Analytic real-time analysis and timed automata

A hybrid methodology for the performance analysis of embedded real-time systems

Journal Article

Author(s):
Lampka, Kai; Perathoner, Simon; Thiele, Lothar

Publication date:
2010-09

Permanent link:
https://doi.org/10.3929/ethz-b-000027956

Originally published in:
Design automation for embedded systems 14(3), https://doi.org/10.1007/s10617-010-9055-1

Des Autom Embed Syst (2010) 14: 193–227
DOI 10.1007/s10617-010-9055-1

Analytic real-time analysis and timed automata: a hybrid methodology for the performance analysis of embedded real-time systems

Kai Lampka · Simon Perathoner · Lothar Thiele

Received: 15 January 2010 / Accepted: 17 May 2010 / Published online: 16 June 2010

Abstract This paper presents a compositional and hybrid approach for the performance analysis of distributed real-time systems. The developed methodology abstracts system components by either flow-oriented and purely analytic descriptions or by state-based models in the form of timed automata. The interaction among the heterogeneous components is modeled by streams of discrete events. In total this yields a hybrid framework for the compositional analysis of embedded systems. It supplements contemporary techniques for the following reasons: (a) state space explosion as intrinsic to formal verification is limited to the level of isolated components; (b) computed performance metrics such as buffer sizes, delays and utilization rates are not overly pessimistic, because coarse-grained analytic models are used only for components that conform to the stateless model of computation. For demonstrating the usefulness of the presented ideas, a corresponding tool-chain has been implemented. It is used to investigate the performance of a two-staged computing system, where one stage exhibits state-dependent behavior that is only coarsely coverable by a purely analytic and stateless component abstraction. Finally, experiments are performed to ascertain the scalability and the accuracy of the proposed approach.

Keywords Performance analysis · Timed automata · Real-time calculus · Hard real-time systems

1 Introduction

The designers of real-time embedded systems need to verify the correctness of their designs already in early design stages. Due to the increasing complexity of modern system architectures, guaranteeing correct system behavior has become one of the most challenging steps

K. Lampka · S. Perathoner (✉) · L. Thiele
Computer Engineering and Networks Laboratory, ETH Zurich, Zurich, Switzerland
e-mail: perathoner@tik.ee.ethz.ch

K. Lampka
e-mail: lampka@tik.ee.ethz.ch

L. Thiele
e-mail: thiele@tik.ee.ethz.ch

in the design process. Empirical methods such as testing or simulation are often inadequate for this task because they are not exhaustive. In order to provide hard guarantees for the system behavior as required for many application domains, formal analysis methods need to be applied.

Timed automata [2] are a well accepted formalism for analyzing real-time systems, see e.g. [17]. However, the finite state/transition system to be derived from some high-level model tends to grow exponentially with the number of clocks and clock constants. Therefore, the detailed analysis of a complex system may be hampered in practice, if not impossible at all. In contrast, purely analytic (or stateless) methods such as provided by the Real-Time Calculus [21, 23], SymTA/S [11] or MAST [9] solely depend on solutions of closed form expressions, yielding a very good scalability with the size of systems to be analyzed. But, this advantage leads to serious drawbacks: (a) analytic methods are limited to the computation of a few specific system measures and (b) each method is restricted to a specific model to which the system specification under analysis must be translated, which in general may lead to overly conservative analysis results. To overcome these shortcomings, this paper aims to combine purely analytic and state-based performance analysis methods. Employing state-based evaluation approaches only for those system components, where stateless analysis delivers overly pessimistic results, will maintain scalability.

In the present work we have chosen to combine TA (Timed Automata) and RTC (Real-Time Calculus), as the former is widespread for verification of real-time systems and RTC is an advanced analytic performance analysis approach, see [6, 21, 23]. However, we would like to point out that the presented method is not limited to RTC. The coupling of TA with other analytic performance evaluation frameworks such as any method from classical real-time analysis or SymTA/S can be reduced to a special case of what is discussed here.

Coupling the Modular Performance Analysis framework (MPA) [23] which is based on RTC [21] and Uppaal [22] for the joint analysis of embedded real-time systems is far from trivial, since (a) the RTC lacks a concrete execution semantics unlike TA, (b) TA can not be verified by evaluating closed form expressions, nor can one in general derive an analytic description from them and (c) RTC and TA not even share the same time domain. TA operate on the conventional time-line, whereas the RTC operates on stream abstractions that are defined on time intervals. To overcome this obstacles this paper provides the following contributions:

- A pattern is described allowing to convert abstract stream models such as PJD (periodic with jitter) or arrival curves to a network of co-operating TA (Sect. 4.2).
- The pattern can be employed for a convex/concave approximation of arrival curves which is the key factor for ensuring simple and scalable TA models.
- Tightness and correctness of the transformation is proven, i.e., the TA solely generate event traces complying with the abstract stream model, and they do this for all conforming event traces (Sect. 4.2.3).
- A pattern is described to automatically derive abstract stream models (such as PJD or arrival curves) from a TA-based system model (Sect. 4.3).
- Finally the paper presents an implementation, analyzes an exemplarily chosen system, and investigates the scalability and the accuracy of the proposed methodology (Sect. 5).

Note that this article builds on the work presented in [15]. In the present version we add several explanation and proves, generalize the pattern for the conversion of arrival curves to TA, formalize the derivation of arrival curves from TA systems, and extend the experimental results by evaluating the scalability and the accuracy of the proposed approach.

2 Related work

There are several other approaches known which also tackle the combination of RTC-based analytic and state-based models for system-wide performance analysis. The authors of [20] bridge the gap between a state-based methodology and the RTC-method as discussed in Sect. 3. However, contrary to this paper the work of [20] is based on event count automata (ECA) [7]. With ECA the user must specify the minimum and maximum number of event arrivals taking place while the ECA resides in the respective location. For translating an RTC-based abstract stream representation into an ECA the authors use the principle of a ring buffer. Each counter represents the number of events associated with the respective number of unit intervals. When it comes to the interfacing of ECA with RTC, i.e., one needs to derive abstract stream representations as used in RTC-curves from ECA specifications, the authors of [20] suggest the usage of observer ECA. They use binary search for extracting the maximum and minimum number of events seen in a window size Δ via reachability analysis. In [19] it is shown how the above approach can be employed within a hybrid framework, allowing to obtain key performance metrics of embedded systems by combining RTC and ECA-based analysis. However, our usage of TA appears to be more beneficial, since contrary to ECA they have an explicit notion of time, whereas ECA advance in a lock-step fashion. In addition, in our work we solely require one observer automaton for a complete staircase function defined over all time intervals Δ rather than one observer per discrete window size Δ.

The authors of [8] present an approach, where a system to be analyzed is mapped to a process network which is analyzed via a compositional response time analysis [11]. The resulting periodic event stream models and the computed response times serve as parameters for pre-defined TA. The high-level descriptions of system properties to be checked are also transformed into TA. Finally, the use of standard model checking procedures allows to check, whether the system model fulfils the desired properties or not. The approach differs from the new results in this paper as the system is not decomposed into components which exhibit substantial state-dependent behavior and those which can be analyzed using analytic approaches. Instead, state-based behavior is not explicitly taken into account.

The authors of [13] also address the combination of RTC-based components and TA. For including the abstract stream representation used in RTC into TA-based system models one operates on an array of clocks. Each clock is associated with the number of events produced so far, as well as with a minimal and maximal number of events to be generated within the respective time interval length. For deriving RTC-based stream representations from the combined model, the authors suggest the usage of observer automata and binary search on the maximal and minimal number of events that appear within any time interval of length Δ, which is in fact similar to the idea of [20]. As one operates on a finite set of time-interval lengths Δ only, it is not clear when to stop with the translation of an abstract event stream representation into a TA and vice versa. The use of observer automata that investigate single time-interval lengths only implies that one either needs one observer automaton with its local clock for each interval length, or one must execute a full state space exploration for each of the interval lengths. Also on the side of the event generating automaton, the number of clocks may be prohibitively large because one basically needs one clock per upper and lower bound for the number of events seen on the stream within the resp. time interval Δ. The approach described in this paper attempts to overcome these limitations by using a compositional leaky bucket representation of event streams.

3 Background theory

In this section we describe the theoretical notions that form the basis of the approach developed in this paper. We start with some basic terminology followed by a brief introduction of the Real-Time Calculus and Timed Automata.

3.1 Terminology

In the following we define some basic terms used in this paper.

- A *timed action* is a pair (a, t) where a is some action such as the occurrence of an event and $t \in \mathbb{R}^{\geq 0}$ is a time stamp.
- A *timed trace* $\tau := (a_1, t_1); (a_2, t_2); \ldots$ is an (infinite) sequence of timed actions ordered by non-decreasing time stamps, i.e., $t_i \leq t_{i+1}$ for $i \geq 1$.
- A timed event trace r, shortly denoted as *event trace*, is a timed trace of the form $r := (e, t_1); (e, t_2); \ldots$ where e is a recurring event in a stream.

3.2 Real-time calculus

Real-Time Calculus (RTC) [6, 21] is a compositional framework for performance analysis that extends the classical Network Calculus, see e.g. [5], towards analyzing distributed (hard) real-time systems. RTC permits to analyze the flow of event streams through a network of processing and communication resources. In the following, we will briefly reproduce the basic concepts of RTC that will be used in this paper.

Definition 1 (Arrival function) Each event trace r can be characterized by an arrival function $r : \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \rightarrow \mathbb{N}^{\geq 0}$ where $r(s, t)$ for $0 \leq s \leq t$ denotes the number of events that arrive in the time interval $[s, t)$ (including s but not t), with $r(s, s) = 0$.

For the sake of simplicity, in the following the notation r is used for both the event trace and the corresponding arrival function; it will be clear from the context to which of the two we refer.

Contrary to most other analysis techniques, in RTC event streams are not represented in the time domain, but in the time-interval domain. In particular, when modeling event streams one can abstract from concrete event traces and describe all traces of an event stream by means of a tuple of *arrival curves* $\alpha(\Delta) := [\alpha^u(\Delta), \alpha^l(\Delta)]$.

Definition 2 (Arrival curves) Let $r(s, t)$ be the arrival function of an event trace r as defined above. Then r, α^u, α^l are related to each other by the inequality

$$\alpha^l(t - s) \leq r(s, t) \leq \alpha^u(t - s) \quad \forall s,\ t \in \mathbb{R}^{\geq 0},\ s \leq t \tag{1}$$

with $\alpha^u(0) = \alpha^l(0) = 0$. If the above inequality holds for an event trace r, we say that r *conforms to* α, denoted as $r \models \alpha$.

Informally, an upper arrival curve $\alpha^u : \mathbb{R}^{\geq 0} \rightarrow \mathbb{N}^{\geq 0}$ provides an upper bound on the number of events seen on the event stream in *any* time interval of length Δ. Analogously, a lower arrival curve $\alpha^l : \mathbb{R}^{\geq 0} \rightarrow \mathbb{N}^{\geq 0}$ denotes a lower bound on the number of events seen in any time interval of length Δ.

Definition 3 (Set of conforming event traces) Let α be a tuple of arrival curves as defined above. The set of all event traces that conform to α is defined by

$$R^{\alpha} := \{r \in R : r \models \alpha\}, \tag{2}$$

where R denotes the set of all event traces.

The conformance of an event trace r to an upper (lower) arrival curve α^u (α^l), as well as the sets R^{α^u}, R^{α^l} are defined accordingly.

Similarly, for representing the availability of processing or communication resources, RTC employs so-called *service curves*. A tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper and lower service curves specifies an upper and a lower bound on the service available from a resource in any time interval of length Δ.

Definition 4 (Service curves) Let $c(s,t)$ be the number of service units (e.g. processing or communication cycles) available from a resource over the time interval $[s,t)$. Then c, β^u, β^l are related to each other by the inequality

$$\beta^l(t-s) \le c(s,t) \le \beta^u(t-s) \quad \forall s,\ t \in \mathbb{R}^{\ge 0},\ s \le t \tag{3}$$

with $\beta^u(0) = \beta^l(0) = 0$.

In the following we will use the term *RTC-conforming curves* to refer generally to both, arrival and service curves.

In RTC arrival and service curves provide the inputs to a single analysis component. For computing the corresponding bounds α' for the outgoing event stream and β' for the remaining resources one commonly applies operators of min-plus and max-plus algebra, see [21, 23]. Overall, this component-based analysis methodology allows to obtain (hard) bounds on job delays, buffer sizes and utilization of hardware units, either for a single component or for complex systems.

3.3 Timed automata

In this paper we use timed safety automata as found in the model checker Uppaal [3]. In the following we briefly re-capitulate only some relevant aspects of TA. For a detailed introduction please refer to [4].

An extended time safety automaton is a graph, consisting of locations, directed edges between them, non-negative clocks and a set of local (integer) variables. Conditions imposed on clocks (time constraints) and variables steer the execution of edges in a TA. An example of TA is shown Fig. 1 which represents the model of a traffic light. Each edge execution in a TA establishes a state-to-state transition in a corresponding state graph. A TA can be expanded into its state graph by iteratively considering all executions until a fixed point is reached. The state graph captures all the timed behavior of a TA and is used for the formal verification of system properties. In a TA conditions related to edges are denoted as guards and conditions related to locations are denoted as location invariants. The execution of an edge can only take place if both the guard of the edge and the invariant of the target location evaluate to true. The execution of an edge is typically followed by clock and variable updates. Note that the invariant of the target location must be satisfied *after* the clock and variable updates of the incoming edge have taken place. A location of a TA can be labeled

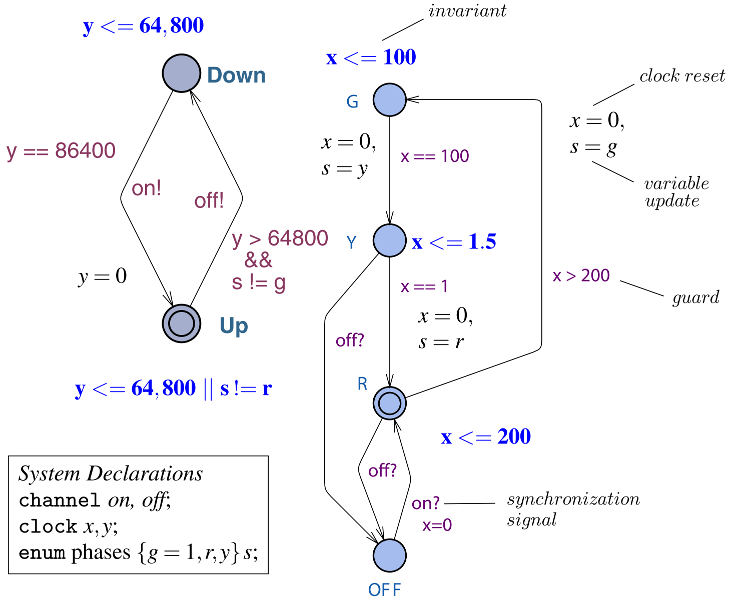


Fig. 1 Timed Automata modelling a traffic light

as urgent. In urgent locations no time can elapse, meaning that once an urgent location is entered, any outgoing edge has to be executed in zero time.

A key feature of the TA modeling approach is modularity. Individual TA models can be composed for the representation of complex systems. The interaction of the different TA modules is based on shared global variables and synchronized edge executions. For instance, in the example of Fig. 1 the system model consists of two TA components. The TA on the right represents a traffic light, whereas the TA on the left represents a controller which switches the light off during some time interval. The two TA make use of a joint (enumeration) variable denoted s which can have the values r, y or g. Moreover, the two TA interact via synchronized execution of their edges labeled with *on* and *off*. Edges of different TA with the same synchronization label, also denoted as channel, have to be executed contemporaneously in an atomic manner. By following Uppaal's nomenclature we will also speak of sender and receivers when referring to the synchronization of different TA. In Uppaal we distinguish the following types of rendez-vous mechanisms:

- *Binary synchronization*
  A sending and a receiving TA synchronize on the joint execution of two dedicated edges, one in the sending TA, whose edge is labeled by a `channel id` and an exclamation mark, and one in the receiving TA, whose edge is labeled by the same `channel id`, but extended with a question mark (see, e.g., the *event*! and *event*?-labeled edges in the TA of Fig. 4(B) and 4(C)).
- *Broadcast synchronization*
  One sender synchronizes with up to n receivers. This refers to the situation where one sending TA executes a sending edge, which can be understood as the emission of a signal and where between 0 and n receiving TA execute a receiving edge, which can be interpreted as the instantaneous reception of this broadcast signal. It is important to note that although the sender can synchronize with any number of receivers between 0 and n, par-

ticipation in the broadcast synchronization is not facultative for receivers. In particular, all TA containing a receiving edge have to execute this edge if at the time of the synchronization it is enabled, that is, if the edge can be executed. (see, e.g., the *event*! and *event*?-labeled edges in the TA of Fig. 5(B–D)).

Note that for the sake of better readability, the TA depicted in this paper are not always syntactically correct. In particular, the `if`-statement that we employ has to be implemented in Uppaal with the `?`-operator of ANSI C. For the max- and min-operator one needs to define individual functions as part of Uppaal's system declaration. Note further that for addressing the evaluation of a clock x or a counter b at some time t we will use the notation $x(t)$ or $b(t)$. In cases where the concrete point in time t is clear from the context, we will also use the clock identifier instead.

4 The approach

In the following we will develop a scheme for interfacing RTC-conforming curves with TA-based system descriptions, as illustrated in Fig. 2. The major result is the transformation of event stream specifications in the form of arrival curves (defined in the time-interval domain) to sets of event traces specified by TA (defined in the time domain) and vice versa. The developed strategy consists of two independent parts which will be discussed separately:

(i) In Sect. 4.2 we show how to implement RTC-conforming **input** curves by a network of TA denoted as *input generator*. This transformation corresponds to the interface denoted with 'RTC→TA' in Fig. 2.

(ii) In Sect. 4.3 we discuss how to derive **output** curves from TA-based system specifications. This transformation corresponds to the interface denoted with 'TA→RTC' in Fig. 2.

These two transformations enable a hybrid approach to performance analysis of distributed systems where the individual components are either abstracted on the basis of the RTC or modeled by means of TA.

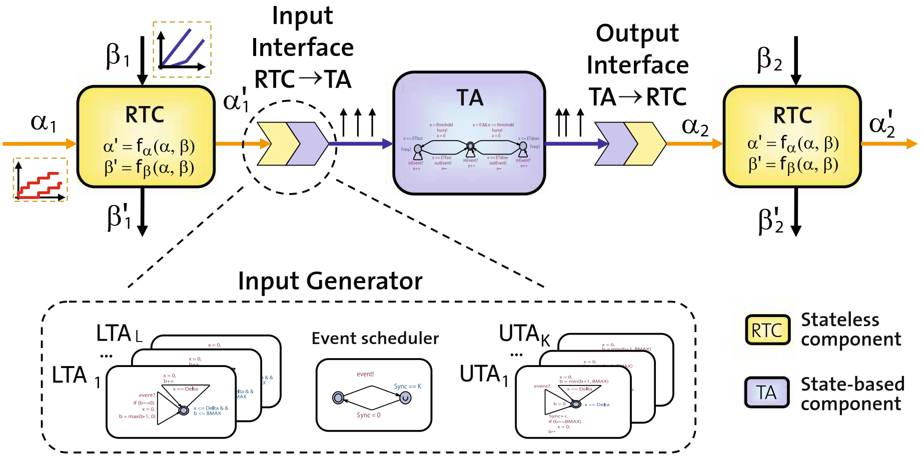
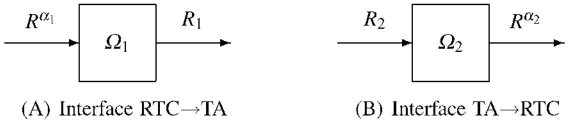


Fig. 2 Overview of hybrid analysis method

Fig. 3 Transformations among sets of event traces for the two described interfaces



In this paper we will focus on the conversion between arrival curves and sets of event traces. Note, however, that the presented approach is not limited to arrival curves, but can also be applied to service curves. The only limitation to consider is that with TA, in particular with Uppaal, one can only make use of discrete variables, rather than continuous ones. Hence, we can consider only systems with discrete numbers of events or resource units, where on the level of RTC this refers to staircase functions.

The major complexity faced when developing the mentioned interfaces is the fact that the bounding functions $\alpha = [\alpha^u, \alpha^l]$ in time-interval domain implicitly define a possibly infinite set of event traces R^α.

4.1 Requirements

The main requirement for the discussed hybrid approach to performance analysis of embedded real-time systems is that the described input and output interfaces are *correct*, in the sense that they do not harm the safety of the analysis. In particular, in order to guarantee conservative analysis results, whenever transforming the abstraction of an event stream to another representation, we have to make sure that the conversion does not suppress any event trace of the stream.

Consider first a generic input interface (RTC→TA) in which a tuple of arrival curves α_1 is converted to an input generator, that is, to a network of TA. Fig. 3(A) illustrates the corresponding transformation among sets of event traces where R^{α_1} denotes the set of event traces that conform to α_1 and R_1 represents the set of event traces specified by the input generator. We denote the set R_1 also as the set of event traces *producible* by the input generator. We say that the input interface is correct iff $R_1 \supseteq R^{\alpha_1}$. More precisely, for the input interface we require that

$$r \models \alpha_1 \Rightarrow r \in R_1 \quad \forall r \in R. \tag{4}$$

Now consider a generic output interface (TA→RTC) in which the output of a TA subsystem is translated to a tuple of arrival curves α_2. Fig. 3(B) shows the corresponding transformation where R_2 denotes the set of traces producible by the TA subsystem and R^{α_2} represents the set of traces that conform to α_2. We say that the output interface is correct iff $R^{\alpha_2} \supseteq R_2$. More precisely, for the output interface we require that

$$r \in R_2 \Rightarrow r \models \alpha_2 \quad \forall r \in R. \tag{5}$$

Note that the above properties guarantee a correct analysis, but do not exclude pessimistic analysis results. In particular, the accuracy of the performance analysis can degrade in cases in which $R_1 \supset R^{\alpha_1}$ holds for an input interface or $R^{\alpha_2} \supset R_2$ holds for an output interface. For instance, this corresponds to the case when the model of a system component is fed with more event traces than originally specified for the ingoing event stream. Such pessimistic transformations are avoided if $R_1 \subseteq R^{\alpha_1}$ holds for all input interfaces and $R^{\alpha_2} \subseteq R_2$ holds for all output interfaces. Depending on the individual case, this requirement can, however, be sacrificed, e.g., to improve the efficiency of the analysis.

4.2 Input interface

For interfacing RTC-conforming curves to TA the approach has to translate time-interval-based functions into TA-based representations of possibly infinitely many timed traces. The main idea for achieving this is based on the observation that the interval-based arrival curves $\alpha = [\alpha^u, \alpha^l]$ can be modeled by sets of individual staircase functions combined by the (nested) application of minimum or maximum. Each of the involved staircase functions $\alpha_i^{\{u,l\}}$ is guarded by its own TA, where an automaton LTA$_i$ guards *lower* curve α_i^l and an automaton UTA$_i$ guards *upper* curve α_i^u. The network of co-operating UTA and LTA emits a dedicated, instantaneous signal *event* to the environment, where this signal can be used for stimulating a user-defined TA-based model description which represents some component of the system under analysis. Emission of the *event*-signal has to be done in such a way that one is capable of producing each trace $r = ((event, t_0); \ldots\ (event, t_n)\ldots)$ of *event*-signals such that $r \models \alpha$. For keeping the discussion as simple as possible, we will start now with the most simple case, where $\alpha^{\{u,l\}}$ are defined by a single staircase-function each.

4.2.1 Linear input pattern

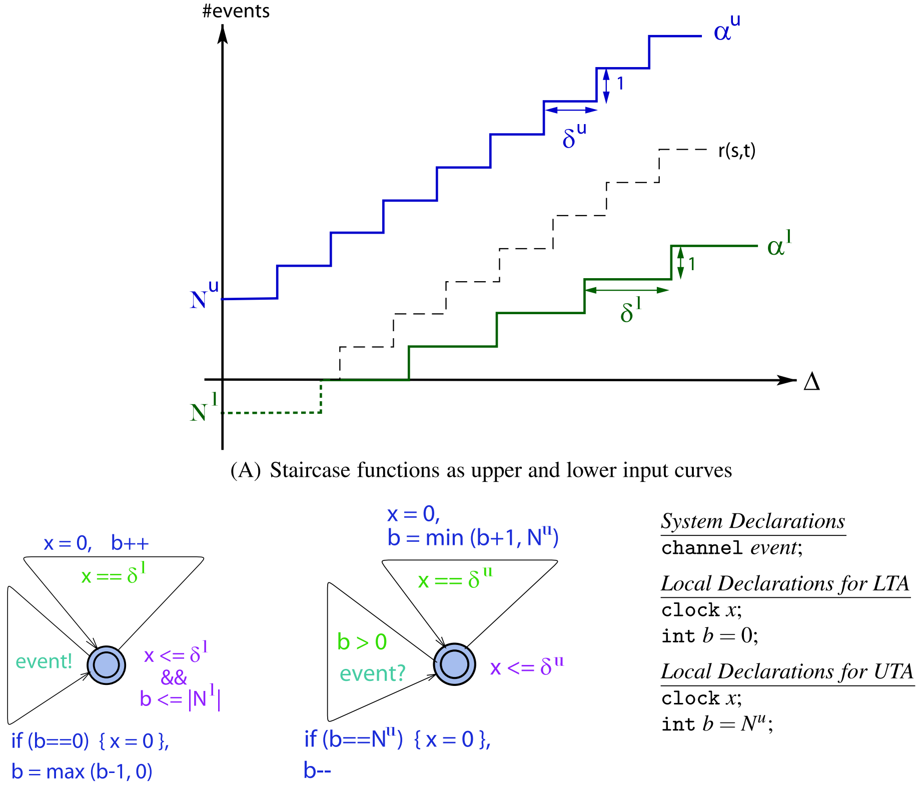
We define an upper or lower staircase function as follows:

$$\alpha^{\{u,l\}}(\Delta) := N^{\{u,l\}} + \left\lfloor \frac{\Delta}{\delta^{\{u,l\}}} \right\rfloor \tag{6}$$

where in our approach each curve is guarded by its own timed automaton denoted UTA, LTA respectively. Binary synchronization enforces UTA and LTA to produce only those traces which contain at least $\alpha^l(\Delta)$ and at most $\alpha^u(\Delta)$ *event* signals for any interval of length $\Delta \in \mathbb{R}^{\geq 0}$.

For exemplification please refer to the curves depicted in Fig. 4. The parameter N^u can be understood as burst capacity, which describes the number of events producible in zero time according to curve α^u. The parameter $\delta^{\{u,l\}}$ specifies the minimum/maximum distance of two successive events with respect to curve $\alpha^{\{u,l\}}$. The absolute values of parameter N^l of the lower curve can be understood as the fictitious numbers of δ^l delays before event emission has to be enforced every δ^l time unit. The above constants provide the values for the free parameters in UTA and LTA.

(A) Implementation The actual implementation of UTA and LTA is shown in Fig. 4(B) and 4(C). Each of them employs its own clock x, counter b, and its constants $N^{\{u,l\}}$ and $\delta^{\{u,l\}}$ (Fig. 4(D)). The edge-guards steering the execution of the different edges, clock resets, variable updates and signal sending and reception are specified next to the respective edge. Location invariants are stated next to the affected location. UTA and LTA cooperate by synchronizing on sending and receiving signal *event*, which enforces the joint execution of the *event*! and the *event*?-labeled edges in the two TA. The non-deterministic emission of events (sending and receiving of the *event*-signal by LTA and UTA) is possible as long as for the UTA $b > 0$ holds. On the other hand emission of the signal *event* has to take place once the local variable b of LTA reaches its local threshold $|N^l|$, which is enforced by the location invariant $x <= \delta^l\ \wedge\ b <= |N^l|$. It is also important to note that for $b = 0$ and the production of an event LTA resets its local clock x, whereas UTA does so once $b = N^u$ holds and the signal *event* is sent.



(A) Staircase functions as upper and lower input curves

System Declarations
`channel` *event*;

Local Declarations for LTA
`clock` x;
`int` $b = 0$;

Local Declarations for UTA
`clock` x;
`int` $b = N^u$;

(B) LTA for guarding α^l (C) UTA for guarding α^u (D) Declarations

Fig. 4 Linear pattern: input curves and their TA-based implementation

(B) Correctness and tightness of interface In the following, we briefly sketch a proof for the correctness and tightness of the described interface. A more detailed proof that considers more complex arrival curves follows in Sect. 4.2.3.

Theorem 1 *Let R^α be the set of event traces that conform to $\alpha = [\alpha^u, \alpha^l]$ defined in* (6)*. Let R^{TA} denote the set of event traces producible by the input generator of Fig.* 4*. Then, $R^\alpha = R^{TA}$.*

Sketch of Proof

- $R^\alpha \supseteq R^{TA}$:
  This will be justified separately for α^u and α^l.

  1. UTA enforces that the input generator can only produce traces with at most $N^u + \lfloor \frac{\Delta}{\delta^u} \rfloor$ timed *event*-actions seen on any interval of length Δ. This is because event emission is blocked once the counter b of UTA equals 0 and because the local clock x of UTA is reset at event emissions for which $b = N^u$ holds.
  2. The invariant defined on the initial location of LTA enforces that after $(N^l + 1) \cdot \delta^l$ time units an event is emitted and from then on at least every δ^l time unit. Therefore, every

trace produced by the input generator contains at least $\alpha^l(\Delta)$ timed *event*-actions for any interval Δ.

- $R^\alpha \subseteq R^{TA}$:
  We have to show that each event trace r such that $r \models \alpha$ is producible by the input generator. This can be shown by contradiction. Consider the upper bound α^u. Let us assume that there exists an event trace r with $r \models \alpha^u$ that is not producible by the input generator. It follows that there is a time instant t at which UTA is blocked, but r contains a timed action $(event, t)$. However, by reasoning about the prefix of any trace possibly produced by the input generator up to time t, it can easily be shown that an additional event at t would violate α^u, which contradicts the assumption $r \models \alpha^u$. Hence, such a trace r does not exist. For the lower bound α^l the reasoning is analogous. □

4.2.2 *Convex and concave input pattern*

Now we extend the discussion to cases where the input functions α^l or α^u are modeled as the maximum or minimum of a set of staircase functions:

$$\alpha^u(\Delta) := \min_i(\alpha_i^u(\Delta)); \qquad \alpha^l(\Delta) := \max_i(0, \alpha_i^l(\Delta)) \tag{7}$$

where $\alpha_i^{\{u,l\}}(\Delta)$ is defined as stated in (6) but now with their individual pairs of parameters $N_i^{\{u,l\}}, \delta_i^{\{u,l\}}$. For exemplification one may refer to the curve(s) depicted in Fig. 5(A). In accordance with (7), as well as with the scenario of Fig. 5 it is required that for all $i < j$ holds:

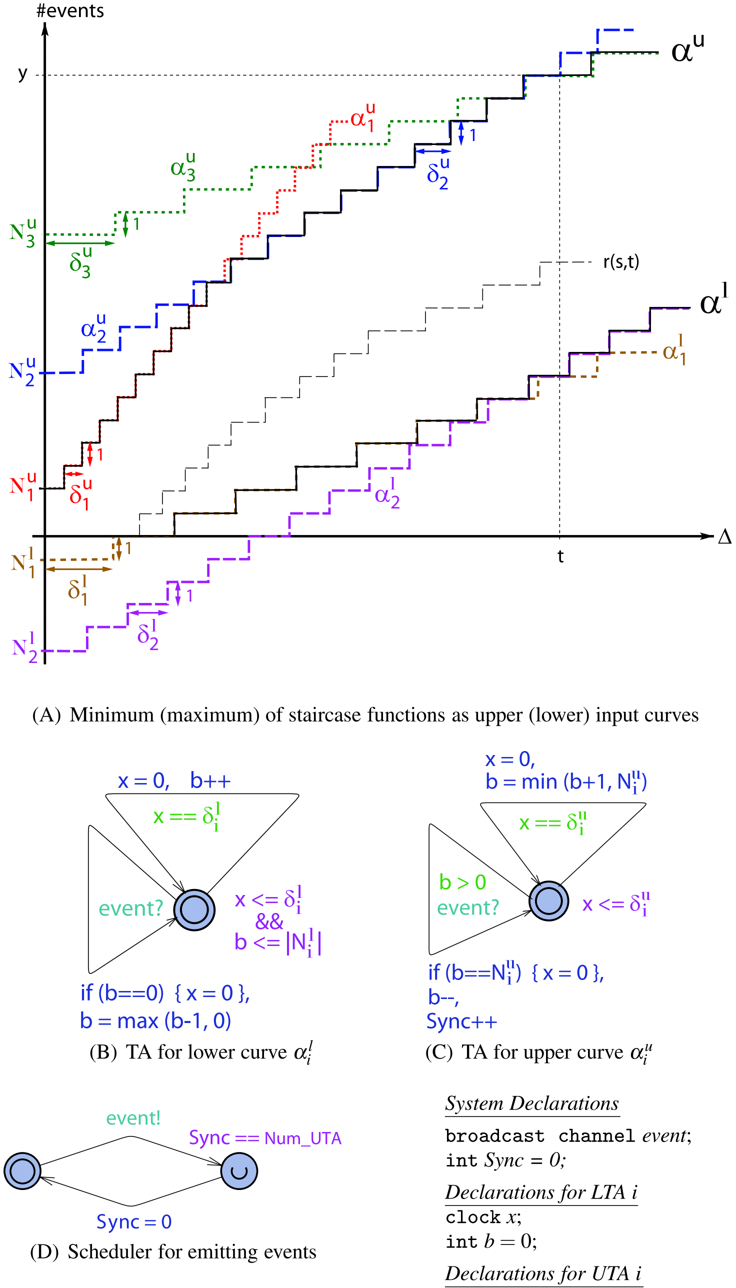
$$|N_i^{\{u,l\}}| < |N_j^{\{u,l\}}|; \qquad \delta_i^{\{l,u\}} > 0; \qquad \delta_i^l > \delta_j^l \quad \text{and} \quad \delta_i^u < \delta_j^u. \tag{8}$$

In the following we informally denote arrival curves $\alpha^{\{u,l\}}$ that fulfill (7) and (8) as *concave* and *convex*, respectively.

(A) Basic idea of the approach The bound imposed by curve $\alpha_i^{\{u,l\}}$ will be guarded by UTA and LTA i, respectively. Cooperation among the UTA and LTA has to be organized in such a way, that it complies with the minimum and maximum-operation as employed in (7). Due to the usage of minimum and maximum the following conditions apply:

1. Minimum-condition: The input generator **may** emit timed action $(event, t) \iff b_i(t) > 0 \; \forall b_i \in \mathscr{B}^u$, where $\mathscr{B}^u$ is the set of the UTA-local b-variables such that $i \in \{1, K\}$ with K as the number of UTA.
2. Maximum-condition: The input generator **has to** emit timed action $(event, t) \iff \exists b_i \in \mathscr{B}^l \wedge \exists x_i \in \mathscr{C}^l$ such that $b_i(t) = |N_i^l| \wedge x_i(t) = \delta_i^l$, where $\mathscr{B}^l$ is the set of the LTA-local b-variables, $\mathscr{C}^l$ is the set of their local clocks and δ_i^l the period for incrementing x_i, with $i \in \{1, L\}$ where L is the number of LTA.

(B) Implementation The implementation is shown in Fig. 5. To be as generic as possible we make use of Uppaal's concept of templates, s.t. clock x, constants $N_i^{\{u,l\}}, \delta_i^{\{u,l\}}$, and the counter b of the TA shown in Fig. 5(B) and Fig. 5(C) are local entities only, and will be indexed accordingly. The instances of Fig. 5(B) and Fig. 5(C) implement the single LTAs and UTAs of the network, respectively. The TA shown in Fig. 5(D) is the scheduler employed for governing *event*-emission, which is necessary since instead of binary synchronization we are using now Uppaal's concept of broadcast channels, where full synchronization among



(A) Minimum (maximum) of staircase functions as upper (lower) input curves

(B) TA for lower curve α_i^l

(C) TA for upper curve α_i^u

(D) Scheduler for emitting events

System Declarations

`broadcast channel` *event*;
`int` *Sync = 0;*

Declarations for LTA i

`clock` x;
`int` $b = 0$;

Declarations for UTA i

`clock` x;
`int` $b = N_i^u$;

(E) Declarations

Fig. 5 Convex, concave pattern: input curves and their TA-based implementation

the UTAs and LTAs has to be enforced. For bounding the number of producible events each instance of an UTA and LTA periodically increments its local counter b_i as before, namely every $\delta_i^{\{u,l\}}$ time units by executing the respective edge, which we denote as clock-tick edge from now on; it is the top edge of the TA templates of Fig. 5(B) and Fig. 5(C). The emission of an event can only take place if the minimum-condition applies, whereas event emission is enforced if the maximum-condition holds. In any case event emission yields an update of local counter b_i which allows each UTA and LTA to track event production accordingly. The minimum and the maximum-condition are implemented as follows:

1. The *minimum-condition* is enforced by the location invariant $Sync = Num\_UTA$ defined for the target location of the *event*!-edge of the scheduler (Fig. 5(D)). The invariant expresses the condition that the location can be entered only if the global variable *Sync* is equal to the number of UTA in the network. Since at each broadcast synchronization each UTA increments *Sync* by exactly 1, we have that event emission can take place only if *all* UTA in the network are participating.
2. The *maximum-condition* is realized by means of the location invariants of the different LTA. A *single* LTA enforces an event generation whenever executing the *event*!-edge is the only way for circumventing the violation of the invariant.

The usage of the unique label *event* within all TA guarantees the joint execution of the *event*!-edge of the scheduler and all *event*?-edges of the LTA and UTA, respectively. Thus either all *event*-edges are jointly executed or none, which yields the nice feature that an input generator of the above kind deadlocks if upper and lower bounding functions are not consistent. In the case of concave/convex input curves this is the case if the upper and the lower curve cross each other. Finally, one may note that as before LTA i resets its local clock x_i once $b_i = 0$ and event emission takes place. UTA i does so once $b_i = N_i^u$ holds and event emission occurs.

4.2.3 Tightness and correctness of the interface

In this subsection we will reason formally about the tightness and correctness of the described interface. In particular, we will show that $R^\alpha = R^{TA}$ holds also for the convex/concave pattern of staircase curves. This will be done in two steps. In step (A) we prove that the input generator cannot violate the upper and lower bounds $\alpha^{\{u,l\}}$, i.e., $R^\alpha \supseteq R^{TA}$. In step (B) we prove that the input generator can produce all the event traces that conform to $\alpha^{\{u,l\}}$, i.e., $R^\alpha \subseteq R^{TA}$. The final result is summarized in (C).

(A) Tightness ($R^\alpha \supseteq R^{TA}$)

Lemma 1 *Let R^{TA} denote the set of event traces producible by the input generator of Fig. 5. Let α^u be defined according to (7) and (8). Then, $r \models \alpha^u \;\; \forall r \in R^{TA}$.*

Sketch of Proof According to the above discussion the parameters N_i^u and δ_i^u of an UTA i correspond to the parameters of the respective step-function α_i^u. It is easy to see that *UTA* i allows the production of at most $N_i^u + \lfloor \frac{\Delta}{\delta_i^u} \rfloor$ events and that with $b_i = 0$ it blocks event production. Minimum-condition as defined above gives, that for any $\Delta \in \mathbb{R}^{\geq 0}$ the max. number of events producible is bounded by $\min_i(N_i^u + \lfloor \frac{\Delta}{\delta_i^u} \rfloor)$. It is important to note that for $b_i(t_0) = |N_i^u|$ an event generation resets clock x_i, such that the above equation yields an upper bound for the number of producible events. This is exactly what was defined for α^u in (7) and (8). □

For exemplification please refer to the graph of Fig. 5(A). If y events are produced in a time interval of length t, then the counter $b_3 = 0$ and hence UTA$_3$ blocks event production until its local clock x_3 expires. From that point on event production is bounded by α_3^u due to the minimum-operation realized by synchronizing the UTA.

Lemma 2 *Let R^{TA} denote the set of event traces producible by the input generator of Fig.* 5*. Let α^l be defined according to* (7) *and* (8)*. Then, $r \models \alpha^l \ \forall r \in R^{TA}$.*

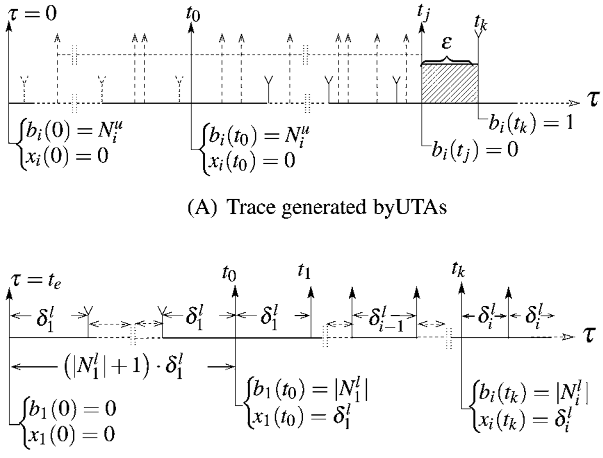
Sketch of Proof Due to the maximum-operation it seems appropriated to argue over the index of the LTA currently enforcing the generation of events and the size of the intervals:

- $0 \leq \Delta < (|N_1^l| + 1) \cdot \delta_1^l$: For intervals of this length event emission does not need to be enforced. Thus by choosing the parameters of LTA$_1$ accordingly the input generator is capable of delaying event emission up to $(|N_1^l| + 1) \cdot \delta_1^l$ time units, since starting with $b_1 = 0$ it is exactly this amount of time which it takes for LTA$_1$ to reach its event generation threshold ($b_1 = |N_1^l|$) and the local clock x_1 expiring ($x_1 = \delta_1^l$) given that no event has been emitted before. This implies also why clock x_1 needs to be reset when emitting an event in case of $b_1 = 0$ (cf. Fig. 5(B))
- $(|N_k^l| + 1) \cdot \delta_k^l \leq \Delta < (|N_{k+1}^l| + 1) \cdot \delta_{k+1}^l$: For each interval of this length LTA k bounds the minimum number of emitted events to $N_k^l + \lfloor \frac{\Delta}{\delta_k^l} \rfloor$, with $N_k^l \leq 0$. This is because when holding its threshold ($b_k = |N_k^l|$) and with the local clock x_k expiring ($x_k = \delta_k^l$) LTA k enforces the generation of an event and form now on every δ_k^l time units (see location invariant of the UTA of Fig. 5(C)). This goes on until LTA $k + 1$ holds its threshold $|N_{k+1}^l|$ and its local clock x_{k+1} expires.
- $(|N_{k+1}^l| + 1) \cdot \delta_{k+1}^l \leq \Delta < \infty$: For intervals of this size we may use the same argument as above, but now starting with LTA $k + 1$.

Hence, we can exclude the existence of an interval in which the input generator can generate less events than specified by α^l. □

For exemplification refer to Fig. 6(B) which illustrates a trace produced by the LTAs. Let event generation take place at time t_e and let $b_1(t_e) = 0$ afterwards s.t. clock x_1 is set to

Fig. 6 Timed *event*-traces and evaluations of $b_i(\tau)$ and $x_i(\tau)$



(A) Trace generated byUTAs

(B) Trace enforced by LTAs

zero. After another $(|N_1^l|+1)\cdot\delta_1^l$ time units where no event emission took place $b_1 = |N_1^l|$ and $x_1 = \delta_1^l$ will hold which immediately enforces event generation (here at time t_0). From now on this is done at least every δ_1^l time units, until $b_2 = |N_2^l|$ and $x_2 = \delta_2^l$ holds which enforces event production now every δ_2^l time units. In general this means that once started LTA $i-1$ enforces the event generation every δ_{i-1}^l time units. This goes on until $b_i = |N_i^l|$ and $x_i = \delta_i^l$ holds, which forces LTA i to take over event production, e.g. at time t_k as shown in Fig. 6(B).

Lemma 3 *Let R^α be the set of event traces that conform to $\alpha = [\alpha^u, \alpha^l]$ defined according to* (7) *and* (8). *Let R^{TA} denote the set of event traces producible by the input generator of Fig.* 5. *Then, $R^\alpha \supseteq R^{TA}$.*

Proof Follows directly from Lemmata 1 and 2. □

(B) Correctness ($R^\alpha \subseteq R^{TA}$)

Lemma 4 *Let R^{α^u} be the set of event traces that conform to α^u defined according to* (7) *and* (8). *Let R^{UTA} denote the set of event traces producible by the input generator of Fig.* 5 *without LTAs. Then, $R^{\alpha^u} \subseteq R^{UTA}$.*

Proof This will be shown by contradiction. One may assume that there is a trace r with $r \models \alpha^u$, but that is not producible by the network of UTAs. From this it follows that there must be a timed action $(event, t)$ where the UTAs are blocked, but the production of an additional event would not be contradictory to α^u. Let $(event, t) \in r$ but $(event, t) \notin r^{TA}$, where r^{TA} is a trace producible by the input generator with the same prefix as r a priori to the occurrence of $(event, t)$. Let $t := t_j + \epsilon$, it must hold that there $\exists b_i \in \mathscr{B}$, s.t. $b_i(t_j+\epsilon) = 0$, otherwise one would be able to produce an event. Let t_j be now the earliest point in time for r where $b_i(t_j) = 0$ holds and let $t_0 < t_j$ be the last point in time where $b_i(t_0) = N_i^u$ was satisfied, which is exactly what we have illustrated in Fig. 6(A). The choice of i and the blocking of events implies now that for the blocking period ϵ it must hold $t_j + \epsilon < t_k$, where t_k is the next time b_i is incremented and the generation of an event would not be blocked anymore. From this it follows that for the number of events $r(t_0, t_j+\epsilon)$ seen on r in the interval $[t_0, t_j+\epsilon]$ it must hold:

$$r(t_0, t_j+\epsilon) = N_i^u + \left\lfloor \frac{(t_j+\epsilon-t_0)}{\delta_i^u} \right\rfloor + 1 = \alpha_i^u(t_j+\epsilon-t_0) + 1,$$

otherwise b_i would not be 0. Obviously this violates the bound imposed by α^u, since at time point t_j the number of events is bounded by α_i^u which is the current minimum and truly smaller than $r(t_0, t_j+\epsilon)$. Thus from $r(0, t_j+\epsilon) > \alpha^u$ we can conclude that such a trace r does not exist. Concerning the assumption that t_0 was the last point in time where $b_i(t_0) = N_i^u$ was satisfied and that for t_k: $b_i(t_k) = 0$ must hold, one may note that for the initial point in time we have $b_i(0) := N_i^u$ and that b_i must be zero at t_j, since otherwise UTA would not block event emission by violating the minimum-condition. □

Lemma 5 *Let R^{α^l} be the set of event traces that conform to α^l defined according to* (7) *and* (8). *Let R^{LTA} denote the set of event traces producible by the input generator of Fig.* 5 *without UTAs. Then, $R^{\alpha^l} \subseteq R^{LTA}$.*

Sketch of Proof Analogously to the proof of Lemma 4, this can be shown by contradiction. We assume that there is a trace r with $r \models \alpha^l$, but that is not producible by the network of LTAs. In particular, we assume that for a given time interval $[s, t]$ with $\Delta = t - s$ the trace r contains less events than the minimum number of events enforced by the network of LTAs for any time interval of length Δ. By reasoning about the behavior of the network of LTAs for different interval sizes as done in the proof of Lemma 2, it can be shown that such an event trace does not exist. □

Lemma 6 *Let R^α be the set of event traces that conform to $\alpha = [\alpha^u, \alpha^l]$ defined according to (7) and (8). Let R^{TA} denote the set of event traces producible by the input generator of Fig. 5. Then, $R^\alpha \subseteq R^{TA}$.*

Proof Follows directly from Lemmata 4 and 5. □

(C) Identity ($R^\alpha = R^{TA}$)

Theorem 2 *Let R^α be the set of event traces that conform to $\alpha = [\alpha^u, \alpha^l]$ defined according to (7) and (8). Let R^{TA} denote the set of event traces producible by the input generator of Fig. 5. Then, $R^\alpha = R^{TA}$.*

Proof Follows directly from Lemmata 3 and 6. □

4.2.4 Extended input generators

In practice systems may show event streams that do not adhere to the above described input pattern. While in such cases the pattern can still be used to conservatively approximate the arrival curves and hence to realize a correct input interface, it is desirable to avoid approximations for the sake of accuracy. In the present subsection, we will therefore briefly sketch two possible refinements of the input pattern.

Shifted staircase curves In the input pattern described in Sects. 4.2.1 and 4.2.2 each linear staircase curve has a uniform step width for all steps. In practice one does, however, often encounter staircase curves that have an initial offset, meaning that they are horizontally shifted. For instance, this is the case for the arrival curves of a periodic event stream with jitter. Figure 7 illustrates an example of such horizontally translated staircase curves.

Such staircase curves with initial offset can be modeled by a slightly more general version of the TA shown in Fig. 5. In the following we will explain the underlying principle by means of $\hat{\alpha}_i^u$. The corresponding automaton $\hat{UTA}_i$ is shown in Fig. 8. The differences to the automaton UTA_i are that at each event generation the counter b_i is decreased by e_i^u units, and that scaled constants $\hat{N}_i^u$ and $\hat{\delta}_i^u$ are used as maximum counter value and counter increase period, respectively.

The idea for achieving the initial offset θ_i^u is now to use a value for e_i^u that is not a factor of $\hat{N}_i^u$. In this way, after the generation of the maximum instantaneous burst of events, the next event can be generated earlier compared to the previous case. This is because after the generation of the maximum burst, the counter will be equal to a residual value $0 < b_i < e_i^u$. The following equations permit to derive e_i^u, $\hat{N}_i^u$, and $\hat{\delta}_i^u$ for given values of θ_i^u, N_i^u, and δ_i^u:

$$\hat{\delta}_i^u = \gcd(\delta_i^u, \theta_i^u); \qquad e_i^u = \frac{\delta_i^u}{\hat{\delta}_i^u}; \qquad \hat{N}_i^u = (N_i^u + 1) \cdot e_i^u - \frac{\theta_i^u}{\hat{\delta}_i^u}. \tag{9}$$

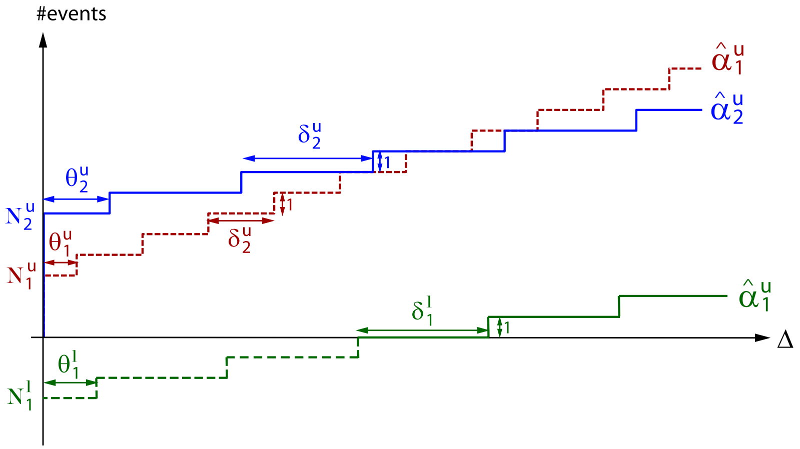
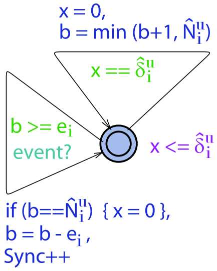


Fig. 7 Shifted staircase curves with offset $\theta_i^{\{u,l\}}$

Fig. 8 TA for shifted upper staircase curve $\hat{\alpha}_i^u$



Consider for instance $\hat{\alpha}_2^u$ shown in Fig. 7 and defined by $N_2^u = 6$, $\delta_2^u = 8$, and $\theta_2^u = 4$. By applying (9) we obtain $\hat{\delta}_2^u = 4$, $e_2^u = 2$, and $\hat{N}_2^u = 13$. One can easily verify, that by using those constants in the automaton of Fig. 8, we generate event traces which conform to $\hat{\alpha}_2^u$.

For lower curves $\hat{\alpha}_i^l$ the reasoning is analogous and results in the following equations:

$$\hat{\delta}_i^l = \gcd(\delta_i^l, \theta_i^l); \qquad e_i^l = \frac{\delta_i^l}{\hat{\delta}_i^l}; \qquad \hat{N}_i^l = N_i^l \cdot e_i^l - \frac{\theta_i^l}{\hat{\delta}_i^l} + 1. \tag{10}$$

Non-convex/concave patterns Another issue is that in practice systems may sometimes not show strictly concave or convex patterns. For instance, the overall upper input or output curves may have parts with decreasing step widths (see, e.g., $\alpha'^u_{2,\mathrm{RTC}}$ in Fig. 17), or the lower curve may contain parts with increasing ones. In the following we will briefly sketch how one can resolve such situations for the input interface.

In cases where non-concave and non-convex patterns occur only finitely often within an arrival curve, one can handle this by simply making use of subsets of UTA, LTA and local synchronization for obtaining local minima and maxima. In order to implement such patterns one solely needs to encapsulate the respective sets of co-operating LTA or UTA in their own sub-system. These subsystems can be implemented analogously to the pattern illustrated above, but requiring slightly adapted TA-specifications w.r.t. the employed thresholds.

Note that in the case of non-concave/convex input curves the specification of an upper and a lower bound might be inconsistent even if the two curves do not intersect. This problem is described in more detail in [1]. In our framework the input generator can run into a deadlock in the presence of such an inconsistency. Hence, we can easily detect such a case by model checking a corresponding query.

4.2.5 Complexity issues related to input modelling

Complexity of model checking TA is exponentially bounded by the number of clocks and clock constants [2]. Thus it is straight forward to see that the efficiency of the approach is closely related to the number of staircase functions employed for modeling lower and upper input curves.

In the following we propose a simple method that permits to approximate a general arrival curve with the convex/concave combination of just a few staircase functions. The approach first approximates the arrival curve by a so called periodic with jitter event arrival model, and then derives the parameters for the corresponding staircase curves $\alpha_i^{\{u,l\}}$. The periodic with jitter event model (or PJD model in short) is commonly used in literature and is a simple representation for the arrival of events in streams [23]. It is specified by a parameter triple (p, j, d), where p denotes the period, $j$the jitter and d the minimum inter-arrival time of events in the modeled stream.

Arrival curves are in general more expressive than PJD models. However, every arrival curve can be conservatively approximated by a PJD model [14]. Given a general arrival curve to feed into a TA-based component, we first use the algorithm described in [14] to approximate it with a PJD model. Subsequently, we convert the PJD parameters to a set of appropriate parameters $N_i^{\{u,l\}}$ and $\delta_i^{\{u,l\}}$ that are used to specify the input generator for the TA-based component as described in Sect. 4.2.2.

The upper bound described by a PJD model can be represented by the minimum of at most two staircase functions α_1^u and α_2^u. In particular, two staircase functions are needed if $d > 0 \wedge d > p - j$, while only one staircase function suffices otherwise. For the lower bound of a PJD model one staircase function α^l is always sufficient. The parameters of the staircase functions are computed as follows:

- Case $d = 0 \vee d \leq p - j$:
  $N^u := \left\lceil \frac{j}{p} \right\rceil + 1; \quad N^l := -\left\lceil \frac{j}{p} \right\rceil; \quad \delta^u := \delta^l := p$
- Case $d > 0 \wedge d > p - j$:
  $N_1^u := 1; \quad \delta_1^u := d; \quad N_2^u := \left\lceil \frac{j}{p} \right\rceil + 1; \quad N^l := -\left\lceil \frac{j}{p} \right\rceil; \quad \delta_2^u := \delta^l := p$

Note that an exact representation of a PJD model by means of staircase functions α_1^u, α_2^u and α^l is not always possible if we exclude horizontal shifting of staircase functions. However, in such a case the above formulae guarantee a correct (i.e., conservative) approximation of the PJD model. On the other hand, if we use the generalized input model for shifted staircase functions described in Sect. 4.2.4, then we can exactly represent any PJD model by means of at most three staircase functions. In this case the parameters of the staircase functions are computed as follows:

- Case $d = 0 \vee d \leq p - j$:
  $\hat{N}^u = x \in \mathbb{N}^+$, $e^u = y \in \mathbb{N}^+$ such that $\frac{x}{y} = \frac{j}{p} + 1 \wedge \gcd(x, y) = 1; \quad \hat{\delta}^u = \frac{p}{e^u}$
  $\hat{N}^l = v \in \mathbb{N}^+$, $e^l = w \in \mathbb{N}^+$ such that $\frac{v}{w} = \frac{j}{p} \wedge \gcd(v, w) = 1; \quad \hat{\delta}^l = \frac{p}{e^l}$
- Case $d > 0 \wedge d > p - j$:
  $\hat{N}_1^u = 1; \quad e_1^u = 1; \quad \hat{\delta}_1^u = d$

$$\hat{N}_2^u = x \in \mathbb{N}^+,\ e_2^u = y \in \mathbb{N}^+ \text{ such that } \tfrac{x}{y} = \tfrac{j}{p} + 1 \ \wedge\ \gcd(x,y) = 1; \quad \hat{\delta}_2^u = \tfrac{p}{e_2^u}$$
$$\hat{N}^l = v \in \mathbb{N}^+,\ e^l = w \in \mathbb{N}^+ \text{ such that } \tfrac{v}{w} = \tfrac{j}{p} \ \wedge\ \gcd(v,w) = 1; \quad \hat{\delta}^l = \tfrac{p}{e^l}$$

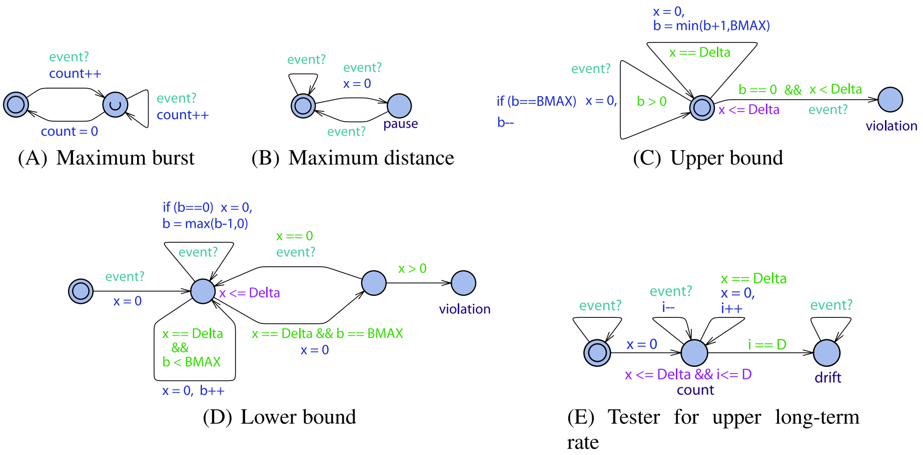
While the approximation of arrival curves with PJD models represents a simple way to coarsely bound an event stream with few staircase functions, in the presented hybrid analysis approach the interface between RTC and TA is of course not limited to PJD curves. Any other algorithm that correctly bounds a general arrival curve with an arbitrary number of staircase functions $\alpha_i^{\{u,l\}}$ can be used as an interface between the two domains.

Now that we have described the interfacing from RTC-based model descriptions to TA, we will discuss the interfacing from TA-systems back to RTC-conforming performance models.

4.3 Output interface

This subsection describes the realization of the output interface, that is, the bounding of the output of a TA subsystem by means of a tuple of arrival curves $\alpha = [\alpha^u, \alpha^l]$. As described in Sect. 4.1, the requirement for a correct output interface is $R^\alpha \supseteq R^S$, where R^S denotes the set of event traces producible by the TA subsystem. In other words, the output of the TA subsystem can be approximated, but the approximation has to be done in a conservative manner. The main concept used for constructing valid output curves $\alpha'^{\{u,l\}}$ can be considered just the reverse of event generation: We construct a set of staircase functions $\alpha_i'^u$ and $\alpha_i'^l$ for the output of the TA subsystem which allows to construct an overall output curve $\alpha'^{\{u,l\}}$ by means of minimum and maximum operators, respectively. For achieving this goal, we couple the system under analysis including the input generator with a set of observing TA. Checking reachability queries for these TA-systems allows to derive the parameters $N_i^{\{u,l\}}$ and $\delta_i^{\{u,l\}}$ that uniquely characterize $\alpha_i'^u$ and $\alpha_i'^l$. In the following we will first describe the TA that are used to verify individual staircase parameters. After that, we will describe the overall composition strategy for constructing a valid output curve $\alpha'^{\{u,l\}}(\Delta)$.

For implementing the above procedure we will employ the observers TA depicted in Fig. 9:



(A) Maximum burst

(B) Maximum distance

(C) Upper bound

(D) Lower bound

(E) Tester for upper long-term rate

Fig. 9 Observer automata for deriving upper and lower output curves

(a) *Maximum burst size:* An upper bound for the maximum number of events that the system can generate simultaneously can be verified by means of the observer depicted in Fig. 9(A) and the query:[1] `A[] (count<=estimate)`.

(b) *Maximal distance between two successively emitted events:* We can verify a bound on the maximum pause time between two output events by employing the observer shown in Fig. 9(B) and the query `A[] (pause imply x<=estimate)`.

(c) *Arbitrary upper staircase curve $\alpha_i'^u$:* For obtaining an individual staircase function we employ the observer TA of Fig. 9(C) which witnesses the violation or invulnerability of the respective curve. The witnessing TA moves into the location `violation`, once the respective curve is violated, i.e. the actual system produces too many events. Thus one simply needs to query for the reachability of location `violation` (`A[] (not violation)`). In other words, given some staircase parameters N_i^u and δ_i^u, we can determine whether the corresponding staircase function is a valid upper bound in time-interval domain of the produced event stream.

(d) *Arbitrary lower staircase curve $\alpha_i'^l$:* For obtaining an individual lower staircase function we employ observer TA of Fig. 9(D) and use the same principle as described above: Given some parameters N_i^l and δ_i^l, we can determine whether the corresponding staircase function is a valid lower bound in time-interval domain of the produced event stream.

(e) *Long-term rates:* In order to construct output curves $\alpha'^{\{u,l\}}$ that approximate the system behavior well also for large time intervals, we need to make sure that we follow the long-term event output rate. By long-term rate of an arrival curve α we mean the inverse of the limit $\lim_{\Delta\to\infty}\frac{\alpha(\Delta)}{\Delta}$, which always exists as detailed in [12]. The largest δ_i^u and the smallest δ_i^l of *any* valid upper and lower output staircase function, respectively, denote upper and lower bounds on the long-term rate of the output. The principle of efficiently verifying that a given staircase function represents this upper or lower bound will be explained by means of $\alpha_i'^u$. The procedure for $\alpha_i'^l$ is analogous and is omitted for conciseness. The idea is to verify whether the observed system can produce an event trace such that for arbitrary long intervals the rate of the trace is not slower than δ_i^u. To do so one may employ the TA depicted in Fig. 9(E). This TA monitors the difference between the number of event arrivals allowed by rate δ_i^u and the number of events actually produced by the observed system. Once this difference exceeds a constant `D`, the TA moves to the location `drift`. If there is a trace for which the observer TA stays indefinitely in the location `count`, it means that we have found a trace that on the long-term never gets slower than δ_i^u, i.e., the rate δ_i^u is not overly pessimistic for the system output. Such a trace can be found as counterexample to the query: `count -> drift`.[2]

Now, we will describe how the TA introduced above can be used to construct valid upper and lower bound curves $\alpha'^{\{u,l\}}(\Delta)$ of the system output. There are many possibilities and the following list summarizes only a few of them:

- A binary search on `estimate` (see (a) and (b)) yields the maximum burst size and the maximal pause time, respectively.
- These values can be used to determine one degree of freedom (of the available two) of an upper or lower staircase automaton, see (c) and (d). For example, using the maximal burst size from (a) and a binary search on the remaining parameter δ_i^u in the automaton of (c) yields an upper staircase function $\alpha_i'^u$ that characterizes the maximal burst rate.

[1] In Uppaal `A[]` stands for 'always invariantly'.

[2] In Uppaal `->` stands for 'always eventually leads to'.

- Fixing any of the two free parameters in the automata of (c), (d) and performing a binary search on the other yields a valid upper and lower staircase bound, respectively.
- Using the automaton of (c) with a large initial burst capacity N_i^u and performing a binary search on δ_i^u leads to a tight upper bound on the maximal long-term rate.
- One may determine a convex hull of the lower and a concave hull of the upper (unknown) RTC-curves of the event stream by calculating a sequence of staircase functions. For example, in case of an upper curve, we can consider a sequence of increasing values N_i^u and use the automaton (c) to determine the corresponding maximal values δ_i^u that bound the system output. The sequence ends if the long-term rate is met, see (e).
- All constructed valid upper and lower staircase functions can be combined to a valid bounding curve by minimum and maximum operations, respectively.

Algorithm 1 defines a simple heuristic procedure for bounding the output of a TA subsystem. The algorithm makes use of the TA of Fig. 9 and will be employed in the case study and in the other experiments of Sect. 5.3. The heuristic returns four vectors $\bar{N}^u$, $\bar{\delta^u}$, $\bar{N}^l$, $\bar{\delta^l}$ that contain the parameters for the linear staircase curves used to bound the output of the TA system. The input parameters have the following meaning:

n, m Maximal number of staircase curves $\alpha_i'^u$ and $\alpha_i'^l$, respectively, that shall be used to bound the system output
B_{MIN}, B_{MAX} Delimit the interval in which the maximum burst size is searched
P_{MIN}, P_{MAX} Delimit the interval in which the maximum pause between two events is searched
$\delta_{MIN}, \delta_{MAX}$ Delimit the interval in which the parameters δ_i^l and δ_i^u are searched
k Scaling factor > 1 for N_i^u, N_i^l

In line 5 the heuristic determines the maximum burst size N_1^u in the system output. This is done by means of the function max_burst which implements a binary search coupled with model checking of the system. In particular, in line 37 the function max_burst calls the Uppaal model checker to verify whether in the observer TA of Fig. 9(A), denoted as OMB, a given event counter value is never exceeded. Similarly, in line 6 the maximum value of δ_1^u is determined by the function max_delta such that the system output never violates the bound specified by the parameter tuple (N_1^u, δ_1^u). In this case the model checker is used to verify whether the observer TA of Fig. 9(C), denoted as OUB, can reach its violation location (line 52). At this point the first staircase curve $\alpha_1'^u$ is fixed. Next, the heuristic enters a loop (line 8) in which at most $n-1$ other staircase curves are determined for the upper bound. At each loop iteration the value T_i^u is scaled by a factor k, where T_i^u represents a tentative value for N_i^u. Line 10 is equivalent to line 6, however when looking for the largest valid δ_i^u the algorithm considers that $\delta_i^u > \delta_{i-1}^u$ and hence uses tighter bounds for the binary search. In line 11 the heuristic calls the function min_N_upper which determines N_i^u by verifying whether the found staircase curve with parameters (T_i^u, δ_i^u) can be further shifted down vertically without being violated by the system output. This is possible, as the staircase curve (T_i^u, δ_i^u) is not necessarily the smallest valid staircase curve with rate δ_i^u. The function min_N_upper is analogous to max_delta, with the only difference that the binary search is carried out on N_i^u instead of δ_i^u. The corresponding pseudo-code is omitted for conciseness. After $\alpha_i'^u$ is fixed, the heuristic calls the function isLongTermRate which uses the TA of Fig. 9(E) to verify whether δ_i^u corresponds to the long-term rate of the system output. If this is the case, there is no point in further increasing T_i^u and the approximation of the upper bound terminates. The derivation of the lower bound for the system output follows the same line of thoughts, with analogous functions max_pause, min_delta_pause,

Algorithm 1 `Bound output of TA component`

```
function DERIVE_BOUNDS
input: $n, m, k, B_{MIN}, B_{MAX}, P_{MIN}, P_{MAX}, \delta_{MIN}, \delta_{MAX}$
output: $\bar{N^u}, \bar{\delta^u}, \bar{N^l}, \bar{\delta^l}$
    // Upper bound
    $N_1^u$ ←max_burst($B_{MIN}, B_{MAX}$)
    $\delta_1^u$ ←max_delta($N_1^u, \delta_{MIN}, \delta_{MAX}$)
    $T_1^u \leftarrow N_1^u$
    for $i \leftarrow 2, n$ do
        $T_i^u \leftarrow k * T_{i-1}^u$
        $\delta_i^u$ ←max_delta($T_i^u, \delta_{i-1}^u, \delta_{MAX}$)
        $N_i^u$ ←min_N_upper($\delta_i^u, N_{i-1}^u, T_i^u$)
        if isLongTermRate($\delta_i^u$) then
            break
        end if
    end for
    // Lower bound
    $P$ ←max_pause($P_{MIN}, P_{MAX}$)
    $[N_1^l, \delta_1^l]$ ←min_delta_pause($P, \delta_{MIN}, \delta_{MAX}$)
    $T_1^l \leftarrow N_1^l$
    for $i \leftarrow 2, m$ do
        $T_i^l \leftarrow k * T_{i-1}^l$
        $\delta_i^l$ ←min_delta($T_i^l, \delta_{MIN}, \delta_{i-1}^l$)
        $N_i^l$ ←max_N_lower($\delta_i^l, N_{i-1}^l, T_i^l$)
        if isLongTermRate($\delta_i^l$) then
            break
        end if
    end for
    remove_redundant_bounds()
    return $\bar{N^u}, \bar{\delta^u}, \bar{N^l}, \bar{\delta^l}$
end function

function MAX_BURST
input: $est_{min}, est_{max}$
output: $N$
    $est \leftarrow \lceil (est_{min} + est_{max})/2 \rceil$
    repeat
        $est_{old} \leftarrow est$
        if verifyta ( A[] (OMB.count $\leq est$) ) = satisfied then
            $est_{max} \leftarrow est$
        else
            $est_{min} \leftarrow est$
        end if
        $est \leftarrow \lceil (est_{min} + est_{max})/2 \rceil$
    until $est = est_{old}$
    return $est$
end function

function MAX_DELTA
input: $N, est_{min}, est_{max}$
output: $\delta$
    $est \leftarrow \lceil (est_{min} + est_{max})/2 \rceil$
    repeat
        $est_{old} \leftarrow est$
        if verifyta ( A[] (not OUB.violation) ) = satisfied then
            $est_{min} \leftarrow est$
        else
            $est_{max} \leftarrow est$
        end if
        $est \leftarrow \lceil (est_{min} + est_{max})/2 \rceil$
    until $est = est_{old}$
    return $est_{min}$
end function
```

and max_N_lower that employ the TA of Fig. 9(B), Fig. 9(D), as well as an adapted version of the TA in Fig. 9(E). One difference to the upper bound is that we cannot directly compute N_1^l given the value of P, the maximum pause between two events. In particular, there are multiple staircase curves that contain the cartesian point $(P, 0)$. Hence, in line 18 the heuristic calls the function min_delta_pause which looks for a curve $\alpha_1'^l$ that contains $(P, 0)$ and has the smallest value of δ_1^l such that $\alpha_1'^l$ is a valid lower bound for the system output. At this point N_1^l is also determined. Finally, in line 28, the heuristic removes redundant staircase curves, that is, $\alpha_i'^u$ for which we have

$$\exists \alpha_j'^u : \big(\big(N_j^u < N_i^u\big) \wedge \big(\delta_j^u \geq \delta_i^u\big)\big) \vee \big(\big(N_j^u \leq N_i^u\big) \wedge \big(\delta_j^u > \delta_i^u\big)\big), \tag{11}$$

and $\alpha_i'^l$ for which we have

$$\exists \alpha_j'^l : \big(\big(\big|N_j^l\big| < \big|N_i^l\big|\big) \wedge \big(\delta_j^l \leq \delta_i^l\big)\big) \vee \big(\big(\big|N_j^l\big| \leq \big|N_i^l\big|\big) \wedge \big(\delta_j^u < \delta_i^u\big)\big). \tag{12}$$

If after termination the upper (lower) long-term rate of the system is not reached, we can either use a larger value for the parameter n (m), or try a larger value for the scaling factor k. In many practical systems, however, the long-term rates of the system output are known a priori. For instance, it is often the case that a component changes the jitter of an event stream, but not its period. In such cases, it is much better to adopt an inverse search strategy in the heuristic. For instance, for the upper bound one would start from the known long-term rate δ_n^u and derive the corresponding value N_n^u in order to fix the last staircase curve $\alpha_n'^u$. $\alpha_1'^u$ could be found as described before using the maximum burst size of the output. Successively, one would use different values N_i^u with $N_1^u < N_i^u < N_i^u$ and find the corresponding values δ_i^u to refine the upper bound.

It remains to show that the heuristic of Algorithm 1 guarantees the correctness of the output interface.

Theorem 3 *Let R^S be the set of event traces producible by a TA subsystem S. Let $\alpha' = [\alpha'^u, \alpha'^l]$ be a tuple of arrival curves derived for the output of S by means of the heuristic of Algorithm 1. Then, $R^{\alpha'} \supseteq R^S$.*

Sketch of Proof We illustrate the idea for the justification of the upper bound α'^u. The reasoning for the lower bound α'^l is analogous. Let $\bar{N}^u$ and $\bar{\delta}^u$ be the parameter vectors derived by the heuristic for the output of S. Let $\alpha_i'^u$ with $i \in \{1, \ldots, n\}$ be the staircase curves defined by those parameters. It is sufficient to show that for each individual staircase curve $\alpha_i'^u$ we have $R^{\alpha_i'^u} \supseteq R^S$, that is, for each output event trace r producible by S we have $r \models \alpha_i'^u$. Consider first $\alpha_1'^u$. The function max_burst called in line 5 implements a binary search on the maximum burst in the output of S. By using the observer TA of Fig. 9(A), the function verifies that a conservative estimate N_1^u is returned for the maximum burst in the stream. Similarly, given N_1^u, by means of the TA of Fig. 9(C) the function max_delta guarantees that a value δ_1^u is returned such that $\alpha_1'^u$ is never violated by the output of S. Hence, $r \models \alpha_1'^u \; \forall r \in R^S$. The same argument holds also for all successive calls of max_delta, since the scaling factor k is such to assure $N_i^u \geq N_1^u$. Thus, $r \models \alpha_i'^u \; \forall r \in R^S \; \forall i \in \{1, \ldots, n\}$. □

5 Experimental evaluation

In this section we evaluate the performance of the proposed analysis methodology. We will first discuss a case study that demonstrates the benefits of the hybrid analysis approach. Subsequently, we will elaborate on the scalability and accuracy of the presented method.

5.1 Case study

The considered system is shown in Fig. 10. It consists of three event-triggered tasks T_1, T_2 and T_3 that run on two distinct processors CPU_1 and CPU_2. We assume that each task is triggered by the events of the corresponding input event stream and that it produces an event on the corresponding output event stream once its execution is completed. The three tasks process two event streams S_A and S_B which are periodic streams with large jitters that lead to bursts. S_A and S_B are specified by the parameter triples $p_A = 7$ ms, $j_A = 28$ ms, $d_A = 1$ ms and $p_B = 7$ ms, $j_B = 23$ ms, $d_B = 6$ ms, respectively. CPU_2 implements a preemptive fixed-priority scheduling policy with T_2 having higher priority than T_3. The execution of each task on its respective CPU takes 10^6 cycles. CPU_2 operates at a constant frequency of 350 MHz. CPU_1 implements a load-dependent frequency adaptation. In particular, it operates at 166 MHz if there are not more than 3 events in its input buffer, and at 500 MHz otherwise. Note that, for the sake of simplicity, we assume that the CPU frequency cannot be changed during the processing of an event. That is, the new CPU frequency is chosen only at the beginning of an event processing (depending on the current buffer fill level) and this frequency is kept constant until the next event processing starts. The goals of the performance analysis are to characterize the event output stream of T_1, to determine the maximum delays and backlogs that events can experience at the single tasks, and to find the maximum end-to-end delay for stream S_A.

In this case study we will compare three different approaches: First we analyze the described system with the abstraction of RTC only using the MPA Toolbox [16, 23]. Subsequently, we carry out the analysis with the presented hybrid analysis approach, where we model the state-dependent behavior of CPU_1 as TA and analyze CPU_2 with RTC. Finally, we verify the performance of the system by means of a dedicated TA model according to the method described in [10], which permits to exploit the simple periodic nature of the input streams.

For the *hybrid analysis approach*, we first represent the input stream S_A by the combination of three staircase functions α_1^u, α_2^u and α^l. Using the equations of Sect. 4.2.5 we get the parameters $N_1^u := 1$, $\delta_1^u := 1$, $N_2^u := 5$, $\delta_2^u := 7$, $N^l := -4$ and $\delta^l := 7$ for the staircase functions. The corresponding event curve α_{S_A} is shown in Fig. 12(A). Given these parameters we automatically create the input generator as described in Sect. 4.2.2. In order to increase the efficiency of the analysis, we merge the input generating network of TA into a single automaton and simplify it slightly by considering that $N_1^u = 1$, that is, for α_1^u we do actually not need a counter variable b, but just a clock to enforce a minimum distance δ_1^u

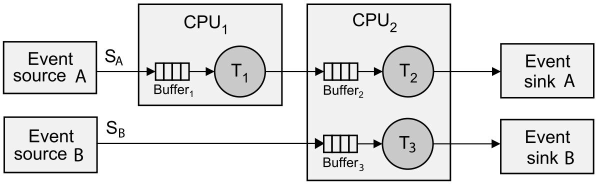


Fig. 10 System architecture

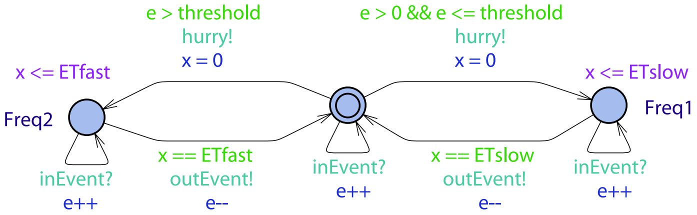


Fig. 11 TA model for CPU_1

Table 1 Results of performance analysis

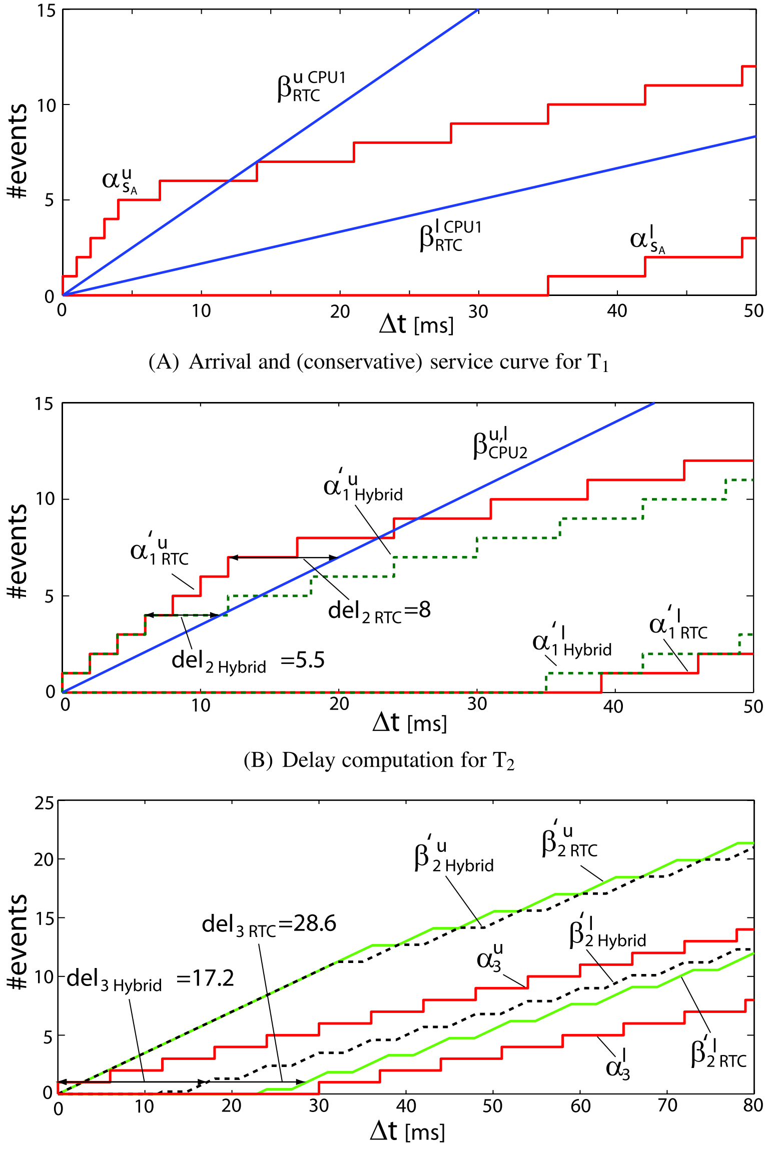
	Max delay [ms]				Max buffer [events]		
	T_1	T_2	T_3	EE_A	T_1	T_2	T_3
RTC	29	8	28.6	31.9	5	3	5
Hybrid	25	5.5	17.2	30.5	5	2	3
TA	25	4.6	14.3	27.9	5	2	3

between consecutive events. This input generator is then coupled with the automaton shown in Fig. 11 which models the load-dependent behavior of CPU_1. In this automaton we use the signals *inEvent* and *outEvent* to distinguish between ingoing events coming from the Event Source A and outgoing events sent to T_2. *Buffer1* of CPU_1 is modelled by means of a local counter variable *e*. The two locations *Freq1* and *Freq2* represent the processing of events at low and high frequency, respectively, with corresponding processing times *ETslow* and *ETfast*. The signal *hurry* belongs to an urgent channel which is always ready for synchronization. This construct enforces greedy event processing. At this point we apply the heuristic of Sect. 4.3 to get arrival curves that bound the output of the TA subsystem, where we choose to represent the upper bound as the minimum of three staircase functions and the lower bound with just one staircase function. The resulting pair of arrival curves is then used as input for the RTC analysis of CPU_2. For the analysis of the maximum delay on CPU_1 in the hybrid setting, we customize the automaton of Fig. 11 following the ideas of [10].

Table 1 summarizes the results of the performance analysis. The worst-case end-to-end delay of stream S_A is denoted as EE_A. Note that in general for a sequence of components the worst-case end-to-end delay can be smaller than the sum of the individual worst-case delays. While in the abstractions of RTC and TA this phenomenon can be captured for EE_A, this is not possible in the hybrid approach.

As can be seen in the table, in terms of accuracy the *hybrid approach* is clearly better than the pure RTC analysis. In particular, the conservativeness of the results is highly reduced, with a maximum delay and backlog at T_2 that are 31% and 33% lower with respect to the RTC analysis, respectively. For the delay and the backlog at T_3, the hybrid approach achieves values that are 40% lower compared to the pure RTC analysis.

The reason for the better results can be understood by looking at Fig. 12, where we employed the RTC-related labeling, i.e. the α-curves refer to input streams of events, the α'-curves to their outgoing counterparts and the β- and β'-curves to the ingoing and outgoing streams of available resources, respectively. A *pure RTC-based analysis* of the above scenario cannot capture the load-dependent behavior of CPU_1. Hence, one has to assume that the processor always operates at 500 MHz in the best case and at 166 MHz in the worst case. This assumption corresponds to using the service curves $\beta_{\mathrm{RTC}}^{uCPU1}$ and $\beta_{\mathrm{RTC}}^{lCPU1}$ (cf. Fig. 12(A)) for the analysis of CPU_1. This yields conservative worst-case processing load



(A) Arrival and (conservative) service curve for T_1

(B) Delay computation for T_2

(C) Delay computation for T_3

Fig. 12 Curves associated with the case study

predictions captured by $\alpha'^{u}_{1,\mathrm{RTC}}$ for T_2. However, a TA-based analysis of CPU_1 produces tighter input bounds captured by $\alpha'^{u}_{1,Hybrid}$ for the RTC analysis of T_2. This leads to smaller worst case delay guarantees, as shown in Fig. 12(B) and 12(C).

The last line of Table 1 contains the exact values for the worst-case performance of the system. These values are determined by means of the *dedicated TA model for the entire system*. As can be seen in the table, the results for the hybrid analysis approach are slightly more conservative. The reason is that the concave (convex) hull determined as bound for the

Table 2 Run-times for performance analysis (All run-times in this paper are referred to a commodity computer with a dual core CPU and 2 GB of RAM)

	RTC	Hybrid	TA
Total run-time	< 1 s	11 min	1 h

output event stream of T_1 does slightly over- (under)-approximate the real behavior of the system. The graphs of Fig. 12 do not show arrival and service curves for the exact internal behavior of the system, as these interfaces are not intrinsic to the dedicated TA model.

The higher degree of accuracy of the hybrid analysis method in comparison to the pure analytic RTC approach has its price, namely a substantially longer run-time, as can be seen in Table 2. This becomes worse if one keeps in mind that we already decided to bound the output curves by a convex (concave) pattern of three staircase functions only. In case of requiring a higher degree of accuracy one needs to adapt the proposed scheme in order to detect non-convex and non-concave patterns and its additional staircase functions. But this once again comes along with clearly higher computation times. Nevertheless, the run-times achieved for the hybrid approach are still significantly better compared to the verification of the pure TA model.

Furthermore, we have observed that for the hybrid approach the run-times for deriving an output curve from a TA component can be considerably reduced if for the representation of the input stream we omit the lower bound, that is, in the event generator we use only UTAs and leave out the LTAs. In the considered case study this corresponds to representing the event stream S_A by the upper bound $\alpha^u_{S_A}$ of Fig. 12(A) only, without specifying the lower bound $\alpha^l_{S_A}$. Note that such a relaxation of the stream specification does not harm the correctness of the analysis. In particular, by omitting the lower bound we specify a superset of timed input event streams with respect to the case with both, upper and lower bounds. That is, all behaviors of the original model are contained in the relaxed model and hence the analysis is safe. However, depending on the behavior of the modeled system component, considering more input streams then in the original model might lead to more conservative analysis results. In the system of Fig. 10 this is not the case, meaning that the same analysis results are achieved when representing the stream S_A without LTA compared to the case with LTA. In terms of verification effort the difference is, however, substantial; by leaving out the LTA the run-time of the hybrid approach is reduced from 11 min to 18 s, which shows that the synchronization of UTAs and LTAs in the input generators is one of the major sources of complexity in the discussed analysis methodology.

5.2 Scalability of the approach

In this subsection we report the results of two different experiments that investigate the scalability of the proposed analysis method. The first experiment demonstrates the clear superiority of the presented compositional methodology in terms of scalability of the verification effort with respect to holistic TA models. The second experiment points out the main limitation of TA-based performance analysis in general, namely poor scalability with respect to non-determinism in the specification of event streams.

5.2.1 Modular vs. holistic TA analysis

In this experiment we consider a larger distributed system consisting of several state-based components. We compare two different TA-based methods for the analysis of the system.

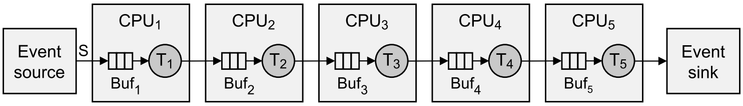


Fig. 13 System instance with five components

Table 3 Parameters for the CPU chain

	CPU_1	CPU_2	CPU_3	CPU_4	CPU_5
f_{low} [MHz]	166	166	166	166	166
f_{high} [MHz]	1000	500	333	1000	500
threshold [events]	1	1	1	1	1

The first approach performs holistic analysis based on a single TA model of the entire system. In the second approach the analysis is strictly modular. More precisely, in the second case each component of the system is analyzed separately by an individual TA model, where we use the staircase-curve based interfaces introduced in this paper to represent the input and output event streams of the components. Obviously, characterizing the input/output interfaces of each component explicitly by appropriate staircase curves will comport some verification overhead. However, we expect better scalability for the modular analysis approach, as in contrast to the holistic method the analysis of each component is totally decoupled from other components. In order to highlight how well the two different approaches scale with the size of systems, we gradually increase the number of components in a predefined system architecture and compare the results and run-times of the analysis methods.

The considered system template is a chain of n tasks, where each task executes on a dedicated processor. We assume that the execution of each task takes 10^6 processor cycles. The tasks successively process the events of an input event stream S. Figure 13 shows an instance of the system for $n = 5$.

Each CPU in the chain implements a load-dependent frequency adaptation (see details below). For the experiments we consider five different system instances, from $n = 1$ to $n = 5$. That is, the first instance consists of T1/CPU1 only, the second instance of T1/CPU1 and T2/CPU2, etc. In order to allow for event bursts also at the last components of the chain, we choose different maximum frequencies for the five processors. The parameters for the processors are summed up in Table 3. The load-dependent frequency adaptation works as follows: if there are not more than *threshold* events in the input buffer of a CPU, it will execute at frequency f_{low}, otherwise at f_{high}, where again we exclude frequency changes during the processing of an event.

The aim of the performance analysis is to determine, for each system instance, the worst-case backlogs at the single event buffers. Note that since we consider a purely feed-forward system architecture, when we increase the size of a system instance by adding one component at the end of the chain, we need to verify only the backlog of the new component, as the previous components are not affected by the extension of the system.

For the input event stream S we assume the same upper bound as for S_A in the previous case study, that is $\alpha_S^u = \alpha_{S_A}^u$. In order to speed up the run-times for both the holistic and the modular analysis approach, we do, however, omit the specification of the lower bound α_S^l as described at the end of Sect. 5.1. The resulting TA model for the input generator consists of two UTAs with parameters $N_1^u = 1$, $\delta_1^u = 1$, and $N_2^u = 5$, $\delta_2^u = 7$.

The results of the performance analysis are reported in Table 4. The first row in the table contains the exact values for the worst-case backlogs. These values are determined by means

Table 4 Worst-case backlogs as derived with the different approaches

	Buf_1	Buf_2	Buf_3	Buf_4	Buf_5
TA holistic	5	5	4	4	3
TA modular	5	5	5	4	5
RTC	5	6	6	6	7

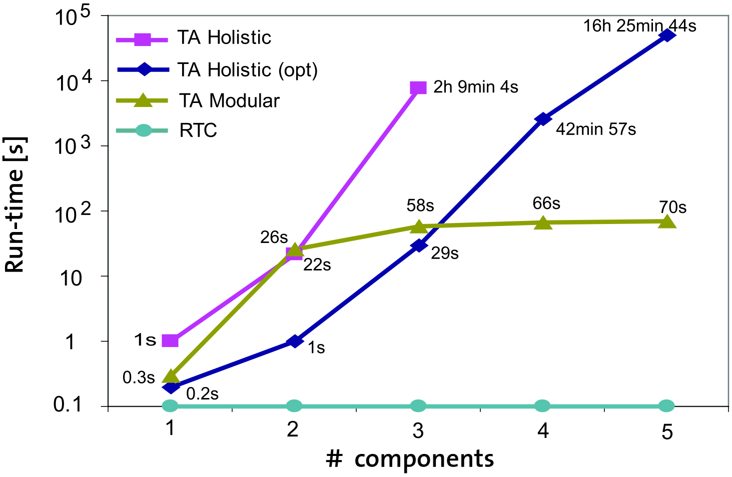


Fig. 14 Computational effort of the modular and the holistic approaches

of holistic TA models for the different system instances. The second row shows the worst-case backlogs as predicted by the modular analysis approach based on TA. The reason for the slightly more conservative results is the same as in the case study of Sect. 5.1: The concave hulls derived as upper bounds for the event streams transmitted between components are an over-approximation of the real streams. In particular, for the sake of efficiency, we decided to represent each input/output stream with a concave pattern of two linear staircase curves only, which is not sufficient to capture the exact behavior of the system. For comparison only, in the last row of Table 4 we report the analysis results achieved by a RTC analysis of the system instances, which is obviously penalized, as the state-based behavior of the components cannot be captured in the RTC models.

Let us now focus on the computational effort required by the considered analysis approaches. Figure 14 displays the run-times of the different methods for the analysis of the five system instances. These run-times are cumulative, meaning that for a system instance with n components they express the total time needed to determine the worst-case backlog values for all n buffers. For the holistic TA analysis we consider two different alternatives for the modeling of the input generator. The first variant uses the staircase-based TA pattern for event generation described in this paper, which in this case corresponds to the combination of two UTAs. The second variant uses an optimized input generator for periodic event streams with jitter/bursts as described in [10]. The chart of Fig. 14 shows a clear trend for the holistic analysis approaches: The run-times increase exponentially with the size of the considered system instance (note the logarithmic scale on the y-axis). This holds for both types of input event generators, the general one based on UTAs and the optimized one designed for periodic streams with jitter/bursts. When we use the general input generator to trigger the holistic TA model, we report a run-time of more than two hours to analyze the first three components. For system instances with more than three components the model checker runs out of memory after several hours of verification. For the optimized input generator the run-times are slightly better with a maximum bearable system size of five components.

Also for the modular TA-based analysis approach the chart of Fig. 14 shows a trend: The run-times increase nearly linearly with the number of considered components. In particular, for each additional component in the chain the run-time increases by roughly 4–30 s. Given the concave hull that describes the input stream of a component, this is the time needed to determine the worst-case backlog of the component and to derive the concave hull that bounds the output stream. The deviations from an exact linear increase pattern can most likely be explained by the varying amount of non-determinism present in the specification of the input streams at the different stages.

The above experiment highlights one of the main advantages of the proposed analysis framework. By adopting appropriate patterns that permit to abstract the input/output interfaces of components, it enables a fully compositional system analysis. In particular, the state-space explosion is limited to the level of isolated components. Consequently, the proposed analysis technique scales to systems of almost arbitrary size, provided that the TA abstractions of the single components are reasonably simple and the representation of the input/output event streams is reasonably coarse.

5.2.2 Non-determinism in event stream specifications

In this second experiment we investigate how sensitive the run-times of the proposed compositional analysis method are with respect to increasing non-determinism in the specification of the input event streams. More precisely, for a simple component modeled as TA, we gradually increase the burstiness of the triggering input event stream, and measure the run-time needed to characterize the output stream of the component.

We consider the component T1/CPU1 from Fig. 10 that implements the load-dependent frequency adaptation described in Sect. 5.1. As input to the component we consider event streams upper bounded by a simple linear staircase curve with step-width $\delta^u = 7$ and seven different levels of burstiness varying from $N^u = 5$ to $N^u = 150$. As for the previous experiment, in order to speed up the verification times, we consider only an upper bound for the input event stream and omit the lower bound. For all seven different input bounds we record the run-time needed by the heuristic described in Sect. 4.3 to characterize the output event stream, where we choose to represent the output bounds as the minimum of two linear staircase functions. In order to ensure that in all seven cases the same number of verification steps is needed to characterize the system output, we set $k := N^u$ in the heuristic.

The results of the experiment are shown in Fig. 15. As can be seen in the chart, the total run-time needed to characterize the output stream increases exponentially with the jitter/burstiness of the input stream. While for the input stream with $N^u = 5$ the derivation of an upper bound for the output event stream is performed in roughly one second, for the input stream with $N^u = 125$ we record a run-time that is three orders of magnitude larger, and for the input stream with $N^u = 150$ the model checker runs out of memory.

The described experiment clearly shows a limitation of TA-based performance analysis: Only event streams with mediocre degree of non-determinism for the timing of event arrivals can be handled with reasonable verification effort. This result is, however, not very surprising, as in general with increasing non-determinism in a TA model the model checker has to explore a larger number of system states.

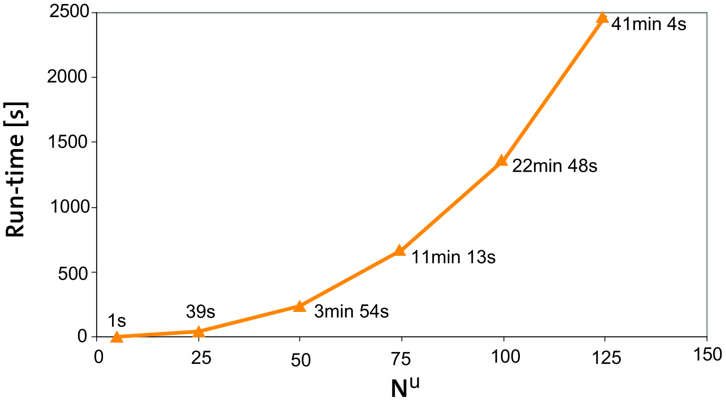
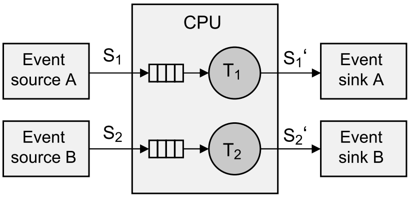


Fig. 15 Total run-time needed to characterize α

Fig. 16 Fixed priority scheduling of two tasks



5.3 Approximation errors

In this final part of the experimental evaluation of the proposed analysis methodology we briefly elaborate on possible approximation errors introduced by bounding the output streams of system components with a convex/concave hull of staircase curves as described in Sect. 4.3. In order to characterize these approximation errors in isolation from other effects, we apply the described TA-based analysis approach to two systems consisting of stateless components only. We compare the obtained bounds with the results of an RTC analysis, which for the considered systems ensures tight results.

Consider first the simple system architecture shown in Fig. 16. The depicted system consists of a CPU that executes two tasks T_1 and T_2. The two tasks are triggered by two strictly periodic streams S_1 and S_2 with periods $p_1 = 60$ ms and $p_2 = 5$ ms, respectively. The CPU schedules the two tasks according to a preemptive fixed priority scheduling policy, where T_1 has higher priority than T_2. We assume that the CPU executes at a constant frequency of 1 GHz and that the execution of T_1 and T_2 takes $60 \cdot 10^6$ and $5 \cdot 10^6$ cycles, respectively. The goal of the analysis is to characterize the output event stream S_2'. For the TA-based analysis of the system we employ a holistic TA model for the preemptive fixed priority scheduling of two tasks, as described in [18].

Figure 17 shows the result for both the RTC analysis and the TA heuristic of Sect. 4.3. The curves $[\alpha'^{l}_{2,\mathrm{RTC}}, \alpha'^{u}_{2,\mathrm{RTC}}]$ (depicted with a solid line in the plot) represent the exact lower and upper arrival curves for the stream S_2' computed by the RTC analysis. The dashed lines in the plot represent the bounds for the output event stream derived by the heuristic, where we decided to represent the upper bound $\alpha'^{u}_{2\mathrm{TA}}$ as the minimum of two linear staircase functions and the lower bound $\alpha'^{l}_{2\mathrm{TA}}$ by one single linear staircase function. As can be seen in the plot, the heuristic clearly over-approximates the real upper bound for S_2'. The reason for

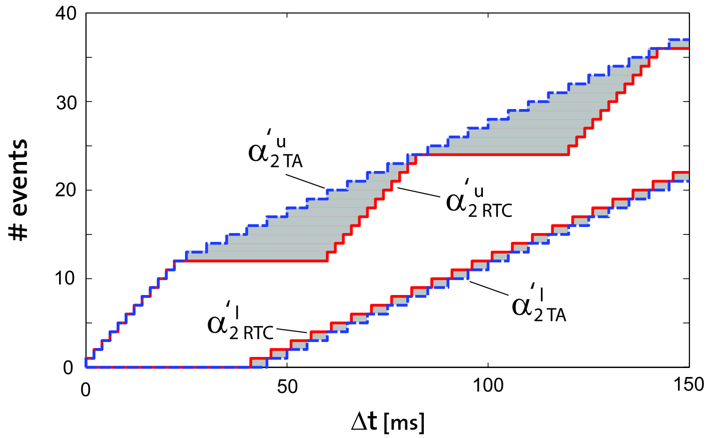


Fig. 17 Bounds for S'_2 determined by RTC (exact) and the TA-heuristic

this approximation error (represented by grey shaded areas in the figure) is that the heuristic constructs only a concave hull of linear staircase functions to upper bound the output stream, whereas the real upper bound of the stream does not have a strictly concave shape. Extending the heuristic of Sect. 4.3 such that it handles such mixed convex/concave output patterns without approximation errors is not trivial and would obviously also considerably slow down the analysis process.

As second experiment for the illustration of approximation errors we consider a simplified version of the component T1/CPU1 from Fig. 10. Assume that instead of the described load-dependent frequency adaptation, CPU1 can arbitrarily change its execution frequency between 166 MHz and 500 MHz. Such a stateless best-case/worst-case component description is ideally suited for an exact RTC analysis of the component. As input for the component we consider the stream S_A as given in Sect. 5.1, that is, a periodic event stream with jitter specified by the parameter triple $p = 7$ ms, $j = 28$ ms, $d = 1$ ms. The goal of the analysis is again to characterize the output event stream of the component. Figure 18 shows the results of both approaches, the RTC analysis and the heuristic of Sect. 4.3. The curves $[\alpha'^{l}_{\mathrm{RTC}}, \alpha'^{u}_{\mathrm{RTC}}]$ represent the exact lower and upper arrival curves for the component output computed by RTC. These curves correspond to a periodic event stream with jitter specified by the parameter triple $p' = 7$ ms, $j' = 32$ ms, $d' = 2$ ms. For the heuristic approach, we decide to represent the upper bound $\alpha'^{u}_{\mathrm{TA}}$ as the minimum of two linear staircase functions and the lower bound $\alpha'^{l}_{\mathrm{TA}}$ by one linear staircase function. The plot shows that the heuristic slightly over-approximates the real upper bound, although the maximum component output follows a concave pattern. Similarly, the lower bound is slightly under-approximated. The reason for this kind of approximation error is that the heuristic described in Sect. 4.3 does not consider horizontal translations of linear staircase functions. In particular, looking at Fig. 18 we see that the offset η^{u}, after which the real upper output bound of the component follows the long-term rate δ^{u}_{2}, is not a multiple of the long-term rate itself. That is, no linear staircase function α'^{u} without horizontal offset will precisely capture the long-term behavior of the component. The reason for the under-approximation of the lower bound is analogous.

Note that in the first part of Sect. 4.2.4 we have described how this kind of approximation error can be avoided when converting known *input* event streams described with PJD parameters to TA input generators. The case of bounding the *output* stream of a TA component

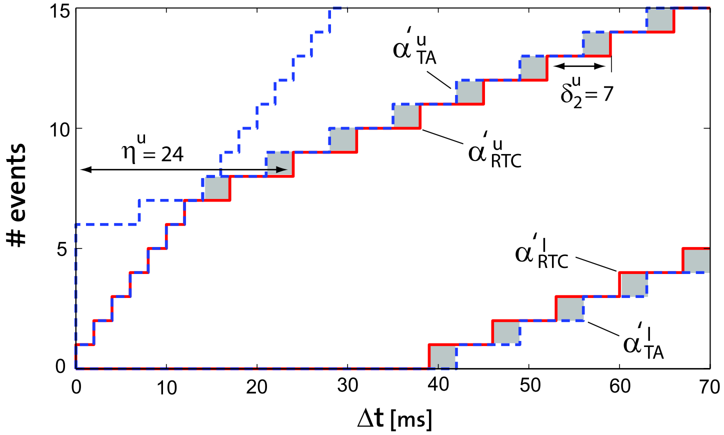


Fig. 18 Output bounds determined by RTC (exact) and the TA-heuristic

is, however, more difficult, as the stream that needs to be bounded is obviously not known a priori. In particular, permitting arbitrary horizontal shifts for the single linear staircase functions would mean adding another degree of freedom for the heuristic and hence considerably slow down the analysis process.

6 Conclusion

In this paper we developed a hybrid analysis methodology that couples analytic (stateless) RTC- and state-based TA analysis. The presented technique is based on the observation that stream abstractions in the form of arrival curves can be obtained by composing individual linear staircase functions by means of minimum and maximum operations. The methodology relies on two different interfaces that were extensively discussed. The input interface converts an arrival curve to a network of TA that triggers a TA component model. The output interface performs the inverse transformation by constructing a tuple of arrival curves for the output of a TA subsystem. In the realization of the input interface each staircase function is guarded by its own TA, where the building of minimum and maximum is implemented by synchronizing groups of TA. In the output interface, the parameters of staircase functions are found by employing observer TA in a binary search based heuristic. For both interfaces correctness is proven, which assures hard performance guarantees.

The proposed methodology limits state space explosion as intrinsic to formal verification to the level of isolated (sub)-components, since loosely coupled TA-based component descriptions can be verified in isolation. For maintaining scalability, we suggest to apply the state-based analysis only to those components, for which an RTC analysis provides overly pessimistic results. As demonstrated by the case study, such cases are found when dealing with components showing state-dependent behavior. Overall, such a strategy will thereby avoid overly conservative performance predictions, but still maintain the scalability of the approach.

As arrival curves represent a more general abstraction for event streams than the widely used PJD (periodic with jitter) event models, the proposed methodology can also directly be applied to couple TA-based timing verification with other analysis tools relying on classical real-time analysis, such as MAST [9] or Symta/S [11].

Lastly, we name some issues that this work leaves open. The heuristic devised for the output interface does not explore shifted staircase curves or non convex/concave patterns. In more general terms, the present work does not assure tightness for the output interface. It does also not consider cycles in the event flow or dependencies among components that require fixed-point iterations in the analysis process. These matters are left for future work.

Acknowledgements This work is funded by the European Union project COMBEST under grant number 215543 and by the Swiss National Science Foundation under grant number 200020-116594.

References

1. Altisen K, Moy M (2010) Arrival curves for real-time calculus: the causality problem and its solutions. In: Esparza J, Majumdar R. (eds) TACAS, pp 358–372
2. Alur R, Dill DL (1990) Automata for modeling real-time systems. In: Paterson M (ed) Proceedings of the 17th international colloquium on automata, languages and programming (ICALP'90). Lecture notes in computer science, vol 443. Springer, Berlin, pp 322–335
3. Behrmann G, David A, Larsen KG (2004) A tutorial on UPPAAL. In: Bernardo M, Corradini F (eds) Formal methods for the design of real-time systems: 4th international school on formal methods for the design of computer, communication, and software systems, SFM-RT 2004. Lecture notes in computer science, vol 3185. Springer, Berlin, pp 200–236
4. Bengtsson J, Yi W (2004) Timed automata: semantics, algorithms and tools. In: Lectures on concurrency and Petri nets. Lecture notes in computer science, vol 3098. Springer, Berlin, pp 87–124
5. Boudec JYL, Thiran P (2001) Network calculus: a theory of deterministic queuing systems for the Internet. Lecture notes in computer science, vol 2050. Springer, Berlin
6. Chakraborty S, Künzli S, Thiele L (2003) A general framework for analyzing system properties in platform-based embedded system designs. Design, automation and test in Europe conference and exhibition, vol 1
7. Chakraborty S, Phan LTX, Thiagarajan PS (2005) Event count automata: a state-based model for stream processing systems. In: Proceedings of the 26th IEEE international real-time systems symposium (RTSS'05), pp 87–98
8. Dierks H, Metzner A, Stierand I (2009) Efficient model-checking for real-time task networks. In: 2nd International conference on embedded software and systems. IEEE Computer Society, Los Alamitos, pp 11–18
9. González Harbour M, Gutiérrez García JJ, Palencia Gutiérrez JC, Drake Moyano JM (2001) Mast: Modeling and analysis suite for real time applications. In: Proceedings of 13th Euromicro conference on real-time systems. IEEE Computer Society, Los Alamitos, pp 125–134
10. Hendriks M, Verhoef M (2006) Timed automata based analysis of embedded system architectures. In: Proceedings of the 20th international parallel and distributed processing symposium (IPDPS 2006). IEEE Press, New York
11. Henia R, Hamann A, Jersak M, Racu R, Richter K, Ernst R (2005) System level performance analysis-the SymTA/S approach. IEEE Proc Comput Digital Tech 152(2):148–166
12. Jonsson B, Perathoner S, Thiele L, Yi W (2008) Cyclic dependencies in modular performance analysis. In: EMSOFT '08: Proceedings of the 8th ACM international conference on embedded software. ACM, New York, pp 179–188. doi:10.1145/1450058.1450083
13. Krcal P, Mokrushin L, Yi W (2007) A tool for compositional analysis of timed systems by abstraction (extended abstract). In: Proceedings of 19th Nordic workshop on programming theory (NWPT07)
14. Künzli S, Hamann A, Ernst R, Thiele L (2007) Combined approach to system level performance analysis of embedded systems. In: Proceedings of the 5th international conference on hardware/software codesign and system synthesis 2007. ACM, New York, pp 63–68
15. Lampka K, Perathoner S, Thiele L (2009) Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In: EMSOFT '09: Proceedings of the seventh ACM international conference on embedded software. ACM, New York, pp 107–116. doi:10.1145/1629335.1629351
16. Modular Performance Analysis Framework and Matlab Toolbox. www.mpa.ethz.ch
17. Norström C, Wall A, Yi W (1999) Timed automata as task models for event-driven systems. In: Proceedings of the 6th international conference on real-time computing systems and applications. IEEE Computer Society, Los Alamitos, p 182

18. Perathoner S, Wandeler E, Thiele L, Hamann A, Schliecker S, Henia R, Racu R, Ernst R, Harbour MG (2007) Influence of different system abstractions on the performance analysis of distributed real-time systems. In: EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on embedded software. ACM, New York, pp 193–202. doi:10.1145/1289927.1289959
19. Phan L, Chakraborty S, Thiagarajan P (2008) A multi-mode real-time calculus. In: Proceedings of the 28th IEEE real-time systems symposium (RTSS 2008). IEEE Computer Society, Los Alamitos, pp 59–69
20. Phan LTX, Chakraborty S, Thiagarajan PS, Thiele L (2007) Composing functional and state-based performance models for analyzing heterogeneous real-time systems. In: Proceedings of the 28th IEEE real-time systems symposium (RTSS 2007). IEEE Computer Society, Los Alamitos, pp 343–352
21. Thiele L, Chakraborty S, Naedele M (2000) Real-time calculus for scheduling hard real-time systems. In: Proceedings of international symposium on circuits and systems, vol 4, pp 101–104
22. The Uppaal timed model checker. www.uppaal.com
23. Wandeler E, Thiele L, Verhoef M, Lieverse P (2006) System architecture evaluation using modular performance analysis: a case study. Int J Soft Tools Technol Transf 8(6):649–667