Automata Based Interfaces for Control and Scheduling

Gera Weiss and Rajeev Alur

University of Pennsylvania
{gera,alur}@seas.upenn.edu

Abstract. We propose the use of formal languages of infinite words over the alphabet of task identifiers as an interface between control designs and software implementations. We argue that this approach is more flexible than the classical real-time scheduling framework based on periodic tasks, and allows composition of interfaces by language-theoretic operations. We show that finite automata over infinite words offer analyzable representation and can capture many interesting interface specifications such as exponential stability of switched linear systems.

1 Introduction

Modern software engineering heavily relies on clearly specified interfaces for separation of concerns among designers implementing components and programmers using those components. The interface of a component describes the functionality and constraints on the correct usage in a succinct manner. For example, the interface of a Java class describes all the methods it supports, along with the types of input and output parameters for each method, and client code is written based on this interface without much understanding of the implementation of these methods. The need for interfaces is evident for assembling complex systems from components, but more so in control applications where the components are designed by control engineers using mathematical modeling tools and invoked by software executing on digital computers. The notion of an interface for a control component must incorporate some information about timing, and standard programming languages do not provide a way of capturing such resource requirements (cf. [1, 2]).

In current practice, typically the real-time aspect of the interface of a control component is captured by a period, sometimes along with a deadline, which specifies the frequency at which the component must execute. The control engineer makes sure that the control objectives will be met as long as the component is executed consistent with its period. The software (for example, the real-time operating system) performs a worst-case execution time analysis on all the components, followed by schedulability analysis to check whether all the timing requirements can be met (cf. [3, 4]). Using period as an interface specification of a control component is simple and intuitive, but has some key deficiencies. First, a specification such as "execute the component every 5ms" does not say

whether the scheduler should or should not execute it more frequently if enough computing resources are available. Second, such specifications do not compose in the sense that a system composed of two control components cannot be specified by a single period. These deficiencies create problems for integrating control components, and this has led researchers to explore many variations of the basic real-time scheduling framework [4, 3, 5].

In this paper, we propose to use formal languages as interfaces for control components. Our approach can be best explained using an example. Consider a control component with two tasks 1 and 2. Assuming that there is a single processor that is allocated in discrete slots of some fixed duration, a schedule with respect to this component can be described by an infinite word over the alphabet $\{0, 1, 2\}$, where 0,1,2, respectively, denote that the processor executes neither, first, second task of this component. The control engineer can express the interface of the component as an ω-language (see [6] for an introduction to theory of languages of infinite words) that contains all acceptable schedules. The software must ensure, then, that the runtime allocation is in this language. The main benefit of this approach is composability: conjoining specifications of two components corresponds to a simple language-theoretic operation on interfaces (for instance, renaming of alphabet symbols and intersection). Schedulability analysis corresponds to checking the emptiness of the language of acceptable schedules. Another benefit of this approach is predictability. A mathematical formulation of sets of schedules allows for an analysis of the type used for static (fixed) schedules. This may allow dynamic scheduling for safety critical control systems [7].

More specifically, we focus on discrete-time switched linear systems, and explore the use of finite Büchi automata as interfaces. Using automata as specifications fits nicely with current trends in type systems and static analysis tools. While control engineers may be less familiar with automata, we show that a variety of scheduling constraints can be expressed as automata. A sample specification expressible by finite automata is the language of all schedules such that any subsequence of length ℓ is contracting at least by ϵ, where ℓ and ϵ are parameters. Periodic schedules can be expressed using automata, and can also be composed (for example, the set of all schedules such that 1 appears at least once every five slots and 2 appears at least once every 6 slots). The properties studied in this paper are, so called, safety properties. Safety properties are properties of systems such that every violation of a property occurs after a finite execution. Such properties can be used for online detection of violation of constraints.

While automata-based specifications are flexible, composable, and analyzable, they can express only regular languages. We show, for example, that set of all schedules that ensure stability is not regular. This fact does not necessarily means that regular languages are not suitable for the purpose of specifying scheduling constraints for control tasks. For example, the set of schedules that ensures stability is not regular because the definition of stability allows unlimited excursion from the control objective. This, of course, is not desirable in practical

applications. We show that some common stronger stability requirements, such as periodic and exponential stability, are regular.

Related Work

The use of automata as formal specification is common. Frameworks such as I/O automata [8] and interface automata [9] are focussed on capturing functionality and interaction of individual components, while timed automata have been used for schedulability analysis [10]. As far as we know, the idea of using formal languages and Büchi automata as an interface to capture the set of acceptable schedules over the alphbet of task identifiers, does not appear in the literature.

There are many approaches to scheduling safety-critical control systems [7, 11–13]. Most dynamic scheduling approaches are based on priority; that is, a task is dynamically chosen according to a priority order. In this case, the analysis of the effect of scheduling on control performance becomes difficult and inaccurate. The other popular approach is static scheduling - tasks are executed in a predetermined order. Static schedules can be analyzed to verify the effect of scheduling on control performance but they restrict the scheduling of sporadic tasks to fixed slots [14]. The approach presented in this paper is a way in the middle: it allows to model a set of schedules for which the effect of scheduling is verified.

Our work also relates to the research on stability of switched systems [15–18]. Most of the stability results for switched systems are about stability under all switching signals. This point of view regards switching as an uncontrolled signal. We, instead, identify those switchings that render the system stable. This point of view is appropriate for scheduling, where we have some control over switching but need the smallest set of constraints.

The only attempt, we know of, to use automata to describe stability related criteria for switched systems is given in [19]. For polyhedral norms, it is shown that if a switched system satisfies $\|A_i\| \leq 1$ for all $i \in I$ then the set $\{\sigma \in I^* : \|A_\sigma\| < 1\}$ is regular. It is also shown that the language $\{\sigma \in I^* : \|A_\sigma\| < \epsilon\}$ may not be context free for $\epsilon < 1$.

2 Problem formulation and main results

A discrete-time switched linear system is modeled by a finite set of real $n \times n$ matrices $\Sigma = \{A_i\}_{i \in I}$, $|I| < \infty$. Infinite words are used to describe schedules. Given a schedule $\sigma \in I^\omega$ and an initial state $x_0 \in \mathbb{R}^n$, the dynamics of the system is given by

$$x_{k+1} = A_{\sigma_k} x_k. \tag{1}$$

One interpretation of this model is as an abstraction of control mode scheduling. In this interpretation, one assumes that the choice of a control mode induces a linear transformation over the state variables. A schedule induces a sequence of transformations that gives rise to the dynamics formulated in equation (1).

Given a finite word $\sigma = \sigma_1 \cdots \sigma_l$, we will use $A_\sigma = A_{\sigma_l} \cdots A_{\sigma_1}$ to denote the transformation induced by the finite schedule σ.

In the following propositions we identify some types of constraints that are expressible by Büchi automata. A Büchi automaton is a tuple $A = (Q, I, \delta, q_0, F)$, where Q is a finite set of states with $q_0 \in Q$ an initial state, I is an alphabet, $\delta : Q \times I \to 2^Q$ is a transition function and $F \subseteq Q$ is a set of accepting states. A run of the automaton A on an infinite word $\sigma \in I^\omega$ is an infinite sequence $q_0, q_1, \ldots$ of states such that $q_{k+1} \in \delta(q_k, \sigma_k)$ for every $k \in \mathbb{N}$. The run $q_0, q_1, \ldots$ is accepting if $q_k \in F$ for infinitely many k's. An infinite word $\sigma \in I^\omega$ is accepted by A if there exists an accepting run for σ. The automaton is called *deterministic* if $|\delta(q, i)| = 1$ for all $q \in Q$ and $i \in I$. Deterministic automata give a unique run for every input word. In the setting of this paper, the alphabet is the set of task identifiers.

A safety automaton is such that there is no transition from a non-accepting state to an accepting state. Such automata specify safety properties, namely, those properties where a violation of the property can be detected after only a finite execution of the system.

Liveness properties, on the other hand, are conditions that will eventually hold. A typical example of a liveness property is: "task x is executed infinitely often". One may use such a requirement, for schedulability analysis, in early design phases when an explicit bound is not provided. Other examples of liveness properties involve interaction with the environment. Since this paper is focused on scheduling, we restricted attention to automata whose alphabet is the set of task identifier. To allow finer specifications, one may add symbols that model observations. Then, a specification such as: "if an event is detected infinitely often, eventually some process is invoked" may be useful.

In diagrams representing Büchi automata, we adopt the following graphical convention: if a symbol does not decorate any transition going out of a state then an implicit transition to a non accepting sink (a state with self loop for all the symbols) is assumed for this symbol.

The next two propositions show that many natural requirements that appear in control applications can be expressed using safety automata.

Proposition 1. *Given a switched system, the following languages can be recognized by a deterministic safety Büchi automaton:*

1. *Exponential stability: for $0 < \epsilon < 1$ and $l \in \mathbb{N}$; the language of all schedules such that any interval of length l is contracting by at least ϵ*

$$ExpStab(l, \epsilon) = \{\sigma \in I^\omega : \|A_{\sigma_{k+l}} \cdots A_{\sigma_{k+1}}\| < \epsilon \text{ for every } k \in \mathbb{N}\}.$$

2. *Directional growth: for $c \in \mathbb{R}^n$, $\delta > 1$ and $l \in \mathbb{N}$; the language of all schedules such that the projection on c grows exponentially fast,*

$$DirG(c, \delta, l) = \{\sigma \in I^\omega : |\langle c, A_{\sigma_{k+l}} \cdots A_{\sigma_{k+1}} x\rangle| > \delta |\langle c, x\rangle| \text{ for every } k \in \mathbb{N} \text{ and } x \in \mathbb{R}^n\}.$$

3. *Cost function: for positive definite matrices $Q_1, \ldots, Q_m$, horizon $h \in \mathbb{R}$ and maximal cost $J \in \mathbb{R}$;*

$$Cost(Q_1, \ldots, Q_m, h, J) = \{\sigma \in I^\omega : \sum_{k=k_0}^{k_0+h} \sum_{i=1}^{m} x_k^T Q_i x_k < J \text{ for every } k_0 \in \mathbb{N} \text{ and } x_{k_0} \in \mathbb{R}^n\}.$$

The above languages are examples of control performance type of constraints. These constraints refer to the model of the system. In the next proposition, we give some languages that represent constraints that are not directly related to the model of the plant as a switched system. These languages represent external considerations such as sensor or actuator usage constraints. They can also represent scheduling constraints that come from the implementation (e.g. if the implementation can only handle periodic schedules).

Proposition 2. *The following languages can be recognized by a deterministic safety Büchi automaton:*

1. *Minimal separation: for $i, j \in I$ and $m_{i,j} \in \mathbb{N}$; the language of all schedules that separate the schedule of mode i from mode j by at least $m_{i,j}$ slots,*

$$MinSep(i, j, m_{i,j}) = \{\sigma \in I^\omega : \sigma_k = i \implies j \notin \{\sigma_{k+1}, \ldots, \sigma_{k+m_{i,j}}\}\}.$$

2. *Maximal separation: for $i, j \in I$ and $M_{i,j} \in \mathbb{N}$; the language of all schedules that separate the schedule of mode i from mode j by at most $M_{i,j}$ slots,*

$$MaxSep(i, j, M_{i,j}) = \{\sigma \in I^\omega : \sigma_k = i \implies j \in \{\sigma_{k+1}, \ldots, \sigma_{k+M_{i,j}}\}\}.$$

3. *Periodic execution: for $i \in I$ and $P_i \in \mathbb{N}$; the language of all schedules that execute mode i every P_i slots,*

$$Per(i, P_i) = \{\sigma \in I^\omega : \sigma_k = i \implies \sigma_{k+P_i} = i \text{ and } \sigma_{k+j} \neq i \text{ for all } j = 1, \ldots, P_i - 1\}.$$

4. *Dependency: for a relation $R \subset I \times I$; the language of all schedules such that mode i executes after mode j only if $(i, j) \in R$,*

$$Dep(R) = \{\sigma \in I^\omega : (\sigma_k, \sigma_{k+1}) \in R \text{ for every } k \in \mathbb{N}\}.$$

5. *Sequential execution: for an ordered tuple $(i_1, \ldots, i_l) \subseteq I^l$; the language of all schedules such that modes $i_1, \ldots, i_l$ are executed periodically in a sequence (other modes may get in between),*

$$Seq(i_1, \ldots, i_l) = \{\sigma \in I^\omega : \pi(\sigma, \{i_1, \ldots, i_l\}) \text{ is a prefix of } (i_1 \cdots i_l)^\omega\}.$$

where $\pi(\sigma, S)$ denotes the projection of the word σ over the set S, i.e., the word obtained by deleting from σ all the letters not in S.

6. *Cyclic schedules: for $P \in \mathbb{N}$; the set of cyclic schedules with cycle length C*

$$Cyc(C) = \{\sigma \in I^\omega : \sigma_{k+C} = \sigma_k \text{ for all } k \in \mathbb{N}\}$$

Since the set of languages expressible by deterministic safety Büchi automata is closed under finite intersections and unions, disjunctions and conjunctions of the above constraints are also expressible. For example, we can express that modes i and j should interlace every 10 execution slots by the formula $Per(i, 20) \cap Per(j, 20) \cap MinSep(i, j, 10) \cap MaxSep(i, j, 10)$. Of course, finite automata can model many other languages. The above list is only a collection of examples, put together in order to convince the reader that many typical specifications are expressible using automata.

The above propositions, together with the succeeding example, show that control engineers do not have to construct automata manually. We envision a tool that accepts specifications in a language tailored to allow easy scheduling specifications for control systems. The tool may take specifications of the form exhibited in the proposition, allowing to combine them using logical operators. The output of this tool can be a finite automaton representing all acceptable schedules for the system.

From the software engineering perspective, a representation of scheduling constraints by finite automata lets a scheduling mechanism use all the flexibility allowed by the control system. This gives the flexibility of dynamic scheduling with the predictability of static scheduling. Assume, for example, that we want to control several systems by the same computer. Each system poses some scheduling constraints. If these constraints are expressed by automata, the intersection can be easily computed. If the intersection is not empty, the system is schedulable. In this setting, each subsystem has a distinct matrix A_0 that models the behavior of the system when the processor is assigned to control another subsystem. In other settings, it can be that controlling one system has an affect on the other systems. Such dependencies can be easily incorporated into the automata composition algorithm using an adequate renaming of the alphabet. This approach also allows to schedule non-control tasks more efficiently. The automaton that represents the intersection can also be used as an admission control mechanism (if we want to allow online registration of new subsystems).

For completeness, we exhibit some limitations of using automata to express scheduling constraints for control systems. In the following proposition, we identify a type of constraint not expressible using automata. A language of finite words is called *regular* if it can be accepted by a finite automaton.

Proposition 3. *For a switched system, if there exits $i \in I$ such that A_i is not stable (one of its eigenvalues is not in the complex unit disc) then the language*

$$L = \{\sigma \in I^* : \|A_\sigma\| < 1\}$$

is either empty or not regular.

This result implies that we cannot model the set of all finite schedules, $\sigma \in I^*$, such that $\|A_\sigma x\| < \|x\|$ for every $x \in \mathbb{R}^n$. This language corresponds to the set of periodic schedules for which every period is contracting.

Intuitively, the reason this language is not regular is because a contracting word may begin with an arbitrarily long expanding prefix. The only way to check that an expansion is compensated is to count arbitrarily high (a formal proof is given in Section 5 below). We argue that the key property that makes this language irregular, also makes it inadequate for scheduling specifications of control modes. Requiring that the transformation be contracting does look natural at the first glance, since it means that perturbations are steered to equilibrium. However, allowing an arbitrarily long expansion is not usually acceptable because it means that perturbations may explode before they are cleared. To fix this problem we offer constraints of the type given in Proposition 1, where an explicit bound on the maximal length of an excursion is given. Such requirements capture practical control constraints and are describable by finite automata.

3 Compositional schedulability analysis

Consider a control system composed of l subsystems of dimensions $n_1, \ldots, n_l$ (the state space of the ith subsystem is $\mathbb{R}^{n_i}$, $i = 1, \ldots, l$, and the state space of the whole system is $\mathbb{R}^n = \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_l}$). Assume that the system is controlled by a single computer and that the control engineer designed m control tasks.

In each computation slot, the states of the subsystems are transformed, depending on the task scheduled for execution in this slot. Assuming that the transformations are linear and time invariant, we can model this dependency by a map

$$A : \{1, ..., m\} \to \mathbb{R}^{n_1 \times n_1} \times \cdots \times \mathbb{R}^{n_l \times n_l}$$

that takes a task identifier to a list of matrices. The ith matrix in $A(j)$ models the transformation of the state of the ith subsystem when task j is scheduled for execution.

Given a requirements specification for each subsystem, based on the propositions in Section 2, we can compute a Büchi automaton for every subsystem. This automaton specifies the sequences of task executions that will not violate the specification of the subsystem.

Since there are efficient algorithms to find an automaton recognizing the intersection of the languages recognized by a finite set of automata, there is an efficient way to get the set of schedules that keep all subsystems within their requirements. An empty intersection means that the system is not schedulable. A nonempty intersection can be used for dynamic or static scheduling mechanisms, or serve for further schedulability analysis (by intersecting it with another automaton).

When a new subsystem is added, we do not need to repeat all the computation. Once the automaton specifying the requirements for the new subsystem is computed, it can be intersected with the product of the other subsystems.

Assume that we want to add a control task $m+1$ such that the ith matrix in the list $A(m+1)$ is the same as the ith matrix in $A(j)$ for some $j \in \{1, \ldots, m\}$. This happens when the effect of task $m+1$ on the ith subsystem is the same as the effect of the task j on that subsystem. In that case, the automaton for the

requirements of the ith subsystem can be adjusted incrementally by adding an $m+1$ transition whenever a j transition exists.

One practical example, is when each task is designed to regulate a specific subsystem. In this case, each subsystem is assigned with a list of matrices. The list $L_i = (A_0, \ldots, A_{m_i})$ describes the possible transformations of the state of the ith system. The first element in this list, A_0, models the response of the system when the controller attends another subsystem. Then, task identifiers are pairs (i, j) and the global map is

$$A(i, j) = (L_1(0), \ldots, L_i(j), \ldots, L_l(0)).$$

Namely, when the jth transformation of the ith subsystem is scheduled for execution, all other subsystems are transformed according to the first entry in their list. In this case, schedulability analysis is completely compositional: when a new subsystem is defined, schedulability analysis reduces to computing the product of the automaton for the new subsystem with the automaton for the rest of the system (in both automata, the missing alphabet symbols are identified with the zero symbol).

4 Examples

Consider the system $\Sigma = \{A_1, A_2\}$ where $A_1 = \begin{pmatrix} 2 & -\frac{7}{4} \\ 2 & -2 \end{pmatrix}$ and $A_2 = \begin{pmatrix} \frac{1}{4} & \frac{7}{4} \\ \frac{1}{4} & -\frac{1}{4} \end{pmatrix}$. Both A_1 and A_2 are stable (their eigenvalues lie in the unit ball of the complex plain), but their product is not stable. In particular, the schedule $\sigma = 1212\ldots$ steers the initial state $\big(1, 1\big)^T$ arbitrarily far from the origin and so does the schedule $\sigma = 2121\ldots$ to the vector $\big(1, 0\big)^T$.

Figure 1 depicts an automaton that models the set of schedules that satisfy $ExpStab(4, 1) = \{\sigma \in \{1, 2\}^\omega : \|A_{\sigma_{k+4}} \cdots A_{\sigma_{k+1}}\| < 1 \text{ for every } k \in \mathbb{N}\}$. This automaton was constructed by first computing the set

$$\{1112, 1121, 1211, 1212, 1222, 2111, 2121, 2122, 2212, 2221\}$$

of the words of length four that are not contracting. Then, an automaton that rejects any infinite word that contains one of these four letter words as a subword was derived.

Assume that, in addition to the above constraint, we are not allowed to apply the first mode consecutively more than two times. Such a constraint may arise when A_1 models the use of some sensor or actuator that needs a time to re-calibrate after two consecutive operations. This is captured by the language $MaxCon(1, 2)$ whose automaton is given in Figure 2. The language that captures both constraints together is $ExpStab(4, 1) \cap MaxCon(1, 2)$ whose automaton is given in Figure 3.

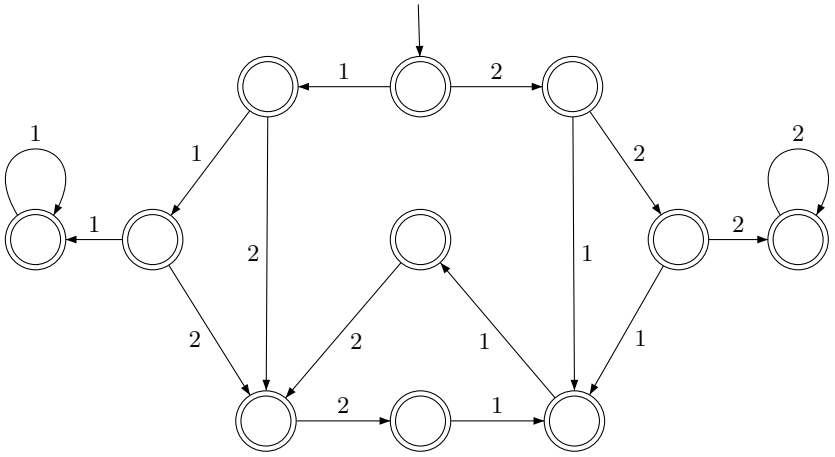

Fig. 1. A deterministic Büchi automaton for $ExpStab(4, 1)$.

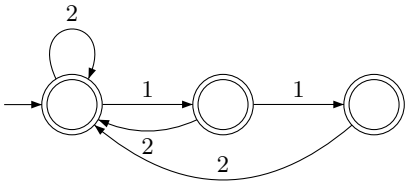

Fig. 2. A deterministic Büchi automaton for $MaxCon(1, 2)$.

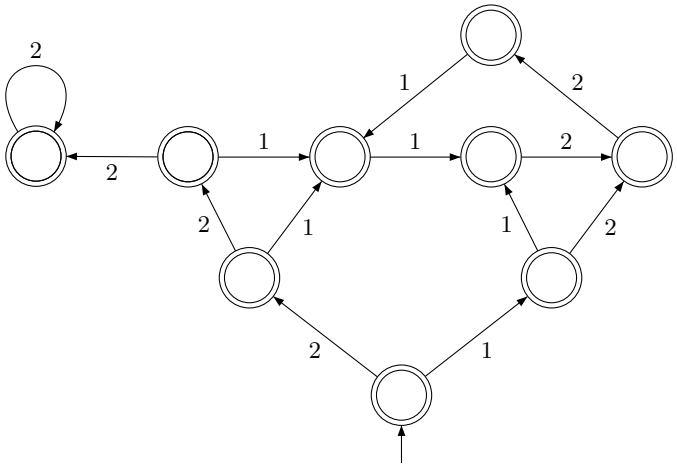

Fig. 3. A deterministic Büchi automaton for $ExpStab(4, 1) \cap MaxCon(1, 2)$.

5 Proofs

In this section we give the proofs of the propositions brought in Section 2. The proofs are constructive. In particular the proofs of Proposition 1 and Proposition 2 suggest algorithms to compute the automata representations. The constructions provided in the proofs will not always give the smallest possible automata. For practical use, an automata minimization algorithm may be needed (cf. [20]).

Since this paper is intended for both control theoretic and computer science audience, we include some details that may seem trivial to some readers.

Proof (of Proposition 1).

The language $ExpStab(l, \epsilon)$ is the language of all infinite words that avoid any word $\sigma \in I^l$ such that $\|A_\sigma\| \geq \epsilon$. For each such word we can construct the automaton that describes the language of infinite words that avoid it as a sub-word (see Figure 4 for an example), and then take the intersection. Note that there are only finite number of such words. Similarly, the language $DirG(c, \delta, l)$ is the language of of all words such $|\langle c, A_\sigma x\rangle| > \delta|\langle c, x\rangle|$ for every subword σ of length l and any $x \in \mathbb{R}^n$. For each $\sigma \in I^l$, we can compute $B_\sigma := \min_{x\in\mathbb{R}^n} \frac{|\langle c, A_\sigma x\rangle|}{|\langle c, x\rangle|}$. Then, $DirG(c, \delta, l)$ is the language of infinite words in which none of the words $\{\sigma \in I^l : B_\sigma < \delta\}$ appear as a subword. Again,there is only a finite number of such words. For the language $Cost(Q_1, \ldots, Q_m, h, J)$, consider the set $S = \{\sigma \in I^l : \max_{x\in\mathbb{R}^n} \sum_{i=1}^{l} \sum_{j=1}^{m} (A_{\sigma_i} \cdots A_{\sigma_1} x)^T Q_j (A_{\sigma_i} \cdots A_{\sigma_1} x) \geq J\}$. The language $Cost(Q_1, \ldots, Q_m, h, J)$ is the set of all infinite words that do not contain any of the words in S as a subword. □

Proof (of Proposition 2).

Figure 5 depicts an automaton that accepts $MinSep(i, j, m_{i,j})$. The automaton counts the letters different from j after each occurrence of i and moves to a non-accepting state when a j is too close to an i. Since we do not need to count more than $m_{i,j}$, the automaton is finite. The automaton for $MaxSep(i, j, M_{i,j})$ is similar. We need to carry the same counting but accept only words for which the counter does not exceed the limit. See Figure 6 for an automaton that accepts $Per(i, 3)$. Generalization to arbitrary period is straightforward. Figure 7 depicts an automaton that accepts only words in which only mode j is allowed to follow mode i. If we are given a relation, R, as a set of ordered pairs, we can construct an automaton as in Figure 7 for each pair and take the union of the languages. See Figure 8 for an automaton that accepts the language $Seq(1, 2, 3)$. This example can be easily extended to longer sequences and other indices. For the language $Cyc(C)$, consider the automaton depicted in Figure 9. Let $L(i, m, C)$ be the language of all infinite words such that, starting with the mth letter, Cth letter is again i. This language can be accepted by an automaton similar to the one given in Figure 9 (possibly with different initial and cycle counts). Then, the language $Cyc(C)$ is given as the union of $L(i, m, C)$ over all $i \in I$ and $m = 1, \ldots, C$. □

Proof (of Proposition 3).

Assume towards contradiction that the language is regular but not empty. By the pumping lemma, there is $p > 0$ such that for every $w \in L$ there are words $x, y, z \in I^*$ such that $w = xyz$, $|xy| \leq p$, $|y| > 0$ and $xy^k z \in L$ for every $k \geq 0$. Since L is not empty there is a word $u \in L$. In particular A_u is contracting. There must be an $m \in \mathbb{N}$ such that $\|A_u^m A_i^p\| < 1$ because $\|A_u^m A_i^p\| \leq \|A_u\|^m \|A_i\|^p$ which is smaller than one for every m larger than $\log_{\|A_u\|}(\|A_i\|^{-p})$. Consider the word $w = i^p u^m$. This word is in L because $\|A_u^m A_i^p\| < 1$. By the pumping lemma, the words $i^l u^m$ must also belong to L for every $l \in \mathbb{N}$. But this leads to a contradiction because we allow arbitrarily large expansion before a fixed contraction which will eventually give an expanding product. □

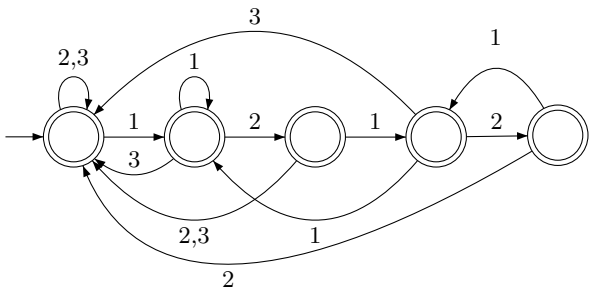

Fig. 4. A deterministic Büchi automaton that accepts all infinite words that avoid the subword 12123.

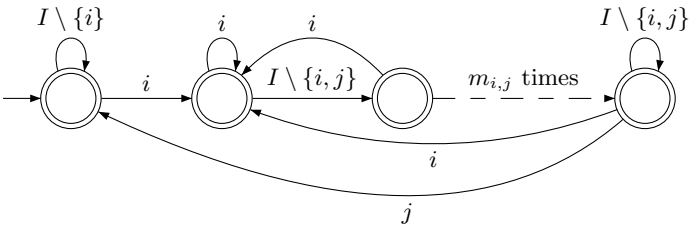

Fig. 5. A deterministic Büchi automaton that accepts $MinSep(i, j, m_{i,j})$.

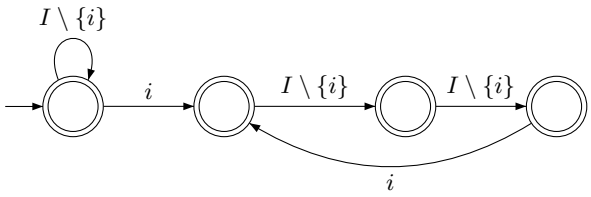

Fig. 6. A deterministic Büchi automaton that accepts $Per(i, 3)$.

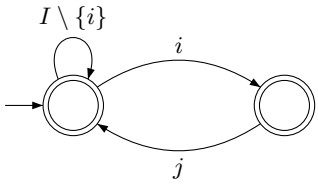

Fig. 7. A deterministic Büchi automaton that accepts $Dep(\{(i, j)\})$.

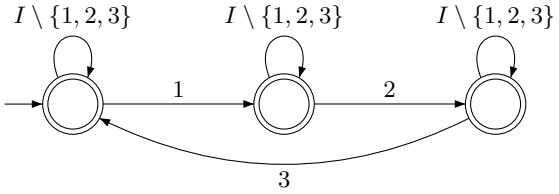

Fig. 8. A deterministic Büchi automaton that accepts $Seq(1, 2, 3)$.

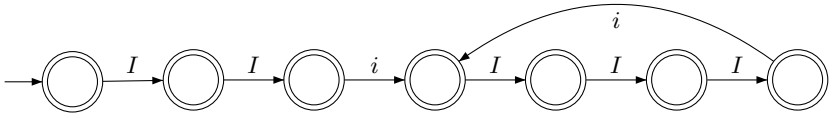

Fig. 9. A deterministic Büchi automaton that accepts words with i as the third letter and then every fourth letters.

Acknowledgments

We thank George Pappas, Oded Maler and Truong Nghiem for fruitful discussions. This research was supported by NSF grants CCR 0410662 and CSR-EHS 0509143.

References

1. Sastry, S., Sztipanovits, J., Bajcsy, R., Gill, H.: Modeling and design of embedded software. Proceedings of the IEEE **91**(1) (2003)
2. Lee, E.: What's ahead for embedded software. IEEE Computer (2000) 18–26
3. Kopetz, H.: Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers (2000)
4. Buttazo, G.: Hard real-time computing systems: Predictable scheduling algorithms and applications. Kluwer Academic Publishers (1997)
5. Shin, I., Lee, I.: Compsitional real-time scheduling framework. In: Proceedings of the 25th IEEE Real-Time Systems Symposium. (2004)
6. Thomas, W.: Automata on infinite objects. In van Leeuwen, J., ed.: Handbook of Theoretical Computer Science. Volume B. Elsevier Science Publishers (1990) 133–191
7. Balarin, F., Lavagno, L., Murthy, P., Sangiovanni-vincentelli, A.: Scheduling for embedded real-time systems. IEEE Design and Test of Computers **15**(1) (1998) 71–82
8. Lynch, N., Segala, R., Vaandrager, F.: Hybrid I/O automata. Information and Computation **185**(1) (2003) 105–157
9. de Alfaro, L., Henzinger, T.: Interface theories for component-based design. In: Embedded Software, First International Workshop. LNCS 2211, Springer (2001) 148–165
10. Abdeddaïm, Y., Maler, O.: Job-shop scheduling using timed automata. In: CAV 01: Proc. of 13th Conf. on Computer Aided Verification. LNCS 2102, Springer (2001) 478–492
11. Balbastre, P., Ripoll, I., Crespo, A.: Control tasks delay reduction under static and dynamic scheduling policies. rtcsa **00** (2000) 522
12. Bate, I., Burns, A.: A framework for scheduling in safety-critical embedded control systems. In: RTCSA '99: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications, Washington, DC, USA, IEEE Computer Society (1999) 46
13. Cervin, A.: Improved scheduling of control tasks. In: Proceedings of the 11th Euromicro Conference on Real-Time Systems, York, UK (1999) 4–10
14. Audsley, N., Tindell, K., Burns, A.: The end of the line for static cyclic scheduling? In: Real-Time Systems, 1993. Proceedings., Fifth Euromicro Workshop on on Real-Time Systems. (1993)
15. Blondel, V.D., Tsitsiklis, J.N.: The boundedness of all products of a pair of matrices is undecidable. Systems Control Lett. **41**(2) (2000) 135–140
16. Hespanha, J., Morse, A.: Stability of switched systems with average dwell-time (1999)
17. Liberzon, D.: Switching in systems and control. Systems & Control: Foundations & Applications. Birkhäuser Boston Inc., Boston, MA (2003)

18. Schultz, P.: Research Problems: Mortality of 2×2 Matrices. Amer. Math. Monthly **84**(6) (1977) 463–464
19. Gurvits, L.: Stability of discrete linear inclusion. Linear Algebra Appl. **231** (1995) 47–85
20. Etessami, K., Holzmann, G.J.: Optimizing Büchi automata. Lecture Notes in Computer Science **1877** (2000) 153+