding this task instance is still schedulable. Such an approach is not satisfactory because a critical task may fail to get into the ready queue. Hence it is useful to consider designing more flexible admission policies.

In this paper, we study this problem from the perspective of controller synthesis. Using a timed automaton, we model the real-time setting as an open system. This open system, called a plant in this context, will have environment states from which the environment can make uncontrollable timed moves to release tasks. Each such uncontrollable action, leading to a system state, will then be immediately followed by an urgent pair of controllable actions: one of them, admitting the just released task instance and putting it into the ready queue and the other one rejecting it. We study the problem of constructing a strategy for choosing the controllable actions so that the admitted task instances, no matter what the environment does, can all be scheduled without missing their deadlines. A trivially safe policy would be to reject all task instances. To prevent this, we also require that the admitted sequences of plant transitions should satisfy an LTL (Linear Time Temporal Logic) specification that will typically demand some liveness and fairness properties.

We note that our framework is expressive enough to distinguish between hard real-time tasks—by specifying they must always be admitted—and soft tasks. Further, we can impose QoS (Quality of Service) requirements on the soft tasks by specifying fairness requirements such as:"Along any run, if (the soft task) τ is released infinitely often, then τ must also be admitted infinitely often". For ease of presentation, however, we do not impose a syntactic distinction between hard and soft tasks.

As in Fersman's report[2], we assume a uni-processor computing resource for the sake of convenience. The EDF (Earliest Deadline First) scheduling policy with preemption is known to be optimal in this setting[1]. In other words, if a task set is schedulable at all, then it is schedulable under EDF with preemption. Hence we assume this scheduling policy. We then find that, in this setting, given a plant in the form of a timed automaton and an LTL specification, one can effectively determine whether there is a winning strategy for admitting tasks so that all admitted tasks are schedulable and all the

runs permitted by the strategy meet the given LTL specification. We also find that whenever there is a winning strategy, there is in fact a winning strategy which can be synthesized as a finite timed automaton called the admission controller. Consequently, the controlled behavior obtained via the parallel composition of the plant and the controller will admit only schedulable ready queues and satisfy the LTL specification.

Our work may be viewed as an extension to an open system framework of the results reported in Fersman's report[2]. As discussed already, the current setting has a natural motivation and it is a pleasing fact that techniques from the controller synthesis domain and timed-automata-based schedulability analysis techniques can be combined in a natural manner to solve the synthesis problem at hand. In the literature, a number of studies[5-9] are available regarding controller synthesis in a timed setting. The key motivation of these studies is to extend classical controller synthesis results for discrete event systems[10-12] to a timed setting. In comparison, though we use the language and techniques of (timed) controller synthesis, our motivation is very different. In the present setting, our admission controller will not have any clock variables of its own. It will be interesting to endow the controller with its own clocks and granularity and study our controller synthesis problem along the lines of timed control[7,8].

A second related line of work is to derive a schedule for a real-time application, given the timed model of the application and a set of resource constraints[13-19] in fact carrying out the work using the controller synthesis paradigm. However, the emphasis in this line of work is to restrict the timed behaviors of the application so as to meet, in a timely fashion, access to shared resources. It will be interesting to extend our work along this line, to multi-processor settings accompanied by resource access protocols for shared resources.

A recent trend in real-time systems is the scheduling of real-time tasks in mixed-criticality frameworks. The key characteristics is, instead of one fixed computation time, a task can be associated with several worst-case execution times, each corresponding to an estimate at a different critical level. A comprehensive review of work in this area can be found in Burns' report[20]. It will be worthwhile to extend our synthesis results to the scheduling of mixed-criticality tasks.

In section 2, we formulate our plant model and define its operational semantics. In section 3, we specify the admission controller synthesis problem and state the our main result. The proof of the main result is presented in section 4 and the prospects for extending the current work is discussed in the concluding section 5.

2 The plant model

We first recall that a timed automaton $\mathcal{A}$, is a structure $(Q, q_0, X, \Sigma, \rightarrow)$ where Q is a finite set of locations and $q_0 \in Q$ is the initial location; X is a finite set of clocks and Σ is a finite set of events. $\rightarrow \subseteq Q \times \Phi(X) \times \Sigma \times 2^X \times Q$ is the transition relation. Here $\Phi(X)$ is the set of clock constraints over X. A clock constraint over X is a finite conjunction of basic constraints of the form $x \prec c$, $x - y \prec c$ where $x, y \in X, c \in \mathbb{N}, \prec \in \{<, \leqslant, >, \geqslant\}$. As usual $\mathbb{N}$ is the set of natural numbers. In what follows, $\mathbb{R}_0$ and $\mathbb{R}_+$ denote the set of non-negative reals and positive reals, respectively. The difference

constraints $x-y \prec c$ will be needed later to capture the behavior of the ready queue.

A clock valuation V over X is a function $X \mapsto \mathbb{R}_+$. For $t \in \mathbb{R}_+$, $V+t$ is the clock valuation $(V+t)(x)=V(x)+t$ for $x \in X$. For $Y \subseteq X$, $V[Y:=0]$ is the clock valuation which maps every clock in Y to zero and agrees with V on other clocks. The notation $V \vDash \varphi$ means V satisfies the clock constraint φ and is defined in the obvious way. The timed behavior of $\mathcal{A}$ is given by the transition system $TS_{\mathcal{A}}=\left[RC_{\mathcal{A}},(q_0,V_0),\mathbb{R}_+ \times \Sigma,\Rightarrow_{\mathcal{A}}\right]$ where $RC_{\mathcal{A}}$ and $\Rightarrow_{\mathcal{A}}$ are the least sets satisfying the following: (1) $(q_0,V_0) \in RC_{\mathcal{A}}$, where $V_0(x)=0$ for every clock variable x; and (2) if $(q,V) \in RC_{\mathcal{A}}$ and there exists $(q \overset{\varphi,a,Y}{\rightarrow} q')$ and $t \in \mathbb{R}_+$ such that $V+t \vDash \varphi$, then $(q,V) \overset{t,a}{\Rightarrow}_{\mathcal{A}} (q',V')$ and $(q',V') \in RC_{\mathcal{A}}$, where $V'=(V+t)[Y:=0]$.

It is well-known[21] that we can take a quotient of $TS_{\mathcal{A}}$ in the form of a finite transition system, called the region automaton $RG_{\mathcal{A}}$ of $\mathcal{A}$. For $x \in X$, let c_x be the maximum constant which appears in basic constraints (of transition guards of) $\mathcal{A}$ of the form $x \prec c$, where $c \in \mathbb{N}, \prec \in \{<,\leqslant,>,\geqslant\}$. Two clock valuations V, V' are region-equivalent, denoted $V \sim V'$, iff the following hold:

(1) For each $x \in X$, either $\lfloor V(x) \rfloor = \lfloor V'(x) \rfloor \leqslant c_x$ or $V(x)>c_x, V'(x)>c_x$. Further, in the former case, $fra[V(x)]=0$ iff $fra[V'(x)]=0$, where $fra(v)$ is the fractional part of v;

(2) For each $x, y \in X$ such that $V(x) \leqslant c_x, V'(x) \leqslant c_x$, and $V(y) \leqslant c_y, V'(y) \leqslant c_y$, we have $fra[V(x)] \leqslant fra[V(y)]$ iff $fra[V'(x)] \leqslant fra[V'(y)]$;

(3) For every basic constraint $\theta: x-y \prec c$ which appears in (transition guards of) $\mathcal{A}$, $V \vDash \theta$ iff $V' \vDash \theta$.

A clock region is an equivalence class of clock valuations. A region is a pair (q,r), where $q \in Q$ and r is a clock region. We have $RG_{\mathcal{A}}=\left[RR_{\mathcal{A}},(q_0,[V_0]),\Sigma,\rightsquigarrow_{\mathcal{A}}\right]$, where $RR_{\mathcal{A}}$ and $\rightsquigarrow_{\mathcal{A}}$ are the least sets satisfying the following:

(1) $(q_0,[V_0]) \in RR_{\mathcal{A}}$;

(2) If $(q,[V]) \in RR_{\mathcal{A}}$ and there exists $(q,V_1) \overset{t,a}{\rightarrow}_{\mathcal{A}} (q',[V_1'])$ where $V_1 \in [V]$, then $(q',[V_1']) \in RR_{\mathcal{A}}$ and $(q,[V]) \overset{a}{\rightsquigarrow}_{\mathcal{A}} (q',[V_1'])$.

2.1 The model

Next we recall how task arrival patterns in a real time environment can be modeled using timed automata as proposed in Fersman's report[2]. The basic idea is to associate a task with each location. Whenever a location is entered, an instance of the task associated with the location is supposed to be released. Here, it will be convenient to associate tasks with the transitions rather than with the locations. We also wish to highlight that we are dealing with an open system model—called a plant in this context—of task arrival patterns on which an admission control policy can be imposed. Formally, we define a plant $\mathcal{P}$ to be a structure $(Q_e,Q_s,q_0,\Omega,C,D,X,\rightarrow_e,\rightarrow_s)$, where:

(1) Q_e and Q_s are disjoint finite nonempty sets of environment states and system states, respectively.

(2) $q_0 \in Q_e$ is the initial state.

(3) Ω is a finite set of task types. The functions C, D: $\Omega \mapsto \mathbb{N}$ associate with each task type a computation time and a relative deadline, respectively. Further, for each $\tau \in \Omega$, $0<C(\tau) \leqslant D(\tau)$.

(4) X is a finite set of clocks.

(5) $\rightarrow_e \subseteq Q_e \times \Phi(X) \times \Omega \times 2^X \times Q_s$ is a set of environment transitions. For each system state q', there exists a unique environment state q and a unique environment transition of the form $(q \overset{\varphi,\tau,Y}{\rightarrow}_e q')$.

(6) $\rightarrow_s \subseteq Q_s \times \{0,1\} \times Q_e$ is a set of system transitions. For each system state q', there exists a

unique environment state q and exactly two system transitions of the form $(q' \xrightarrow{0}_s q)$ and $(q' \xrightarrow{1}_s q)$.

From now on, we fix a plant $\mathcal{P}$ defined above. Informally, the plant model consists of a timed automaton whose events are interpreted as tasks, in case they are associated with environment transitions. For system transitions, we allow only the events {0, 1} which will be used to capture the decisions made by the controller. The semantics of the plant will implicitly impose a zero-delay on the system states. In other words, as soon as a system state is entered, the controller will make the decision to either accept the task that has just been released by the environment, this is captured by the 1-labeled transition going out of the system state. On the other hand, the 0-labeled transition going out of a system state models the decision to reject the just released task. We could have assigned a special clock variable to capture the immediacy of these transitions, but we have not done so for convenience. As mentioned above, the decision as to whether a just released task is to be admitted or not is made as soon as the task is released by the environment. Thus system moves come in pairs and each such pair is uniquely associated with an environment move. Further, the environment is oblivious to the admission policy being followed by the system. This explains the restrictions placed on the structure of the transitions. In what follows, we shall often denote both $\rightarrow_e$ and $\rightarrow_s$ as $\rightarrow$. An example of a plant is shown in Fig. 1.

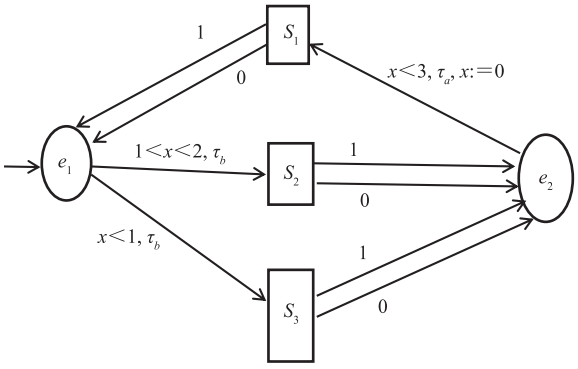

Fig. 1　A simple plant

2.2　The ready queue states

The semantics of $\mathcal{P}$ is to be understood with respect to a scheduling policy. Here we work with the simple framework consisting of a single processor. The scheduling policy we shall assume is EDF (Earliest Deadline First) with preemption. At any given time, the task that is executing on the processor is the one with earliest relative deadline among all the tasks currently in the ready queue. Whenever a fresh task τ' arrives, if its relative deadline is less than the (current) relative deadline of the currently executing task τ, then τ is preempted, placed back in the ready queue (at the head of the queue actually) and τ' will start executing. Thus the state of the plant will consist of the current location, the values of the clocks associated with plant and the state of the ready queue. This motivates the following definition.

A ready queue η over Ω is a finite sequence $\sigma_1 \cdots \sigma_n$ where, for $i=1,\cdots,n$, $\sigma_i = \langle \tau_i, c_i, d_i \rangle$, with $\tau_i \in \Omega, c_i, d_i \in \mathbb{R}_0, c_i \leqslant C(\tau_i), d_i \leqslant D(\tau_i)$. Intuitively, σ_i is a task instance at position i of type τ_i with remaining computation time c_i and remaining relative deadline d_i. We assume the convention that σ_1 is the head of the queue. The empty queue is denoted by ε.

The order in which the task instances appear in the queue reflects the order in which these task instances will be scheduled. If the queue is non-empty, the task instance at the head of queue is currently supposed to be executing. When time passes, both the remaining computation time (c_i) and the relative deadline (d_i) of the task at the head of queue will decrease while just the relative deadlines of the remaining tasks

will decrease. When the task at the head of queue finishes, it will leave the ready queue and the task behind it will be promoted to the head of queue and will start executing. A fresh task instance, when admitted will be inserted into the ready queue at the appropriate slot as dictated by the relative deadline of this instance and the current relative deadlines of the task instances in the queue.

The way in which the state of the ready queue changes due to passage of time (and the execution of tasks) is modeled by the function Comp. This function takes a queue η and a time duration t as inputs and returns a queue η' resulting from computing the tasks in η for t time units on the processor and is defined as follows. Firstly, $\mathrm{Comp}(\varepsilon,t)=\varepsilon$. Next suppose $\eta=\sigma_1\cdots\sigma_n$ where $\sigma_i=\langle\tau_i,c_i,d_i\rangle$ for $1\leqslant i\leqslant n$. If $t<c_1$, then $\mathrm{Comp}(\eta,t)=\eta'$, where $\eta'=\sigma_1'\sigma_2'\cdots\sigma_n'$ with $\sigma_1'=\langle\tau_1,c_1-t,d_1-t\rangle$ and $\sigma_i'=\langle\tau_i,c_i,d_i-t\rangle$ for $1<i\leqslant n$.

If $t\geqslant c_1$, then $\mathrm{Comp}(\eta,t)=\mathrm{Comp}(\eta',\ t-c_1)$ where $\eta'=\sigma_1'\sigma_2'\cdots\sigma_{n-1}'$ with $\sigma_i'=\langle\tau_{i+1},c_{i+1},d_{i+1}-c_1\rangle$ for $1\leqslant i\leqslant n-1$.

The way in which a task that has just been admitted inserted into the ready queue is captured by the function Sch, which takes as inputs a ready queue η and a task instance J, and returns a ready queue η'.

Let $\eta=\langle\tau_1,c_1,d_1\rangle\cdots\langle\tau_n,c_n,d_n\rangle$ be a ready queue, where $d_1\leqslant\cdots\leqslant d_n$, and let $J=\langle\tau,C(\tau),D(\tau)\rangle$ be a task instance. Then $\mathrm{Sch}(\eta,J)=\eta'$, where $\eta'=\langle\tau_1,c_1,d_1\rangle\cdots\langle\tau_{k-1},c_{k-1},d_{k-1}\rangle\langle\tau,C(\tau),D(\tau)\rangle\langle\tau_k,c_k,d_k\rangle\cdots\langle\tau_n,c_n,d_n\rangle$ with k being the least index such that $D(\tau)<d_k$. Clearly this reflects the EDF policy.

A queue $\eta=\langle\tau_1,c_1,d_1\rangle\cdots\langle\tau_n,c_n,d_n\rangle$ is schedulable iff $\sum_{i=1}^{m}c_i\leqslant d_m$ for $1\leqslant m\leqslant n$. That is, assuming that no fresh task instances are inserted into the ready queue, each task instance currently residing in the ready queue can be scheduled (according to EDF) and run to completion before its current deadline expires. In what follows, all non-schedulable queues will be identified with the single designated (non-schedulable) queue named Err. The functions Comp and Sch are extended in the obvious way to reflect this convention.

2.3 The plant semantics

The semantics of the plant $\mathcal{P}$ is defined by a transition system $TS_{\mathcal{P}}=\left(RC_{\mathcal{P}},\langle q_0,V_0,\varepsilon,\#\rangle,\mathrm{ACT}_{\mathcal{P}},\Rightarrow_{\mathcal{P}}\right)$, where $\mathrm{ACT}_{\mathcal{P}}=(\mathbb{R}_+\times\Omega)\cup\{0,1\}\cup\{\mathrm{err}\}$; $RC_{\mathcal{P}}$ is the set of (reachable) configurations; $\Rightarrow_{\mathcal{P}}$ are the least sets satisfying the following:

(1) $\langle q_0,V_0,\varepsilon,\#\rangle\in RC_{\mathcal{P}}$;

(2) Suppose $\langle q,V,\eta,\#\rangle\in RC_{\mathcal{P}}$ and $q\xrightarrow{\varphi,\tau,Y}_e q'$. If there exists $t\in\mathbb{R}_+$ such that $V+t\vDash\varphi$, then $\langle q',V',\eta',J\rangle\in RC_{\mathcal{P}}$ and $\langle q,V,\eta,\#\rangle\overset{t,\tau}{\Rightarrow}_{\mathcal{P}}\langle q',V',\eta',J\rangle$, where $V'=(V+t)[Y:=0]$, $\eta'=\mathrm{Comp}(\eta,t)$, $J=\langle\tau,C(\tau),D(\tau)\rangle$;

(3) Suppose $\langle q,V,\eta,J\rangle\in RC_{\mathcal{P}}$ and $q\xrightarrow{0}_s q'$. Then $\langle q',V,\eta,\#\rangle\in RC_{\mathcal{P}}$ and $\langle q,V,\eta,J\rangle\overset{0}{\Rightarrow}_{\mathcal{P}}\langle q',V,\eta,\#\rangle$;

(4) Suppose $\langle q,V,\eta,J\rangle\in RC_{\mathcal{P}}$ and $q\xrightarrow{1}_s q'$. If $\mathrm{Sch}(\eta,J)=\eta'\neq\mathrm{Err}$, then $\langle q',V,\eta',\#\rangle\in RC_{\mathcal{P}}$ and $\langle q,V,\eta,J\rangle\overset{1}{\Rightarrow}_{\mathcal{P}}\langle q',V,\eta',\#\rangle$. If $\mathrm{Sch}(\eta,J)=\mathrm{Err}$, then $\mathrm{Err}\in RC_{\mathcal{P}}$ and $\langle q,V,\eta,J\rangle\overset{1}{\Rightarrow}_{\mathcal{P}}\mathrm{Err}$;

(5) $\mathrm{Err}\overset{\mathrm{err}}{\Rightarrow}_{\mathcal{P}}\mathrm{Err}$.

Configurations of the form $\langle q,V,\eta,\#\rangle$ are called environment configurations where $\#$ is a special symbol, while those of the form $\langle q,V,\eta,J\rangle$ are called system configurations. The sets of environment configurations and system configurations are denoted $RC_{\mathcal{P}}^e$ and $RC_{\mathcal{P}}^s$, respectively. Note that Err is neither an environment configuration nor a system configuration. Intuitively, $TS_{\mathcal{P}}$ is the game graph of a real-time system where the environment triggers tasks in an uncontrollable manner, while the system

admits or rejects them in a controllable way. Note that we require environment moves to consume a nonzero amount of time and system moves to consume no time.

In what follows, we shall often omit the subscript $\mathcal{P}$. For a configuration σ, we define $\mathrm{Act}(\sigma)=\{a\in\mathrm{ACT}\mid \exists\sigma',\sigma\xrightarrow{a}\sigma'\}$. That is, $\mathrm{Act}(\sigma)$ is the set of possible actions that can be taken at configuration σ. Note that $\mathrm{Act}(\sigma)\subseteq\mathbb{R}_+\times\Omega$, if σ is an environment configuration, $\mathrm{Act}(\sigma)=\{0,1\}$; if σ is a system configuration, $\mathrm{Act}(\mathrm{Err})=\{\mathrm{err}\}$.

Without loss of generality, we shall assume that the plant has no dead configuration, that is, $\mathrm{Act}(\sigma)\neq\varnothing$ for $\sigma\in RC$. Given the restrictions on the structure of the plant transitions, we can uniquely recover a sequence of plant transitions from each sequence of transitions of *TS* with the help of the projection operator $\mathrm{Prj}_{\mathcal{P}}$. Let $\delta=\big\langle\langle q,V,\eta,\#\rangle,(t,\tau),\langle q',V',\eta',J\rangle\big\rangle$ be an environment transition in *TS*. Then clearly, there is a unique transition $\beta=(q\xrightarrow[e]{\varphi,\tau,Y}q')$ in $\mathcal{P}$ such that $V=t\vDash\varphi, V'=(V+t)[Y:=0]$. In this case, we define $\mathrm{Prj}_{\mathcal{P}}(\delta)=\beta$. For a system move $\delta=\big\langle\langle q,V,\eta,J\rangle,m,\langle q',V',\eta',\#\rangle\big\rangle$ in *TS*, we define $\mathrm{Prj}_{\mathcal{P}}(\delta)=(q\xrightarrow[s]{m}q')$. For the special move $\delta=\langle\mathrm{Err},\mathrm{err},\mathrm{Err}\rangle$, we define $\mathrm{Prj}_{\mathcal{P}}(\delta)=\delta$. Using the operator $\mathrm{Prj}_{\mathcal{P}}$, it is clear that we can uniquely associate a sequence of transitions of the plant with each transition sequence of *TS* (modulo the handling of the err moves).

3 The admission controller synthesis problem

The plant $\mathcal{P}$ is said to be schedulable iff $\mathrm{Err}\notin RC$. That is, no task will ever miss its deadline, even if we never reject any task. It is proved in Fersman's report[2]—in a related closed system setting—that one can effectively determine if $\mathcal{P}$ is schedulable, under the preemptive EDF policy for a single processor.

Here we consider plants which may not be schedulable and study the problem of designing an admission policy under which the restricted behavior of the plant becomes schedulable. Clearly, one can trivially achieve schedulability by constantly rejecting all arrived tasks. To rule out this, we shall assume that we are also given a liveness specification. Here we shall assume that LTL is the chosen specification language. In order to formulate this, we fix a finite set of atomic propositions *AP* and a labeling function $\lambda: TR_{\mathcal{P}}\mapsto 2^{AP}$, where $TR_{\mathcal{P}}=\rightarrow_e\cup\rightarrow_s$. The target state of both 0 and 1 transitions coming out of a system state is the same. To capture liveness properties, we need to get the admission decisions made by the system during a run. Hence we assign atomic propositions to transitions than to states. From now on, we shall assume our plant model to be augmented with *AP* and a labeling function λ as specified above.

The set of LTL formulas based on *AP* is as usual given by the syntax with *p* ranging over *AP*:

$$\mathrm{LTL}(AP)::=p\mid\sim\psi\mid\psi\vee\psi'\mid O(\psi)\mid\psi\,\mathcal{U}\,\psi'$$

Let TR_{TS} be the set of transitions of *TS*, that is, $\Rightarrow_{\mathcal{P}}$. Using the operator Prj, we extend λ to TR_{TS}. By abuse of notation, this extension will also be denoted as λ and it is given by: $\lambda(\delta)=\lambda\big[\mathrm{Prj}(\delta)\big]$ for every $\delta\in TR_{TS}$. By convention, we will set $\lambda\big(\langle\mathrm{Err},\mathrm{err},\mathrm{Err}\rangle\big)=\varnothing$. Let $\chi=\beta_0\beta_1\cdots$ be an infinite sequence of transitions of *TS*. Set $\chi(i)=\beta_i$ for each natural number *i*. The notion of the LTL formula ψ being satisfied at $\beta(i)$, denoted $\chi,i\vDash\psi$, is defined in the standard way. For instance, $\chi,i\vDash p$ iff $p\in\lambda\big[\chi(i)\big]$. We shall say that χ is a model of ψ,

and denote this as $\chi \vDash \psi$, iff $\chi, 0 \vDash \psi$.

We are now ready to define the notion of a strategy. Define the set of (finite) runs of *TS*, denoted *Runs*, to be the least subset of $TR_{TS}^{\star}$ satisfying: firstly, $\varepsilon \in Runs$ and the end state of ε is $\langle q_0, V_0, \varepsilon, \#\rangle$; secondly, let $\rho \in Runs$ and the end state of ρ be σ. If there exists $\delta = \langle \sigma, a, \sigma' \rangle \in TR_{TS}$, then $\rho\delta \in Runs$ and the end state of $\rho\delta$ is σ'. In what follows, We denote the end state of a run ρ by $\mathrm{end}(\rho)$.

A strategy f for the plant $\mathcal{P}$ is a function $f : Runs \mapsto 2^{\mathrm{ACT}}$, which satisfies the following conditions. These conditions are stated for a run ρ whose end state is σ: If $\sigma \in RC^s$, then $\varnothing \neq f(\rho) \subseteq \mathrm{Act}(\sigma) = \{0,1\}$; If $\sigma \in RC^e \cup \{\mathrm{Err}\}$, then $f(\rho) = \mathrm{Act}(\sigma)$.

Thus a strategy recommends a set of moves at the end state of each run. The moves recommended are a subset of the moves enabled at the end state and at least one move is recommended. In this sense a strategy is, by definition, non-blocking. Note that a strategy does not restrict the moves of the environment in any way.

Let f be a strategy. The notion of a finite run ρ being according to f is defined inductively. ε is according to *f*. Next suppose ρ is a run according to f and $\sigma = \mathrm{end}(\rho)$. If $a \in f(\rho)$ and $\delta = \langle \sigma, a, \sigma' \rangle \in TR_{TS}$, then $\rho\delta$ is a run according to *f*. Infinite runs, and the notion of infinite runs according to a strategy, are defined in the obvious way.

We say f is safe iff Err does not appear in any run according to *f*. Given an LTL specification ψ, we will say that the strategy f is ψ-winning iff the following conditions are satisfied: firstly, f is safe; secondly, if $\rho = \delta_0\delta_1\cdots$ is any infinite run of TR_{TS} according to f, then ρ is a model of ψ.

The admission controller synthesis problem is to determine, given a plant $\mathcal{P}$ and an LTL specification ψ, whether or not there exists a ψ-winning strategy. Our main result can now be stated.

Theorem 1 There is a uniform decision procedure using which one can determine for each pair $(\mathcal{P}, \psi)$ whether or not there exists a ψ-winning strategy.

Moreover, whenever a ψ-winning strategy exists, one can effectively realize a ψ-winning strategy as a finite timed automaton $\mathcal{C}$, so that the parallel composition of $\mathcal{P}$ and $\mathcal{C}$ produces only non-blocking safe runs of the plant each of which is a model of ψ.

4 The main result

Our goal here is to prove Theorem 1. The proof is based on two ideas. Firstly, as shown in Fersman's report[2], we can associate clock variables with the ready queue and capture its dynamics with the help of a mildly extended timed automaton model that uses only the queue's clock variables. The second idea is that if there is a winning strategy at all then there is one which respects the regional equivalence induced by the plant's and the queue's clock variables. Consequently, we can work with the regional version of *TS* which will be a finite transition system *RG*. Using standard techniques, it is then easy to reduce the problem of determining the existence of a winning strategy to the emptiness problem for a tree automaton $\mathcal{B}$ that we can effectively construct. The automaton $\mathcal{B}$ will run over $\{\top, \bot\}$-labeled trees whose underlying tree will be the computation tree induced by *RG*. Such a tree will be accepted by $\mathcal{B}$ iff the $\{\top, \bot\}$-labeling represents a winning strategy. This will settle the decidability problem. Further, due to Rabin's tree theorem[22], in case the language accepted by $\mathcal{B}$ is non-empty, then

it in fact accepts a regular $\{\top,\bot\}$-labeled tree. This regular tree can be effectively computed, represented as a finite structure and this structure can be naturally viewed as a timed automaton. This automaton will constitute the admission controller we seek.

4.1 Timed automaton for the ready queue

In this section, we briefly summarize the construction in Fersman's report[2], a timed automaton extended with subtraction $\mathcal{A}'_{\mathcal{Q}}$ which describes the dynamics of the ready queue under the preemptive EDF scheduling policy in a uniprocessor setting. In what follows, we will often drop the subscript $\mathcal{Q}$.

A timed automaton extended with subtraction is just an ordinary timed automaton, in which a constant integer value may be subtracted from a clock value during a transition. However such subtractions will not be allowed in arbitrary contexts. They will be used in a manner which ensures that no clock value becomes negative via subtraction. Moreover, a clock value will be subtracted from only when its value is below the maximum constant associated with it. Due to these two properties, the region construction of ordinary timed automata can also be applied to $\mathcal{A}'$.

Observe that a schedulable ready queue contains at most $\lceil D(\tau)/C(\tau)\rceil$ instances of each task type $\tau\in\Omega$. Let (τ,j) denote the j-th instance of τ, for $j=1,\cdots,\lceil D(\tau)/C(\tau)\rceil$. The function stat, to be associated with the locations of $\mathcal{A}'$ will give the status of each potential instance (τ,j).

The status of an instance can be out (not in the ready queue), in (released and in the ready queue, but has not started execution), exec (being executed by the processor) or pre (started execution but has been preempted by another instance). The set of locations of $\mathcal{A}'$ consists of these status functions and a special location Err_Q designated to represent all non-schedulable ready queues. The initial location is the status function representing the empty queue ε, denoted ε_Q.

The event alphabet is $\Omega\cup\{\mathrm{err}_Q\}$. An event $\tau\in Y$ signifies the arrival of an instance of Y. The event err_Q is a dummy one which is used only for the special location Err_Q.

For each instance (τ,j), two clocks $xc(\tau,j)$ and $xd(\tau,j)$ are used to keep track of its remaining computation time and remaining relative deadline. The set $X'_{\mathcal{Q}}$ of all such clocks is the clock set of $\mathcal{A}'$. The clocks evolve as follows.

(1) $xd(\tau,j)$ is reset to zero when (τ,j) is released, $xc(\tau,j)$ is reset to zero when it starts execution;

(2) When (τ,j) finishes execution (and exits the ready queue), we subtract $C(\tau)$ from every $xc(\sigma,k)$ whose status is pre.

Let $TS_{\mathcal{A}'}$ (TS' for short) be the transition system associated with $\mathcal{A}'$. From Fersman's report[2], we have the following.

Proposition 2 There exists a one-to-one correspondence Ξ between ready queues and configurations in TS' which satisfies:

(1) $\Xi(\varepsilon)=(\varepsilon_Q,V'_0)$, where $V'_0(x')=0$ for each $x'\in X'_{\mathcal{Q}}$, and $\Xi(\mathrm{Err})=(\mathrm{Err}_Q,V'_0)$;

(2) Suppose $\Xi(\eta_1)=(q_1',V_1')$ and $\Xi(\eta_2)=(q_2',V_2')$, let $t\in\mathbb{R}_+$ and $J=\langle\tau,C(\tau),D(\tau)\rangle$ be a task instance. Then $\eta_2=\mathrm{Comp}(\eta_1,t)$ iff $\langle(q_1',V_1'),t,(q_2',V_2')\rangle$ is a transition of TS'. Further, $\eta_2=\mathrm{Sch}(\eta_1,J)$ iff $\langle(q_1',V_1'),\tau,(q_2',V_2')\rangle$ is a transition of TS'.

Hence, from now on, we shall replace the ready queue component in configurations of TS by the corresponding configurations in TS'. To be specific, the configuration $\langle q,V,\eta,\kappa\rangle$ of TS will now be represented as $\langle q,V,q',V',\kappa\rangle$ where $\Xi(\eta)=(q',V')$.

We define the regional equivalence relation ~ to be the least equivalence relation on RC satisfying: suppose $\sigma_1=\langle q_1,V_1,q_1',V_1',\kappa_1\rangle$, and $\sigma_2=\langle q_2,V_2,q_2',V_2',\kappa_2\rangle$ are configurations of TS. Then $\sigma_1 \sim \sigma_2$ iff $q_1=q_2$, $q_1'=q_2'$, $\kappa_1=\kappa_2$ and the clock valuations $V_1 \cdot V_1'$ and $V_2 \cdot V_2'$ belong to the same clock region with respect to the clocks in $X \cup X'$, where $V \cdot V'$ denotes the clock valuation over $X \cup X'$ which agrees with V on clocks of X and agrees with V' on clocks of X'.

Using the arguments developed in Fersman et al.[2], it is easy to prove the next result.

Proposition 3 The following hold:

(1) Let σ_1,σ_2 be environment configurations of TS with $\sigma_1 \sim \sigma_2$. If $\sigma_1 \overset{t,\tau}{\Rightarrow} \sigma_3$, then there exists $t' \in \mathbb{R}_+$ such that $\sigma_2 \overset{t',\tau}{\Rightarrow} \sigma_4$ and $\sigma_3 \sim \sigma_4$;

(2) Let σ_1,σ_2 be system configurations of TS with $\sigma_1 \sim \sigma_2$. Suppose $\sigma_1 \overset{m}{\Rightarrow} \sigma_3$ and $\sigma_2 \overset{m'}{\Rightarrow} \sigma_4$, then $m=m'$ iff $\sigma_3 \sim \sigma_4$.

We are now ready to define $RG_{\mathcal{P}}$, the regional version of $\mathcal{P}$. A region of $\mathcal{P}$ is an equivalence class of ~ on RC. We define $RG=(RR,[\langle q_0,V_0,\varepsilon_{\mathcal{Q}},V_0',\#\rangle],\mathrm{ACT}_{RG},\rightsquigarrow)$ where $\mathrm{ACT}_{RG}=\Omega \cup \{0,1\} \cup \{\mathrm{err}\}$. The set of reachable regions RR and $\rightsquigarrow$ are the least sets satisfying:

(1) $[\langle q_0,V_0,\varepsilon_{\mathcal{Q}},V_0',\#\rangle] \in RR$. If $[\sigma_1] \in RR$ and $\sigma_1 \overset{t,\tau}{\Rightarrow} \sigma_2$, then $[\sigma_2] \in RR$ and $[\sigma_1] \overset{\tau}{\rightsquigarrow} [\sigma_2]$;

(2) If $[\sigma_1] \in RR$ and $\sigma_1 \overset{m}{\Rightarrow} \sigma_2$ where $m \in \{0,1\}$, then $[\sigma_2] \in RR$ and $[\sigma_1] \overset{m}{\rightsquigarrow} [\sigma_2]$;

(3) If $[\mathrm{Err}] \in RR$, then $[\mathrm{Err}] \overset{\mathrm{err}}{\rightsquigarrow} [\mathrm{Err}]$.

It follows that RR is finite and the construction of RG is effective. Call a region $[\sigma]$ an environment or system region according to whether σ is in RC^e or RC^s. RR^e and RR^s denote the sets of environment and system regions, respectively.

4.2 Region-respecting strategies

Set TR_{RG} to be $\rightsquigarrow$. We define the set of (finite) runs of RG, denoted $Runs_{RG}$, to be the least subset of $TR_{RG}^{\star}$ satisfying the following. Firstly, $\varepsilon \in Runs_{RG}$ and the end state of ε is $[\langle q_0,V_0,\varepsilon_{\mathcal{Q}},V_0',\#\rangle]$. Next, suppose $\varrho \in Runs_{RG}$ and the end state of ϱ is $[\sigma]$. If $\delta=\langle[\sigma],b,[\sigma']\rangle \in TR_{RG}$, then $\varrho\delta \in Runs_{RG}$ and the end state of $\varrho\delta$ is $[\sigma']$. By abuse of notation, we also denote the end state of a run $\varrho \in Runs_{RG}$ as $\mathrm{end}(\varrho)$.

For a transition $\delta=\langle\sigma,a,\sigma'\rangle \in TR_{TS}$, we define its projection on RG, denoted $\mathrm{Prj}_{RG}(\delta)$, to be $\delta'=\langle[\sigma],b,[\sigma']\rangle$, where $b=\tau$ if $a=(t,\tau)$; and $b=a$ otherwise. It is clear that $\delta' \in TR_{RG}$. Extend Prj_{RG} to runs of TS in the obvious way.

We say a strategy f is region respecting iff for $\rho,\rho' \in Runs$, if ρ and ρ' are according to f and $\mathrm{Prj}_{RG}(\rho)=\mathrm{Prj}_{RG}(\rho')$, then $f(\rho)=f(\rho')$.

The following observation is the key to proving decidability.

Lemma 4 There exists a ψ-winning strategy for $\mathcal{P}$, iff there exists a ψ-winning strategy which is region respecting.

Proof: Suppose f is a ψ-winning strategy for $\mathcal{P}$. We inductively construct RE_f, a "representative" prefix-closed subset of $Runs(f)$, the set of (finite) runs of TS according to f, as follows. Firstly, $\varepsilon \in RE_f$. Now suppose $\rho \in RE_f$ and $\mathrm{end}(\rho)=\sigma$ is an environment configuration. We will say that $[\sigma']$ is a region successor of σ iff there exists $\sigma'' \in [\sigma']$ and (t,τ) such that $\langle\sigma,(t,\tau),\sigma''\rangle \in TR_{TS}$. Let $[\sigma_1],[\sigma_2],\cdots,[\sigma_k]$ be the set of region successors of σ. We pick $\sigma_1' \in [\sigma_1]$, $\sigma_2' \in [\sigma_2]$, $\cdots$, $\sigma_k' \in [\sigma_k]$ and (t_i,τ_i) for each $i \in \{1,2,\cdots,k\}$, such that $\delta_i=\langle\sigma,(t_i,\tau_i),\sigma_i'\rangle \in TR_{TS}$ for each $i \in \{1,2,\cdots,k\}$. We now let $\rho\delta_i \in RE_f$ for each $i \in \{1,2,\cdots,k\}$. One may see a need for appealing to the axiom of choice here. However the choice of t_i and σ_i' can be made effectively with a bit of work. Next, suppose $\rho \in RE_f$ and $\mathrm{end}(\rho)=\sigma$ is a system

configuration. Then for each $m\in f(\rho)$, $\rho\delta\in RE_f$ where $\delta=\langle\sigma,m,\sigma'\rangle\in TR_{TS}$.

It is clear that RE_f is indeed a prefix-closed subset of *Runs*(*f*). Now suppose that $\rho\in Runs$. Then from the construction of RE_f, it is easy to see that there is at most one $\rho'\in RE_f$ such that $\mathrm{Prj}_{RG}(\rho)=\mathrm{Prj}_{RG}(\rho')$.

We now define a strategy $\hat{f}$ as follows. Let $\rho\in Runs$ and $\mathrm{end}(\rho)=\sigma$. If σ is a system configuration and there exists $\rho'\in RE_f$ such that $\mathrm{Prj}_{RG}(\rho)=\mathrm{Prj}_{RG}(\rho')$, then $\hat{f}(\rho)=f(\rho')$; otherwise, $\hat{f}(\rho)=\mathrm{Act}(\sigma)$. It is now straightforward to argue that $\hat{f}$ is a region respecting ψ-winning strategy.

4.3 Decidability

From Lemma 4, it follows that to check if a ψ-winning strategy exists, it suffices to determine if a ψ-winning region respecting strategy ("rr-strategy" for short) exists.

A strategy function is a function g which maps $Runs_{RG}$ to $2^{\mathrm{ACT}_{RG}}$ such that, if ρ is in $Runs_{RG}$ with $\mathrm{end}(\rho)=[\sigma]\in RR^e$, then $g([\sigma])=\{\tau\mid \exists\sigma',[\sigma]\overset{\tau}{\rightsquigarrow}[\sigma']\}$. If $[\sigma]\in RR^s$, then $\varnothing\neq g([\sigma])\subseteq\{0,1\}$; If $[\sigma]=[\mathrm{Err}]$, then $g([\sigma])=\{\mathrm{err}\}$. Clearly, there is a 1-1 correspondence between strategy functions and rr-strategies. We define the notion of a strategy function being ψ-winning in the expected manner. It follows that there is also a 1-1 correspondence between ψ-winning rr-strategies and ψ-winning strategy functions.

We can represent strategy functions as labeled infinite trees and design a tree automaton to recognize the set of winning rr-strategies. We first fix some basic terminology, let Γ be a finite set. A *Γ-tree* T is a prefix-closed subset of $\Gamma^\star$, elements of T are nodes with ε being the root. The set of successors of $w\in T$, denoted $\mathrm{succ}_T(w)$, is $\{wv\in T\mid v\in\Gamma\}$. An infinite path π of T is a subset of T such that $\varepsilon\in\pi$ and every node in π has exactly one successor in π. The direction of a node w, denoted $\mathrm{dir}(w)$, is defined as follows, $\mathrm{dir}(\varepsilon)$ is a special element \$ that is not in Γ. For $wv\in T$, where $v\in\Gamma$, define $\mathrm{dir}(wv)=v$. Let Σ be a finite alphabet of labels. A Σ-labeled *Γ-tree* is a pair(T,μ), where T is a *Γ-tree* and μ is a function $T\mapsto\Sigma$.

A strategy tree is a $\{\top,\bot\}$-labeled *TR_{RG}-tree* of the form $(Runs_{RG},\mu)$ satisfying: (1) $\mu(\varepsilon)=\top$; and (2) let $\rho\in Runs_{RG}$. If $\mathrm{end}(\rho)\in RR^e\cup\{\mathrm{Err}\}$, then for every successor ρ' of ρ, $\mu(\rho')=\top$. If $\mathrm{end}(\rho)\in RR^s$, then $\mu(\rho')=\top$ for at least one successor ρ' of ρ.

Clearly, there is a 1-1 correspondence between strategy trees and strategy functions. We now wish to construct a non-deterministic Büchi tree automaton which will run over $\{\top,\bot\}$-labeled *TR_{RG}-tree* of the form $(Runs_{RG},\mu)$. It will accept such a tree iff it is a strategy tree that corresponds to a ψ-winning strategy function.

In what follows, we set $\Sigma=\{\top,\bot\}$ and assume basic background concerning tree automata[23]. The tree automaton we wish to construct will of the form $\mathcal{B}=(S,S_{in},\Sigma,\Delta,\mathcal{F})$, where S is a finite set of states and $S_{in}\subseteq S$, the set of initial states and $\mathcal{F}\subseteq S$, the set of accepting states viewed as a Büchi acceptance condition. The transition relation Δ will be a subset of $\bigcup_{k=1}^{n} S\times\Sigma\times S^k$, where n is the maximum out-degree of the states of *RG*. We will assume the set of transitions of *RG* (that is TR_{RG}) to be an ordered set.

It will be convenient to view $\mathcal{B}$ as the intersection of three non-deterministic Büchi tree automata $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$. The automaton $\mathcal{B}_1$ will check if the Σ-labeled *R_{RG}-tree* is running over a strategy tree. $\mathcal{B}_2$ will check if the strategy represented is safe and $\mathcal{B}_3$ will check if every infinite run according to the strategy represented is a model of ψ.

Let $\mathcal{B}_1=(S_1,S_1^{in},\Sigma,\Delta_1,\mathcal{F}_1)$. A state in S_1 will be of the form (dir, md, lb) with dir$\in\{\$\}\cup TR_{RG}$, giving the direction from which the node that the automaton is currently visiting was arrived at. The component md$\in$ {acc, rej} ("accept", "reject") will specify the current mode of the automaton, and lb$\in\Sigma$ will specify the label the automaton expects the node it is visiting to have. S_1^{in} will be a singleton with the member ($,acc,$\top$). Now suppose $s=(\langle[\sigma],b,[\sigma']\rangle,\mathrm{md},\mathrm{lb})$ is a state, where $\{\delta_1,\delta_2,\cdots,\delta_k\}$ is the set of transitions in TR_{RG} whose first component is $[\sigma']$. Then each move in Δ_1 whose first component is s will be of the form $(s,\widehat{\mathrm{lb}},s_1,s_2,\cdots,s_k)$. Moreover, the first component of each s_i will be δ_i. A typical move will be $\big[(\delta,\mathrm{acc},\top),\bot,(\delta_1,\mathrm{rej},\top),(\delta_2,\mathrm{rej},\top),\cdots,(\delta_k,\mathrm{rej},\top)\big]$, where the automaton has guessed that the label on the node is visiting to be $\top$, but the actual label it reads is $\bot$. On this node, it will enter the rej mode and guess $\top$ (this is irrelevant actually) and propagate a copy to all the successor nodes. Once it enters the rej mode, it will just keep propagating this mode. Naturally, $\mathcal{F}_1=\big\{(\mathrm{dir},\mathrm{md},\mathrm{lb})\in S_1\mid \mathrm{md}=\mathrm{acc}\big\}$. Another typical move is $\big[(\delta,\mathrm{acc},\top),\top,(\delta_0,\mathrm{acc},\bot),(\delta_1,\mathrm{acc},\top)\big]$ with $\delta=\big([\sigma],b,[\sigma']\big)$ and $[\sigma']$ being a system region. In this case, there will be exactly two successor nodes and along the "0-direction", the automaton guesses a label $\bot$ and along the "1-direction" the label $\top$. If the end state of the current node is an environment state (or [Err]), then the automaton will guess uniquely all the successor nodes to have the label $\top$. On the other hand, if the end state of the current node is an system state, then it could guess that only the "0-successor" node has label $\top$, or that only the "1-successor" node has label $\top$ or that both have labels $\top$ signifying that the strategy is at this stage non-deterministic.

The automaton $\mathcal{B}_2$ will have a similar structure. It will also start in an accepting mode and ensure that along $\top$-paths (*i.e.* a path in which every node is labeled with $\top$), no node is encountered whose end state is the region [Err].

It is well-known[24] that the LTL-formula ψ can be represented as a non-deterministic Büchi automaton $\mathcal{B}_\psi$ running over ω-strings over the alphabet 2^{AP}. The third tree automaton $\mathcal{B}_3$ we construct will simulate $\mathcal{B}_\psi$ along $\top$-paths and accept—using the accepting states of $\mathcal{B}_\psi$—iff every $\top$-path (more precisely, its induced LTL-model) is a model of ψ.

The detailed definitions of $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$ are tedious but straightforward and we omit them due to lack of space. The required tree automaton $\mathcal{B}$ is the intersection of $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$. Clearly, the language of Σ-labeled trees of the form ($Runs_{RG}$, μ) accepted by $\mathcal{B}$ is non-empty iff there is a ψ-winning rr-strategy and this settles the first part of Theorem 1.

As for the second part, assume that there exists a ψ-winning strategy. Then by Lemma 4, there is a ψ-winning rr-strategy. Hence the language accepted by $\mathcal{B}$ is non-empty. By Rabin's tree theorem[22], $\mathcal{B}$ accepts a regular tree of the form ($Runs_{RG}$, μ) which can then be effectively represented as a finite state transition system $\mathcal{C}$. This transition system can be viewed as a timed automaton. Each location will be a node of ($Runs_{RG}$, μ) and it will be accompanied by a labeling function ξ. The initial location will be ε with $\xi(\varepsilon)=[\langle q_0,V_0,\varepsilon_Q,V_0',\#\rangle]$, whereas to all other locations ξ will assign a transition of *RG* of the form $\langle[\sigma],a,[\sigma']\rangle$. From the region $[\sigma']$, we can recover a clock constraint θ over the clocks $X\cup X'$ and view it as the clock invariant associated with the location. To be precise, we must introduce an extra clock $\hat{x}$ to ensure that the system states are urgent.

We convert each edge (l,l') of $\mathcal{C}$ to a transition as

follows. Let $\xi(l')=\delta'=\langle[\sigma],b,[\sigma']\rangle$. If $[\sigma']\in RR^s$, then there exists a unique transition $\delta''=(q,\varphi,a,Y,q')\in TR_{\mathcal{P}}$ such that $\mathrm{Prj}(\delta)=\delta''$ for every $\delta\in TR_{TS}$ with $\mathrm{Prj}_{RG}(\delta)=\delta'$. Hence we convert (l,l') to $(l,\mathrm{true},a,Y\cup\{\hat{x}\},l')$. If $[\sigma']\in RR^e$, then we convert (l,l') to $(l,\hat{x}=0,b,\varnothing,l')$. It is not difficult to show that the parallel composition of $\mathcal{P}$ and $\mathcal{C}$ produces only safe runs which are models of ψ.

According to Vardi's report[25], the complexity of emptiness testing of $\mathcal{B}$ is $O(|S|^2)$. We have $|S|=O\left(|RR|^2\times|Y|\times 2^{O(|\psi|)}\right)$[24]. Again, it is not difficult to see that $|RR|=O\left\{N_{\mathcal{P}}\times N_{\mathcal{Q}}\times 2^{O[k\log(kc)]}\times 2^d\right\}$[21]. Where $N_{\mathcal{P}}$ is the number of locations of the plant; $N_{\mathcal{Q}}$ is the number of locations of the queue automaton $\mathcal{A}'$; k is the total number of clock variables; c is the largest constant appearing across all the clock constraints and d is the total number of difference constraints.

5 Conclusion

We have formulated here a timed open system model to describe task arrival patterns and have studied the problem of synthesizing admission controllers that guarantee schedulability while meeting an LTL specification. Our work can be extended in a number of ways. One can consider using branching time requirements based on *CTL* or *CTL**. Since Lemma 4 will still go through, one can use the techniques developed in Kupferman's report[26] to solve the resulting admission controller synthesis problems.

In the present paper, we have assumed that our controller has read-only access to the clocks of the plant (and the clocks of the queue automaton). This is a justifiable assumption since the plant model merely describes the expected task arrival patterns and this information can be assumed to be available to the controller as well. Nevertheless, it will be worth exploring settings where the controller is endowed with its own clocks with pre-specified granularities. It will be equally interesting to study controllers that can interact with the ready queue, so that tasks can be rejected after they have been admitted to the ready queue. It will also be worthwhile considering settings with multi-processors, shared resources, and mixed-criticality tasks.

In summary, using the controller synthesis paradigm to design admission policies in real time tasking environments promises to be a fruitful approach.

References

[1] Buttazzo GC. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications [M]. Holland: Kluwer Academic Publishers, 1997.

[2] Fersman E, Pettersson P, Yi W. Timed automata with asynchronous processes: schedulability and decidability [C] // Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2002: 67-82.

[3] Stigge M, Yi W. Graph-based models for real-time workload: a survey [J]. Real-Time Systems, 2015, 51(5): 602-636.

[4] Liu JWS. Real-Time Systems [M]. London: Prentice-Hall, 2000.

[5] Maler O, Pnueli A, Sifakis J. On the synthesis of discrete controllers for timed systems [C] // Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science, 1995: 229-242.

[6] Asarin E, Maler O, Pnueli A. Symbolic controller synthesis for discrete and timed systems [C] // Proceedings of Hybrid Systems II, 1999: 1-20.

[7] D'Souza D, Madhusudan P. Timed control synthesis for external specifications [C] // Proceedings of

the Annual Symposium on Theoretical Aspects of Computer Science, 2002: 571-582.

[8] Bouyer P, Deepak DS, Madhusudan P, et al. Timed control with partial observability [C] // Proceedings of the International Conference on Computer Aided Verification, 2003: 180-192.

[9] Sankur O, Bouyer P, Markey N, et al. Robust controller synthesis in timed automata [C] // Proceedings of the International Conference on Concurrency Theory, 2013: 546-560.

[10] Buchi JR, Landweber LH. Solving sequential conditions by finite-state strategies [J]. Transactions of the American Mathematical Society, 1969, 138(1): 295-311.

[11] Ramadge PJ, Wonham WM. Supervisory control of a class of discrete event processes [J]. SIAM Journal on Control and Optimization, 1987, 25(1): 206-230.

[12] Pnueli A, Rosner R. On the synthesis of asynchronous reactive module [C] // Proceedings of the International Colloquium on Automata, Language, and Programming, 1989: 652-671.

[13] Ben-Abdallah H, Choi JY, Clarke D, et al. A process algebraic approach to the schedulability analysis of real-time systems [J]. Real-Time Systems, 1998, 15(3): 189-219.

[14] Kwak H, Lee I, Philippou A, et al. Symbolic schedulability analysis of real-time systems [C] // Proceedings of the IEEE Real-Time Systems Symposium, 1998: 409-418.

[15] Bertin V, Poize M, Pulou J, et al. Towards validated real-time software [C] // Proceedings of the Euromicro Conference on Real-Time Systems, 2000: 157-164.

[16] Niebert P, Yovine S. Computing optimal operation schemes for chemical plants in multi-batch mode [C] // Proceedings of the International Workshop on Hybrid Systems: Computation and Control, 2000: 338-351.

[17] Henzinger TA, Bengamin H, Meyer KC. Embedded control systems development with giotto [C] // Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems, 2001: 64-72.

[18] Altisen K, Gößler G, Sifakis J. Scheduler modeling based on the controller synthesis paradigm [J]. Real-Time Systems, 2002, 23(1-2): 55-84.

[19] Majumdar R, Saha I, Zamani M. Performance-aware scheduler synthesis for control systems [C] // Proceedings of the ACM International Conference on Embedded Software, 2011: 299-308.

[20] Burns A, Davis RI. Mixed-criticality systems: a review [Z]. 2017-01[2017-04-01]. http:// www-users.cs.york.ac.uk/~burns/review.pdf, 2017.

[21] Alur R, Dill DL. A theory of timed automata [J]. Theoretical Computer Science, 1994, 126(2): 183-235.

[22] Rabin MO. Decidability of second-order theories and automata on infinite trees [J]. Bulletin of the American Mathematical Society, 1968, 141(5): 1-35.

[23] Thomas W. Automata on infinite objects [M] // Handbook of Theoretical Computer Science, Elsevier, 1990: 133-191.

[24] Vardi MY, Wolper P. Reasoning about infinite computations [J]. Information and Computation, 1994, 115(1): 1-37.

[25] Vardi MY, Wolper P. Automatia-theoretic techniques for modal logics of programs [J]. Journal of Computer and System Sciences, 1986, 32(2): 183-221.

[26] Kupferman O, Madhusudan P, Thiagarajan PS, et al. Open systems in reactive environments: control and synthesis [C] // Proceedings of the International Conference on Concurrency Theory, 2000: 92-107.