Acceptance and Random Generation of Event Sequences under Real Time Calculus constraints

Kajori Banerjee and Pallab Dasgupta
Dept. of Computer Science & Engineering, Indian Institute of Technology Kharagpur, India - 721302
Emails: {kajori.banerjee,pallab}@cse.iitkgp.ernet.in

Abstract— Simulation platforms for complex networked real time systems require random input pattern generators for simulating input distributions. They also require monitors for checking whether the output of the system satisfies the desired throughput. In this paper we study the acceptance and generation problems in a setting where the constraints defining the input distributions as well as the constraints defining the expected output distributions are specified in real time calculus (RTC). We prove that event patterns satisfying a given set of RTC constraints can be described by a ω-regular language. We propose a method for constructing an automaton that can be used for online generation of random admissible event patterns. This is significant, considering the known problems of deadlock in less informed generators for streams satisfying RTC constraints.

I. Introduction

Model based evaluation of complex architectures for real time embedded systems has become an important requirement in practice. The model of the system is simulated with constrained random inputs and the output of the system is studied for conformance with expected performace requirements. Traditionally input distributions have been described through probability distributions (such as Poisson, Exponential, Erlang, etc) and outputs have been monitored using statistical metrics (such as worst case latency, average throughput, etc).

With the increasing complexity of networked embedded systems, it is not always possible to fit complex input and service patterns into standard probability distributions. In recent times, Real Time Calculus (RTC) [1] has been used extensively to provide a succinct summary for event patterns in complex networked embedded systems [2]–[6]. The primary motivation for this has been to facilitate analytical evaluation of the system.

We believe that RTC can also be used as a specification language, both for specifying constraints on input patterns (which can be used to constrain the random input generator in a simulation platform) and for specifying requirements that the output patterns must satisfy (which can be used to monitor the output during simulation). To enable the former, we require a random timed sequence generator from RTC constraints, and for the latter we require an acceptor.

An RTC constraint specifies a lower and upper bound on the number of events in a given interval of time. A finite RTC specification, which consists of a finite set of RTC constraints. Automatic extraction of RTC constraints from a given set of timed event patterns is a well studied problem [7].

The extracted RTC constraints can be used to generate inputs in a constrained random simulation environment for a target architecture. For example, finite sets of RTC constraints on various types of activities in a cell phone (such as calls, SMS, music, etc) can be extracted from customer workload patterns on cell phone usage, and then these RTC constraints can be used to generate constrained random inputs for simulating a new cell phone architecture. This provides better coverage than simulating the new architecture with only the given workload traces.

RTC specifications may be also be used to specify the design intent. For example, we may specify the desired reliability of a communication system such at most two messages are lost from every 10 consecutive frames, and at most five messages are lost from every 30 consecutive frames. This requirement can be expressed in RTC as, $R = \{\langle 10, 0, 2\rangle, \langle 30, 0, 5\rangle\}$. Such a specification may be used in verifying the reliability of the model of the communication system.

RTC has been studied in the context of discrete time [8]–[11] as well as in the context of dense time [7], [10], [11]. Since simulation platforms necessarily assume a discrete granularity of time (called a time step), the focus of this paper in only on the discrete model of time. Therefore we are concerned with the events that happen in a time step, but we are not concerned with the exact time at which they happen within that time step.

The problem of generating random event patterns that agree with a given set of RTC constraints is a non-trivial problem [10], [11], because not all valid finite event patterns can be extended to a valid infinite event sequence. Therefore a less informed generator may reach a forbiden state from which it is impossible to proceed without violating a constraint. Though this problem has been observed and studied [9]–[11], the existing techniques for identifying the forbidden states are computationally too expensive to be used on-the-fly for generating random event patterns from a given finite set of RTC constraints.

The problem of reaching forbidden states does not exist when we consider the task of accepting a given event sequence with respect to a given set of RTC constraints. Automata based acceptors for event patterns satisfying a given set of RTC constraints has been studied in the past, with various proposals including timed automata [7], [12] and event count automata [8] based formulations. Since these formulations

978-3-9815370-2-4/DATE14/©2014 EDAA

are primarily intended to develop acceptors, the problem of forbidden states was not addressed in these papers.

In this paper we show that patterns satisfying a given finite set of RTC constraints define an interesting fragment of ω-regular languages [13]. While ω-regular languages are accepted by non-deterministic Büchi automata [14], we show that for this special fragment it is possible to construct a deterministic finite automata (DFA) that can be modified to act as an acceptor as well as a generator for the language defined by the RTC constraints. Since this automaton can be constructed apriori, the proposed formulation is well suited for generating random event patterns on-the-fly. We believe that this paper presents for the first time an automata based approach for generating random arrival patterns that are admissible with a given set of RTC constraints.

The organization of the paper is as follows. Section II studies the relationship between RTC and ω-Regular Languages and demonstrates an approach for developing an deterministic automaton for accepting the language defined by RTC constraints. Section III studies the problem of developing a automata theoretic approach towards generating random event sequences satisfying given RTC constraints. Section IV presents the conclusions.

II. Real Time Calculus and ω-Regular Languages

A finite *Real Time Calculus* (RTC) specification, R, consists of a finite set of RTC constraints. Formally, a RTC constraint is a 3-tuple, $\langle \Delta, \alpha^L(\Delta), \alpha^U(\Delta) \rangle$, where $\alpha^L(\Delta)$ and $\alpha^U(\Delta)$ respectively specify the minimum and maximum number of events within every time interval of Δ.

Let η be the minimum among the $\alpha^U(\Delta)$ values of the constraints in R. Then η is an obvious upper bound on the number of events that can happen in an unit of time. We define Σ to be the set of integers in the range $[0, \eta]$.

A word $\mathcal{W} = w_1, w_2, ...$ is an infinite sequence of integers over Σ, where each w_i represents the number of events that have occurred during the unit time interval $[i-1, i)$. A word $\mathcal{W}$ is said to be *admissible* with respect to the constraint $\langle \Delta, \alpha^L(\Delta), \alpha^U(\Delta) \rangle$, if the sum of the number of events occurring in every sequence of length Δ in $\mathcal{W}$ lies in between $\alpha^L(\Delta)$ and $\alpha^U(\Delta)$ (boundary values included).

Formally for every $i > 0$,

$$\alpha^l(\Delta) \leq \sum_{j=i}^{i+\Delta-1} w_j \leq \alpha^u(\Delta)$$

Example 1. Consider the RTC constraint $\langle 3, 4, 7 \rangle$. The word $\mathcal{W}_1 = (2, 1, 3, 1)^\omega$ is admissible with respect to this RTC constraint. On the other hand, the word:

$$\mathcal{W}_2 = 2, 1, 3, 1, 2, 1, 0, 1, (3, 1, 2)^\omega$$

is not admissible because $w_6 + w_7 + w_8 < \alpha^L(3)$, that is, $1 + 0 + 1 < 4$. □

A word is *admissible* or *valid* with respect to a set R of RTC constraints, iff it is admissible with respect to each constraint in R.

Let $\mathcal{Z}$ be the set of Δ-length sequences that violate a given RTC constraint, $A = \langle \Delta, \alpha^L(\Delta), \alpha^U(\Delta) \rangle$. Then the language consisting of the admissible infinite patterns with respect to this constraint can be defined as the following ω-regular language:

$$\mathcal{L}(A) = \Sigma^\omega - \Sigma^* \rho \Sigma^\omega, \quad \text{where, } \rho \in \mathcal{Z}$$

Theorem 1. *The event patterns that are admissible with respect to a given finite set of RTC constraints define a ω-regular language.*

Proof: Given two RTC constraints, A_1 and A_2, the language of patterns admissible with respect to both of them is the intersection of the ω-regular languages, $\mathcal{L}(A_1)$ and $\mathcal{L}(A_2)$. Since ω-regular languages are closed under intersection, we conclude that the language $\mathcal{L}(A_1) \cap \mathcal{L}(A_2)$ is ω-regular. The proof follows from the application of intersection over the given set of RTC constraints. □

It is in fact quite easy to construct an acceptor for the ω-regular languages defined by RTC constraints. This is because RTC constraints are essentially *safety properties*, that is, an inifinite word refuting an RTC constraint must necessarily have a finite prefix that violates the constraint. The set $\mathcal{Z}$ of Δ-length sequences that violate a RTC constraint, $A = \langle \Delta, \alpha^L(\Delta), \alpha^U(\Delta) \rangle$, is finite. Therefore the following language is a regular language:

$$\mathcal{L}(A') = \Sigma^* \rho \Sigma^*, \quad \text{where, } \rho \in \mathcal{Z}$$

The language, $\mathcal{L}(A')$, defines the set of finite sequences that violate the RTC constraint, A. A DFA, $D(A')$ for accepting this language will get trapped in a final state whenever a pattern in $\mathcal{Z}$ is detected in its input.

By exchanging the final and non-final states of $D(A')$, and by applying the Büchi acceptance criterion (that is, some final state is visited infinitely often), it is easy to see that we get an acceptor for the language $\mathcal{L}(A)$. Intuitively this means that any infinite walk on $D(A')$ which avoids the final states of $D(A')$ defines a word of $\mathcal{L}(A)$.

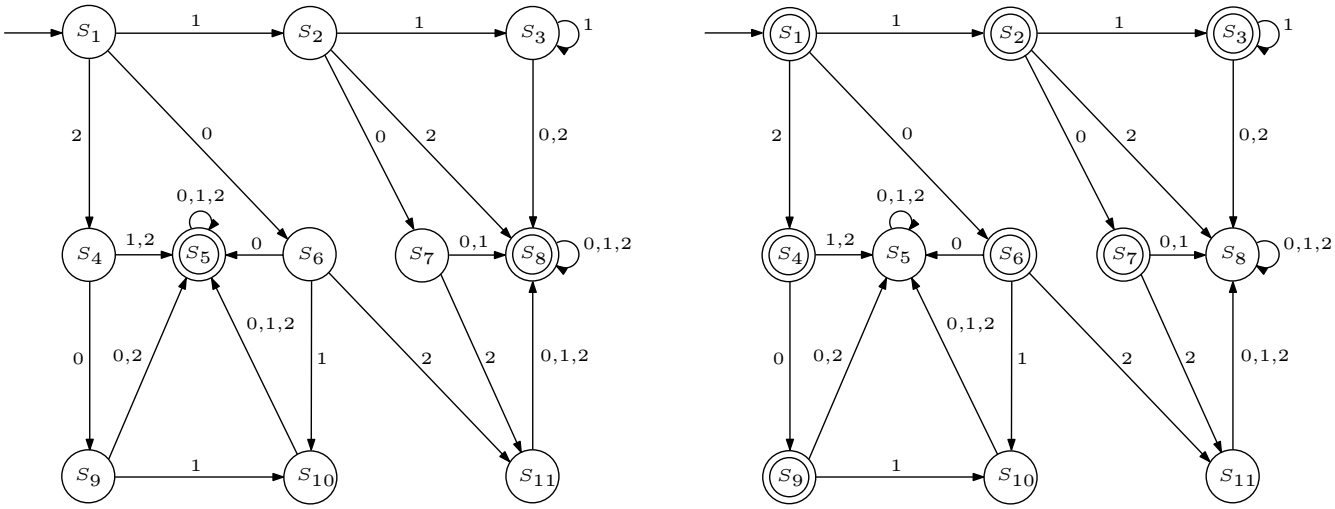
Example 2. Consider the following set of RTC constraints :

$$A = \{\langle 2, 1, 2 \rangle, \langle 3, 3, 3 \rangle\}$$

Figure 1 shows the DFA, $D(A')$, and the corresponding Büchi automaton, $B(A)$. It is easy to see that the accepting runs of $B(A)$ are precisely the infinite walks in $D(A')$ that avoid its final states. The sequence of states $s_1, s_2, s_3, s_3, s_3, ..$ is an infinite run on $D(A')$ avoiding its final states and also an accepting run on $B(A)$. On the other hand, $s_1, s_4, s_5, s_5, s_5, ..$ is an infinite run on $D(A')$ but not an accepting run on $B(A)$. □

III. Random Generation of Admissible Event Sequences

In the previous section we showed that the DFA, $D(A')$, for accepting invalid finite sequences with respect to a RTC specification can be transformed to a deterministic Büchi



(a) $D(A')$: DFA for $\mathcal{L}(A')$ (b) $B(A)$: Büchi automaton for $\mathcal{L}(A)$

Fig. 1: Automata for constraints given in Example 2

automaton for accepting valid infinite event sequences. Any infinite walk on $D(A')$ that avoids the final states of $D(A')$ defines a valid word.

Typically, $D(A')$ may contain states from which reaching a final state is inevitable. Any valid random walk needs to avoid these states as well. This section develops the theoretical basis for this requirement based on the core tennets of real time calculus and then proposes an algorithmic solution to the task of generating random event sequences.

An on-the-fly sequence generator will essentially work by extending the sequence generated so far with a chosen number of events for the next time frame. It needs to determine for each time frame whether the chosen number of events leads to any immediate or future violation of the RTC constraints.

Let $\mathcal{L}(R)$ denote the language defined by the words satisfying the given set R of RTC constraints.

Definition 1 (Prefix)**.** *Let $\mathcal{W} = w_1, w_2, .., w_j, w_{j+1}...$ be a word over an alphabet Σ. A prefix of length j of $\mathcal{W}$ is the finite sequence $w_1, w_2, ...w_j$. Let Pref($\mathcal{W}$) denote the set of all finite length prefixes of the word $\mathcal{W}$, and let Pref($\mathcal{L}$) denote the set of all finite length prefixes of the words in the language $\mathcal{L}$.* □

Definition 2 (Invalid Prefix)**.** *A prefix $w_1, w_2, ...w_l$ belonging to Pref($\mathcal{W}$) is invalid with respect to a given set R of RTC constraints, if it contains a sequence violating a member of R. Let InvPref($\mathcal{L}(R)$) denote the set of all invalid prefixes in the words not belonging to the language of R.* □

It is obvious that valid words do not contain any invalid prefix, that is,

$$InvPref(\mathcal{L}(R)) \bigcap Pref(\mathcal{L}(R)) = \{\}$$

However there exists finite sequences that do not violate any member of R and yet they do not belong to any valid infinite word. Formally:

$$\Sigma^* \neq Pref(\mathcal{L}(R)) \bigcup InvPref(\mathcal{L}(R))$$

The following example illustrates the existence of such sequences.

Example 3. Consider the following set R of RTC constraints:

$$R = \{\langle 3, 0, 3\rangle, \langle 5, 5, 9\rangle\}$$

Let us consider the prefix $P = 0, 2, 1, 0, 2, 0$, representing the sequence of events generated upto the 6^{th} time frame. It is easy to verify that P is a valid prefix as per Definition 2.

Let us now consider the problem of extending P to a valid prefix of length 7. Let w_i denote the number of events at time instance $(i-1, i]$. Therefore from P, we have $w_1 = 0, w_2 = 2, w_3 = 1, w_4 = 0, w_5 = 2, w_6 = 0$, and our goal is to determine a value for w_7 such that P concatenated with w_7 is also valid.

R imposes the following constraints involving w_7:

$$[0 \leq w_5 + w_6 + w_7 \leq 3],$$
$$[5 \leq w_3 + w_4 + w_5 + w_6 + w_7 \leq 9]$$

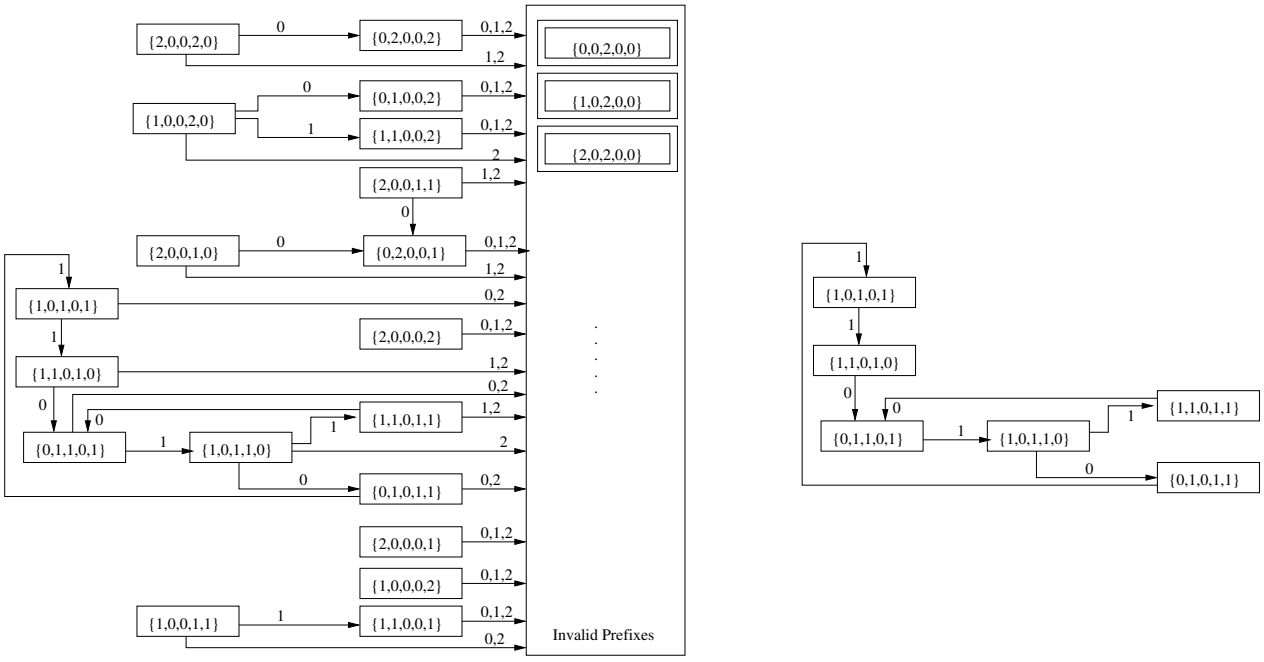
Substituting the values of $w_3, \ldots, w_6$, we get:

$$[0 \leq 2 + w_7 \leq 3],$$
$$[5 \leq 3 + w_7 \leq 9]$$

The first constraint yields $w_7 \leq 1$ and the second constraint yields $w_7 \geq 2$. Since these two constraints are not satisfiable together, it follows that P cannot be extended without violating R. □

Definition 3 (Unrealizable Prefix)**.** *An unrealizable prefix is a member of the following language:*

$$\Sigma^* - \Big(Pref(\mathcal{L}(R)) \bigcup InvPref(\mathcal{L}(R))\Big)$$



$D(A')$: DFA for $\mathcal{L}(A')$ $TS(\ D(A'))$: Transition System $\mathcal{L}(A)$

Fig. 2: DFA and corresponding Transition System for the constraints in Example 4

□

The members of Pref($\mathcal{L}(R)$) will be called *realizable* prefixes.

An on-the-fly generator of a valid word must guarantee that it does not generate an unrealizable prefix. The existence of unrealizable prefixes in RTC specifications has been observed in prior literature [10], [11], however the main challenge is in avoiding unrealizable prefixes during the generation of random patterns. We belive that at runtime it is too expensive to determine whether each choice in the sequence creates an unrealizable prefix. We propose a suitable alternative where this decision is based on an automaton that is constructed a priori and used at runtime.

A. Automata for Constructing Realizable Prefixes

The intuitive idea of the proposed construction is quite simple. As described in Section II, we construct a DFA for the language InvPref($\mathcal{L}(R)$). We then remove all accept states in the DFA along with those states from which reaching an accept state is inevitable. We prove that any random walk on the remaining DFA defines a realizable prefix.

We define Δ_{max} as follows:

$$\Delta_{max} = \max\{\Delta | \langle \Delta, \alpha^L(\Delta), \alpha^U(\Delta) \rangle \in R\}$$

We construct a DFA, $D = \langle Q, \Sigma, \delta, F \rangle$, where:

- Σ is the alphabet as defined in Section II,
- Q is the set of states. Each state, q, is represented by a sequence of length Δ_{max}, denoted as $q[1], \ldots, q[\Delta_{max}]$. Intuitively a state is represented by the pattern of events in the previous Δ_{max} cycles, with $q[i]$ denoting the number of events in the i^{th} preceding cycle.
- F is the set of final states. A state q is a final state if the sequence $q[\Delta_{max}], \ldots, q[1]$ belongs to InvPref($\mathcal{L}(R)$). To determine whether $q[\Delta_{max}], \ldots, q[1]$ belongs to $InvPref(\mathcal{L}(R))$ we need to check whether the RTC constraints are violated in this Δ_{max}-length sequence. We do not need to computer $InvPref(\mathcal{L}(R))$.
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function, defined as follows.
  - For each $q \in F$ and each $w \in \Sigma$, we have $\delta(q, w) = q$.
  - For $q \in Q - F$ and $w \in \Sigma$, we have $\delta(q, w) = q'$ iff $q'[1] = w$ and $\forall j, 2 \leq j \leq \Delta_{max}$, $q'[j] = q[j-1]$.

The initial state of the DFA will be defined later. It may be noted that each final state in F is a sink state, that is, the DFA stays in that state forever. We define a *dead state* recursively as follows.

Definition 4 (Dead State)**.** *Given the DFA,* $D = \langle Q, \Sigma, \delta, F \rangle$*, a state* $q \in Q$ *is a dead state iff:*

- $q \in F$*, or*
- $\forall w \in \Sigma$*,* $\delta(q, w)$ *is a dead state.*

□

The proposed method for random generation of event sequences satisfying the set R of RTC constraints is as follows:

Sequence Generation Method

1. Construct the DFA, $D = \langle Q, \Sigma, \delta, F \rangle$,

RTC Specifications (R)	No. of States in $D(R')$ $(\Sigma^{\Delta_{max}})$	No. of States in $TS(D(R'))$	Time (s)
$\{\langle 3,1,3\rangle, \langle 5,4,5\rangle, \langle 13,9,11\rangle\}$	3^{13}	321	4
$\{\langle 5,4,5\rangle, \langle 13,9,11\rangle\}$	5^{13}	1738	54
$\{\langle 4,2,4\rangle, \langle 8,5,7\rangle, \langle 12,9,10\rangle\}$	4^{12}	13704	68
$\{\langle 8,5,7\rangle, \langle 12,9,10\rangle\}$	7^{12}	37919	452
$\{\langle 8,2,7\rangle, \langle 12,4,9\rangle, \langle 15,6,10\rangle\}$	7^{15}	1497500	9363
$\{\langle 5,1,3\rangle, \langle 14,5,7\rangle\}$	3^{14}	14146	61
$\{\langle 5,1,4\rangle, \langle 7,3,5\rangle, \langle 11,5,7\rangle\}$	4^{11}	5113	24
$\{\langle 5,3,5\rangle, \langle 7,5,7\rangle, \langle 11,9,11\rangle\}$	5^{11}	7089	46
$\{\langle 5,1,4\rangle, \langle 7,3,5\rangle, \langle 11,5,7\rangle\}$	4^{11}	5113	24
$\{\langle 5,3,5\rangle, \langle 7,5,7\rangle, \langle 13,10,12\rangle\}$	5^{13}	37169	188

TABLE I: Representative State Spaces

as defined above.
2. Remove dead states from D to obtain a transition system, $TS(D)$.
3. Perform a random walk on $TS(D)$ to define a random event sequence.

The transition system $TS(D)$ is obtained from D by removing the dead states in D. It follows from the definition of dead states that every state of $TS(D)$ has at least one outgoing transition. Therefore a random walk in $TS(D)$ will never reach a dead end.

The random sequence is generated by traversing $TS(D)$ as follows. We randomly choose any state, q, of $TS(D)$ as the initial state. Then $q[\Delta_{max}], \ldots, q[1]$ is the initial prefix of the sequence. With every transition (q, w, q') taken in the random walk, we add w to the sequence.

Example 4. Consider the following set of RTC constraints :

$$A = \{\langle 3,0,2\rangle, \langle 5,3,4\rangle\}$$

Figure 2 shows the DFA, $D(A')$, and the transition system $TS(D(A'))$ obtained by removing the dead states from $D(A')$. To generate an event sequence we start from any state, and peform a random infinite walk in $TS(D(A'))$. For example, $01101(110)^{\omega}$ is an event sequence generated by an infinite walk in the transition system $TS(D(A'))$. □

In the remainder of this section, we prove the soundness and completeness of our generation approach.

Theorem 2. [Soundness]: *The sequences generated by the proposed approach satisfy the given RTC constraints.*
Proof: We show that every infinite walk, $\pi = s_1, s_2, \ldots$, in $TS(D(R'))$ defines a valid event sequence with respect to given RTC specification R.

Let the event sequence defined by π be $w_1, w_2, \ldots$. By the definition of $TS(D(R'))$, s_i is represented by the Δ_{max}-length sequence, $w_{\Delta_{max}+i-1}, \ldots, w_i$. For example, s_1 is represented by the sequence $w_{\Delta_{max}}, \ldots, w_1$, s_2 is represented by the sequence $w_{\Delta_{max}+1}, \ldots, w_2$, and so on.

The event sequence defined by π is valid unless it contains some Δ_{max}-length sequence which refutes a RTC constraint in R. Suppose there exists such an invalid sequence, $w_k, \ldots, w_{\Delta_{max}+k-1}$. But that implies that the state s_k is a dead state. This is a contradiction since $TS(D(R'))$ has no dead states. □

Theorem 3. [Completeness]: *Every infinite sequence that satisfies the given RTC specification R can be generated by some walk on $TS(D(R'))$.*
Proof: Suppose $w_1, w_2, \ldots$ is an infinite sequence satisfying R. We define a path $\pi = s_1, s_2, \ldots$, such that s_i is the state of $D(R')$ represented by the Δ_{max}-length sequence, $w_{\Delta_{max}+i-1}, \ldots, w_i$. The only possible reason why the path may not exist in $TS(D(R'))$ is that π contains some dead state s_k. Since the sequence contains no refutation of R, s_k cannot be an invalid sequence. Therefore it must be the case that s_k is a *forbidden state*, that is, all paths from s_k lead to invalid states. But that would mean that some invalid sequence will also exist in the event sequence $w_1, w_2, \ldots$, which is a contradiction. Therefore every infinite sequence that satisfies R can be generated by a walk over $TS(D(R'))$. □

It is important to note that Theorem 3 refers to infinite sequences only. There are finite sequences not generated by $TS(D(R'))$ that do not refute any member of R. These finite

sequences necessarily end in forbidden states, and therefore, avoiding such sequences is precisely the purpose of this methodology.

IV. Conclusion

The appreciation of RTC as a specification language motivated us to study the problems of acceptance and enumeration from a language theoretic perspective. While we have presented the basic findings of our study, there is scope of substantial performance gains through the use of more sophisticated symbolic approaches. These will no doubt determine the scalabity of meaningful usage of RTC as a specification language.

Nevertheless, we developed a code implementing the construction of the proposed automata to study the nature of growth in the state space. Our findings are shown in Table I. The main intention of showing these results is to demonstrate that with as few as three constraints, the fraction of states that are not dead states is quite small. Therefore pre-computing $TS(D(R'))$ and using it at runtime to generate random event sequences is feasible from a practical point of view.

It is also observed that in general the number of states in $TS(D(R'))$ increases with Δ_{max}, but it depends on the pruning imposed due to the lower bounds as well. The runtimes to compute $TS(D(R'))$ was obtained on a Intel Core i7 with 8GB of memory. Once the transition systems is pre-computed the time to generate a random sequence will be negligible because it can be done through a random walk.

Real world problems involving large values of Δ_{max} are rare. In such cases, the value domain can be scaled down by dividing all numbers with a common denominator and neglecting the remainders in a conservative way such that satisfying the modified constraints guarantee that the original constraints are satisfied. Explaining the details of this approximation is beyond the scope of this paper.

Since the focus of this paper is on using the generators and acceptors on discrete event simuation platforms, we have studied the problem in a discrete time setting. Solving these problems in the dense time setting remains an interesting open problem.

References

[1] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *ISCAS*, 2000, pp. 101–104.

[2] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse, "System architecture evaluation using modular performance analysis: a case study," *Int. J. Softw. Tools Technol. Transf.*, vol. 8, no. 6, pp. 649–667, Oct. 2006.

[3] D. B. Chokshi and P. Bhaduri, "Performance analysis of flexray-based systems using real-time calculus, revisited," in *Proc. of the 2010 ACM Symposium on Applied Computing*, ser. SAC '10. New York, NY, USA: ACM, 2010, pp. 351–356.

[4] S. Chakraborty, S. Knzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *In DATE*, 2003.

[5] V. Pollex, H. Lipskoch, F. Slomka, and S. Kollmann, "Runtime improved computation of path latencies with the real-time calculus," in *Proc. of the 1st International Workshop on Worst-Case Traversal Time*, ser. WCTT '11. New York, NY, USA: ACM, 2011, pp. 58–65.

[6] O. Sokolsky and A. Chernoguzov, "Performance analysis of aadl models using real-time calculus." in *Monterey Workshop*, ser. LNCS, C. Choppy and O. Sokolsky, Eds., vol. 6028. Springer, 2008, pp. 227–249.

[7] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: A hybrid methodology for the performance analysis of embedded real-time systems," *Design Automation for Embedded Systems*, vol. 14, no. 3, pp. 193–227, 2010.

[8] S. Chakraborty, L. T. X. Phan, and P. S. Thiagarajan, "Event count automata: A state-based model for stream processing systems," in *IN RTSS*, 2005.

[9] K. Altisen and M. Moy, "Connecting real-time calculus to the synchronous programming language lustre." Verimag Research Report, Tech. Rep. TR-2009-14, 2009.

[10] M. Moy and K. Altisen, "Arrival curves for real-time calculus: The causality problem and its solutions." in *TACAS*, ser. LNCS, J. Esparza and R. Majumdar, Eds., vol. 6015. Springer, 2010, pp. 358–372.

[11] ——, "Arrival curves for real-time calculus: the causality problem and its solutions," Verimag Research Report, Tech. Rep. TR-2009-15, 2009.

[12] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems," in *Proc. of the seventh ACM international conference on Embedded software*, ser. EMSOFT '09. New York, NY, USA: ACM, 2009, pp. 107–116.

[13] M. O. Rabin, *Automata on infinite objects and Church's problem*, ser. Regional conference series in mathematics. Providence, R.I. Published for the Conference Board of the Mathematical Sciences by the American Mathematical Society, 1972, expository lectures from the CBMS regional conference held at Morehouse College, Atlanta, Georgia, September 8-12, 1969.

[14] J. R. Büchi, *On a Decision Method in Restricted Second Order Arithmetic*. Elsevier, 1966, vol. 44, pp. 1–11.

[15] G. Garay, J. Ortega, and V. Alarcon-Aquino, "Comparing real-time calculus with the existing analytical approaches for the performance evaluation of network interfaces," in *Electrical Communications and Computers (CONIELECOMP), 2011 21st International Conference on*, 2011, pp. 119–124.

[16] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems," in *Proc. of the seventh ACM international conference on Embedded software*, ser. EMSOFT '09. New York, NY, USA: ACM, 2009, pp. 107–116.

[17] K. Altisen and M. Moy, "Causality closure for a new class of curves in real-time calculus," in *Proc. of the 1st International Workshop on Worst-Case Traversal Time*, ser. WCTT '11. New York, NY, USA: ACM, 2011, pp. 3–10.

[18] K. Altisen, Y. Liu, and M. Moy, "Performance evaluation of components using a granularity-based interface between real-time calculus and timed automata," in *QAPL*, 2010, pp. 16–33.

[19] "Inchron." [Online]. Available: http://www.inchron.com/