Real-Time Systems (2019) 55:598–638
https://doi.org/10.1007/s11241-019-09331-1

A comparison of schedulability analysis methods using state and digraph models for the schedulability analysis of synchronous FSMs

Chao Peng[1] · **Haibo Zeng**[2] · **Marco Di Natale**[3]

Published online: 1 March 2019

Abstract
Synchronous reactive models are widely used in the development of embedded software and systems. The schedulability analysis of tasks obtained as the code implementation of synchronous finite state machines (FSMs) can be performed in several ways. One possible option is to leverage the correspondence between the execution of actions in an FSM and the execution of jobs in a digraph task model, thereby applying all the analysis methods developed for these digraph task systems. Another option is to directly leverage the state information and use dynamic programming methods to compute the worst possible sequence of (state dependent) reactions for a given FSM model. In this paper we compare these analysis methods in terms of accuracy and runtime.

Keywords Embedded systems · Synchronous reactive (SR) models · Schedulability analysis · Request/interference bound functions

✉ Haibo Zeng
hbzeng@vt.edu

Chao Peng
pengchao06@gmail.com

Marco Di Natale
marco@sssup.it

1 National University of Defense Technology, Changsha, Hunan, China

2 Virginia Polytechnic Institute and State University, 302 Whittemore (0111), Virginia Tech, Blacksburg, VA 24061, USA

3 Scuola Superiore Sant'Anna, Via Moruzzi, 1, Pisa 56124, Italy

1 Introduction

In the development of embedded controllers, the use of the synchronous reactive (SR) modeling formalism (Lee and Varaiya 2011) is becoming widespread. Examples of tools supporting SR models include SCADE (Esterel Technologies 2014) and Simulink (Mathworks 1994). In this paper, we focus on Simulink models for convenience, but the results apply to other tools.

A Simulink model is a network of blocks. Each block processes a set of input signals and produces a set of output signals. For the purpose of this work, we are interested in the subset of Simulink/Stateflow models that allows the automatic generation of a code implementation, that is, blocks with discrete time input and output signals in a model with a fixed-step solver. Simulink blocks are of two types. *Dataflow* blocks are invariably executed at their periods (which are integer multiples of the system-wide *base period*). Other blocks are Mealy type extended *finite state machines* (FSMs), called *Stateflow* blocks. In Stateflow blocks, each event may cause a state transition and trigger the execution of a set of *actions* (functions defined by designers). *The events that trigger the reaction of the block are obtained from value transitions of periodic signals, and therefore can only occur at the periodic times when these signals can change their values*. If no trigger event is specified, the Stateflow block reacts according to the model base rate. Hence, each Simulink block, when executed, computes two functions: the *state update* function (possibly omitted in Dataflow blocks), which updates the next block state based on the current state and the values of the input signals, and the *output update* function, computing the new values for the output signals as a function of the current state and the inputs.

When implementing a Simulink model (Natale et al. 2010; Zeng and Di Natale 2011) and in particular those containing Stateflow blocks (Zhao et al. 2017), the implementation should be feasible (e.g., with respect to time and memory constraints) while preserving the logical-time execution semantics (the rate and order of execution for the blocks and the communication flows). The commercial code generation tool Embedded Coder/Simulink Coder (Mathworks 1994) automatically generates a semantics-preserving implementation, which assumes a single periodic task implementation for each Stateflow block code, scheduled with fixed priority. The period of the task is the greatest common divisor of the periods of the trigger signals. Each time the task is activated, it checks for any active trigger and, if there is any, processes them.

In this work, we assume the existence of a semantics-preserving implementation where each synchronous FSM is realized as *a single task scheduled by static priority*. This assumption matches the common implementation of commercial code generation tools, and we focus on the problem of system schedulability that concerns whether each task meets its deadline. However, a multi-task implementation of a Stateflow block can provide more flexibility and efficiency in terms of schedulability (Di Natale and Zeng 2012; Zhu et al. 2013).

In our previous work (Zeng and Natale 2012), we outlined the possible correspondence of the execution of reactions in a synchronous FSM to the execution of jobs in a digraph task model. In Sect. 4 of this paper, we elaborate on the transformation

of FSM to the digraph task model (Stigge et al. 2011) and summarize the possible analysis options.

Digraph tasks are a very general task model for which schedulability analysis is still tractable (pseudo-polynomial time) for bounded-utilization systems with earliest deadline first (EDF) scheduling (Stigge et al. 2011). The exact analysis under *static priority scheduling* is shown to be strongly coNP-hard (Stigge and Yi 2012). This is because of the need to check the combination of job releases (paths in the task graph) from different tasks. A combinatorial abstract refinement technique (Stigge and Yi 2013, 2015a) is proposed to significantly speed up the exact analysis by iteratively refining the abstraction of paths. Approximate analysis is presented in (Guan et al. 2014), with pseudo-polynomial complexity and bounded speedup factors. Also, to improve the efficiency of calculating the request and demand bound functions for digraph tasks, results from max-plus algebra are leveraged to demonstrate the linear periodicity of the two functions (Zeng and Di Natale 2013, 2015; Peng and Zeng 2018).

A different approach consists in the explicit consideration of state dependencies and the event activation offsets. In this case, the analysis are performed by exploring the possible state traversals in the hyperperiod of the trigger events, and considering all the possible busy periods defined by the event activation offsets. Knowledge of the activation offsets allows to reduce the pessimism with respect to the existing offset-free model of digraph task analysis.

1.1 Our contribution and paper outline

In this paper, we present the analysis framework for calculating the exact and approximate response times of synchronous finite state machine models through a digraph model abstraction, or directly, through state analysis. As synchronous FSMs are triggered by synchronized periodic events, unlike the analysis in Stigge et al. (2011) where task digraphs are assumed to have arbitrary offsets, our analysis considers periodic tasks with synchronized offsets.

We present the semantics of synchronous finite state machines in Sect. 2, followed by a short introduction to max-plus algebra, which provides the mathematical foundation that is used to speed up the computation of the execution time requests in a given time interval by a given FSM. Next, we present the methods for transforming the FSM into a digraph model Sect. 4. We also discuss analysis methods for the transformed digraph tasks, including techniques for speeding up the computation by leveraging the periodicity of the request and interference bound functions. In Sect. 5 we introduce an analysis method based on the explicit consideration of the state dependencies and the consideration of the activation event arrival times (and offsets). In both cases, the analysis method is based on an abstraction of execution time matrix that allows to leverage periodicity in the request and interference bound functions and to compute them for large time intervals by exploiting dynamic programming. In the case of state-based analysis, the execution matrix is defined by the state space, instead of the size of the task digraph, giving rise to interesting questions about the relative accuracy and speed of the two methods. To settle this issue, in Sect. 6, we use randomly generated

systems of synchronous finite state machines to evaluate these analysis techniques and compare their accuracy and runtime.

2 Synchronous finite state machines

The synchronous execution model considered in this paper consists of a set of mealy finite state machines (FSMs). Each FSM block is characterized by a set of input signals, a set of trigger events, and a set of output signals. The internal behavior of a Simulink Stateflow block follows the semantics and notation of extended (hierarchical and concurrent) state machines (Harel 1987).

Following the original Statecharts specification (Harel 1987), the actual Stateflow semantics also allow *concurrent states*, *superstates*, *entry* actions, *exit* actions, *while* actions, *join transitions*, and others. Also, in Stateflow (as in most extended FSM formalisms), transitions can be triggered by a logical combination of events. An FSM with hierarchical superstates with AND/OR composition can be transformed into an equivalent flat FSM whose states are obtained from the set product of the states of the composed machines. Examples can be found in the seminal paper (Harel 1987), and the procedure is defined in Lee and Varaiya (2011) for parallel and series compositions. Of course, the resulting number of states can be very high and a general type of transformation may not exist when all the other Stateflow semantic extensions are considered. For simplicity, in this paper we assume *standard (flat) FSMs, in which each transition is associated with a single event*.

Formally, an FSM is defined by a 7-tuple $\mathcal{F} = (\mathbb{S}, s_\alpha, \mathbb{I}, \mathbb{O}, \mathbb{E}, \mathbb{A}, \mathbb{V})$, where $\mathbb{S} = \{s_1, s_2, \ldots s_{|\mathbb{S}|}\}$ is the set of *states*, $s_\alpha \in \mathbb{S}$ is the *initial state*, $\mathbb{I} = \{i_1, i_2, \ldots i_{|\mathbb{I}|}\}$ is the set of *input* signals, $\mathbb{O} = \{o_1, o_2, \ldots o_{|\mathbb{O}|}\}$ is the set of *output* signals, and $\mathbb{V}$ is the set of internal variables. Each i_j (o_j) is also denoted as α_j (as their direction is not relevant in this paper). Each signal α_j is associated with a period T_{α_j} that must be an integer multiple of the system *base period* T_b, i.e., $T_{\alpha_j} = k_{\alpha_j} \cdot T_b$ where k_{α_j} is an integer. Signal values only change at multiples of its period and are persistent between updates. In this sense, α_j is a function defined over a discrete time domain that is consistent with T_b.

$\mathbb{E}$ is the set of *activation or trigger events*. Each event e_j occurs at (rising or falling) edges of a signal α_j, and therefore appears only at time instants belonging to a time base with period $T_{e_j} = T_{\alpha_j}$, an integer multiple of T_b. At each time $i \cdot T_{e_j}$ the event may be *present* (if there is an edge on the signal) or *absent* (otherwise). Fig. 1 shows an example following the Stateflow graphical notation for the input/output view of an FSM, i.e., denoted with inputs, outputs, and trigger event signals.

To better define the internal behavior at each instant $i \cdot T_b$, even when no event is present, the notion of *stuttering behavior* is conventionally added (Lee and Varaiya 2011 is a textbook describing stuttering as well as some of the composition rules for transforming hierarchical state machines to flat FSMs). Formally, an *absent event*, denoted as $\perp$, is added to the set of input events and assumed as present at all multiples of the base period when no event is present. The reaction of the Stateflow chart to the absent event is the following: the state does not change, neither do the output signal values.

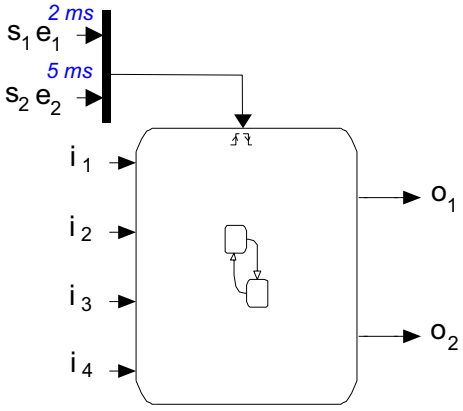


Fig. 1 An example of FSM input/output view in Stateflow. The events are computed from (periodic) signals

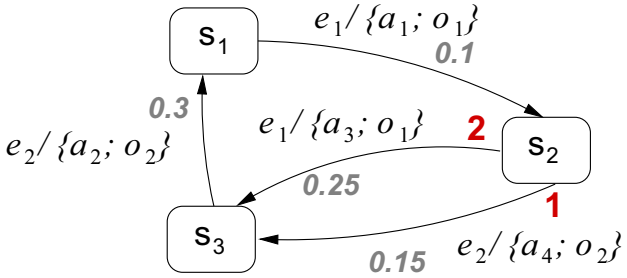


Fig. 2 An example of FSM. The guard conditions are empty thus are omitted

$\mathbb{A}$ is the set of transitions. Each transition $\theta_j \in \mathbb{A}$ consists of a 6-tuple $\theta_j = (\text{src}(\theta_j), \text{snk}(\theta_j), e(\theta_j), g_j, a_j, p_j)$, where a_j is the *action*, $\text{src}(\theta_j)$ is the source state, $\text{snk}(\theta_j)$ is the sink state, $e(\theta_j) \in \mathbb{E}$ is the trigger event, g_j is the guard condition (an expression of the input and output signals and internal variable values), and p_j is the evaluation order. The evaluation order determines the order by which the conditions for the transitions are evaluated (according to event presence and guard conditions), and consequently which transition should be taken when two or more transitions can be taken out of a state at the same time instant. The event triggering a specific action a_j may also be labeled as $e(a_j)$.

Because of guards there may be cases in which, despite an event is present, the transition is not taken. However, events are already conditionally generated based on signals, and guards have no safety effect on worst case timing analysis. In our paper, guards are included for a better match to real-world FSM models (Stateflow).

For timing analysis, we assume each action a_j is characterized by its worst case execution time (WCET) C_{a_j} (Please refer to Wilhelm et al. 2008 for a survey of WCET techniques). The hyperperiod of an FSM $H_{\mathcal{F}}$ is the least common multiple of all the periods of its events. The system hyperperiod H is defined as the least common multiple of all the (hyper)periods of the blocks (of type Dataflow or Stateflow).

Figure 2 shows an example of the notation used to describe the states and transitions, along with the event, guard condition, and action associated to each transition. The

execution time (in grey) and execution order (red/dark, near the source state) of each transition are also denoted in the figure.

The evaluation order associated with transitions makes the FSM behavior deterministic. For example, in the FSM of Fig. 2, if events e_1 and e_2 have periods 2 ms and 5 ms, they will occur *simultaneously* every 10 ms. In the following, we use ms as the default time unit, unless otherwise specified. If the system is in state s_2, the order associated with the outgoing transitions indicates that the transition with action a_4 (order 1) will be taken, not the transition with action a_3 (order 2).

In synchronous FSMs all events occur with periods that are multiples of the base period and with the same phase. Therefore, sets of events arrive at exactly the same time (hence the name *synchronous* FSMs). Also, the task implementing an FSM is checked for overruns at the beginning of its execution. Hence, every reaction must complete before the next trigger event arrives (i.e., with *constrained deadline*). We use $\mathbb{T}_{\mathcal{F}}$ to denote the *ordered* set of time instants that are integer multiples of event periods of FSM $\mathcal{F}$

$$\forall t \in \mathbb{T}_{\mathcal{F}}, \quad \exists e_i \in \mathbb{E} \text{ and } j \in \mathbb{N} \text{ such that } t = j \times T_{e_i}$$

where $\mathbb{N}$ is the set of non-negative integers. By the semantics of Stateflow, the release times and absolute deadlines of $\mathcal{F}$ are members of the set $\mathbb{T}_{\mathcal{F}}$.

When an FSM is translated into code, for each possible state and input signal, the task implementing the reactions must define the *state update* and *output update* functions with the corresponding *actions* (the two functions are sometimes merged). *In this paper, we assume a single-task implementation for each FSM, and the terms FSM and task are used interchangeably*. Multi-task implementations are possible (Di Natale and Zeng 2012; Zhu et al. 2013) and, while we believe our analysis can be easily extended to such implementations, the formal proof is beyond the scope of this work and left to future extensions.

In the following section, we introduce max-plus algebra and the matrix calculus on it. Max-plus algebra is used in the following sections as a computation tool, to speed up the computation of the execution time requested by (higher priority) tasks and the evaluation of the schedulability conditions.

3 Max-plus algebra and periodicity on matrix power sequences

A max-plus algebra (Baccelli et al. 1992) is defined over the domain $\mathbb{R}^* = \mathbb{R} \bigcup \{-\infty\}$, with operations maximum (denoted by the max operator $\oplus$) and addition (denoted by the plus operator $\otimes$) defined as

$$x \oplus y = \max(x, y), \quad x \otimes y = x + y \tag{1}$$

It is easy to verify that $-\infty$ is neutral with respect to $\oplus$, i.e. $x \oplus (-\infty) = x, \forall x \in \mathbb{R}^*$. Likewise, 0 is neutral with respect to $\otimes$, $x \otimes 0 = x, \forall x \in \mathbb{R}^*$.

Similar to traditional algebra, in max-plus algebra, a matrix $\mathbf{X}$ and its elements $x_{i,j}$ are defined over the domain $\mathbb{R}^*$. The matrix operations over $\mathbb{R}^*$ are defined in the same

way as the matrix operation over any field. For example, $\mathbf{Z} = \mathbf{X} \oplus \mathbf{Y}$ is defined by taking the maximum operation of the corresponding elements of the matrices $\mathbf{X}$ and $\mathbf{Y}$, i.e. $z_{i,j} = x_{i,j} \oplus y_{i,j}$.

For matrices $\mathbf{X} \in \mathbb{R}^*(m,k)$ and $\mathbf{Y} \in \mathbb{R}^*(k,n)$, the result of the multiplication is a matrix $\mathbf{Z} \in \mathbb{R}^*(m,n)$, where its elements are

$$z_{i,j} = (x_{i,1} \otimes y_{1,j}) \oplus \cdots \oplus (x_{i,k} \otimes y_{k,j}) = \max_{l=1}^{k}(x_{i,l} + y_{l,j}) \tag{2}$$

The k-th power of a square matrix $\mathbf{X} \in \mathbb{R}^*(n,n)$ is iteratively defined by $\mathbf{X}^{(k)} = \mathbf{X}^{(l)} \otimes \mathbf{X}^{(k-l)}$ (max-plus multiplication) with $\mathbf{X}^{(1)} = \mathbf{X}$.

In the remaining of the section and the following two sections, we summarize results on how the powers of max-plus matrices can be used to compute the worst case request/interference functions from a set of actions for a given time interval. Since the length of the time interval corresponds to the power of the matrix, finding efficient ways to compute these matrices is very important. The following definitions and results provide insight on how to partition the computation of the elements of matrix powers with a large exponent into a recurrent part plus some possible initial terms.

Definition 1 A sequence $x^* = (x^{(r)})$, $r \in \mathbb{N}^+$ is defined as *almost generally periodic* if there exists a pair of integers d and p and a set of numbers $q(k) \in \mathbb{R}^*$, $k = 1, \ldots, p$ such that

$$\forall k = 1, \ldots, p, \forall r > d, r \equiv k \bmod p, \quad x^{(r+p)} = x^{(r)} + p \times q(k)$$

The smallest number p (d) with the above properties is called the *generalized period (generalized defect)* of x^*, denoted as $p = gper(x^*)$ ($d = gdef(x^*)$). q is called the *generalized factor* of x^*, denoted as $q = gfac(x^*)$.

Definition 2 The matrix $\mathbf{X} = (x_{i,j})$ is defined as *almost generally periodic* if for each element $x_{i,j}$ in its power sequence $\mathbf{X}^* = (X^{(r)})$, $r \in \mathbb{N}^+$ the sequence $x^*_{i,j}$ is almost generally periodic. The matrix $gfac(\mathbf{X}^*) = (gfac(x^*_{i,j}))$ is called the generalized factor matrix of $\mathbf{X}$, the number $gdef(\mathbf{X}) = \max gdef(x^*_{i,j})$ is called its generalized defect, and $gper(\mathbf{X}) = lcm\{gper(x^*_{i,j})\}$ is its generalized period.

The following theorem states that the periodicity property is applicable for every matrix.

Theorem 1 (Molnárová 2005; Zeng and Natale 2012) *Every matrix over max-plus algebra (hence the execution request matrix of any FSM) is almost generally periodic.*

In Molnárová (2005) shows that the problem of computing the $gper$ and $gfac$ terms is NP-hard by providing a polynomial-time transformation from a known NP-complete problem to this computation problem. Nevertheless, the complexity is a function of the size of the matrix, but is asymptotically independent from the power of the matrix. Fortunately, with respect to irreducible matrices (whose corresponding

digraphs are strongly connected, i.e., any node of the digraph can be reached from any other one), there are polynomial-time algorithms for calculating an upper bound of the generalized defect and the generalized period and factor. More specifically, for irreducible matrices, Hartmann and Arguelles (1999) derive an upper bound on the generalized defect and develop an $O(n^3)$ algorithm to compute it. Gavalec (2000) gives an $O(n^3)$ algorithm for computing the generalized period and Young et al. (1991) present an $O(nm \log n)$ algorithm for the generalized factor where m is the edge number of the corresponding digraphs. Moreover, Zeng and Di Natale (2013, 2015) discuss the periodicity property of (irreducible or reducible) matrices and the corresponding digraphs.

4 Analysis using digraph task model

In this section, we discuss the schedulability analysis of FSMs by leveraging existing results on digraph task models. Following the traditional real-time schedulability analysis, the work on digraph tasks considers the execution of FSM reactions in the time domain, identified as tasks or jobs. The digraph task model is not state-oriented but represents the atomic units of execution and their activation assumptions or execution dependencies.

Along this line of research, a series of graph-based models with different expressiveness and analysis complexity is proposed. A classification of the task models that are typically used in schedulability analysis can be attempted based on the concept of *task graph*. A task in the model is represented by a graph, where the vertices represent different kinds of jobs, and the edges are the possible flows of control. Each vertex (or type of job) is characterized by its WCET and relative deadline. Each edge is labeled with the minimum separation time between the release of the two vertices it connects.

Among the proposed task graph models, the digraph model (Stigge et al. 2011) allows arbitrary directed graphs (hence arbitrary cycles) to represent the release structure of jobs, which significantly increases the expressiveness. The extended digraph model (Stigge and Yi 2015b) adds global minimum inter-release constraints between any two vertices, including those that have no connecting edge (or functional dependency). An extended digraph task can be transformed into a plain digraph task with the same schedulability property (Stigge and Yi 2015b). The model of timed automata with tasks (Norström et al. 1999) is a generalization of all the above models, with complex dependencies between job release times and task synchronization. However, its schedulability analysis is very expensive and even undecidable in certain variants of the model (Fersman et al. 2007).

We refer the readers to a recent survey (Stigge and Yi 2015b) for a discussion on the expressiveness and complexity of schedulability analysis for task graph models. In the following, we focus our attention to the schedulability of digraph task model since it is expressive enough to capture the structure of synchronous finite state machines. However, the existing analysis techniques summarized in this section are all inaccurate, as they are developed for systems with arbitrary offsets. On the other hand, synchronous FSMs are better analyzed with techniques for static offsets, as in the next section.

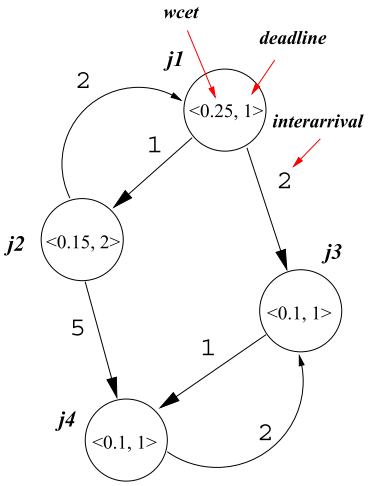


Fig. 3 Notations for a real-time task digraph

4.1 Digraph task models and schedulability analysis

We first introduce the formal definition of the digraph task model (from Stigge et al. 2011).

Definition 3 A *digraph task* τ is a directed graph $\mathcal{D}(\tau) = (\mathbb{V}, \mathbb{E})$ where the *vertices* $\mathbb{V} = \{v_1, v_2, \ldots\}$ represent the types of jobs that can be released for τ, and the *edges* $\mathbb{E}$ ($\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$) represent possible flows of control, i.e., the release order of the jobs of τ. Each vertex $v_i \in \mathbb{V}$ (or type of job) is characterized by an ordered pair $\langle e(v_i), d(v_i) \rangle$, where $e(v_i)$ and $d(v_i)$ denote its WCET and relative deadline, respectively. Each edge $(v_i, v_j) \in \mathbb{E}$ is labeled with a parameter $p(v_i, v_j)$ that denotes the minimum separation time between the releases of v_i and v_j.

A *path* $\pi \in \mathcal{D}(\tau)$ is a sequence of vertices $(v_1, v_2, \ldots, v_l)$ ($v_i \in \mathbb{V}$) where each (v_i, v_{i+1}) is an edge in $\mathbb{E}$. In this work, we assume the digraph task with *constrained deadlines* (Stigge et al. 2011; Guan et al. 2014), i.e., $\forall (v_i, v_j) \in \mathbb{E},\ d(v_i) \leq p(v_i, v_j)$. Hence, for each edge, the absolute deadline of a job is no larger than the release time of the subsequent job.

Figure 3 shows an example task digraph, which has constrained deadlines. We briefly summarize the schedulability analysis techniques of digraph task models (Please refer to Stigge and Yi 2013; Guan et al. 2014; Stigge and Yi 2015a; Zeng and Di Natale 2015; Peng and Zeng 2018 for details on these techniques). To formally analyze the schedulability of a digraph task system *under static priority*, we recapture the concepts of request and interference functions to abstract the execution of a path.

Definition 4 (*Stigge and Yi* 2013, 2015a; *Guan et al.* 2014) For a digraph task τ, the maximal *cumulative execution request* from a path $\pi = (v_1, \ldots, v_l)$ in $\mathcal{D}(\tau)$ within any time interval of length t is defined as its *request function* (RF) $\tau.rf_\pi(t)$, i.e.,

$$\tau.rf_\pi(t) = \max\{e(\pi') | \pi' \text{ is a prefix of } \pi \text{ and } p(\pi') < t\}$$

where $e(\pi) = \sum_{i=1}^{l} e(v_i)$ and $p(\pi) = \sum_{i=1}^{l-1} p(v_i, v_{i+1})$.

Definition 5 (*Guan et al.* 2014) For a digraph task τ, the maximal amount of *executed load* from a path $\pi = (v_1, \ldots, v_l)$ in $\mathcal{D}(\tau)$ within any time interval of length t is defined as its *interference function* (IF) $\tau.if_\pi(t)$, i.e.,

$$\tau.if_\pi(t) = \max\{ee(\pi')|\pi' \text{ is a prefix of } \pi \text{ and } p(\pi') < t\}$$

where $ee(\pi) = \sum_{i=1}^{l-1} e(v_i) + \min(e(v_l), t - p(\pi))$ and $p(\pi) = \sum_{i=1}^{l-1} p(v_i, v_{i+1})$.

The request and interference functions are used to calculate the exact response time for digraph tasks with *arbitrary offsets*. For any vertex v with its interfering digraph task set Γ (the set of digraph tasks with higher priority than the digraph task of v), its exact response time can be expressed by Guan et al. (2014)

$$R(v, \mathbb{D}(\Gamma)) = \max_{\sigma \in \mathbb{D}(\Gamma)} \left\{ \min_{t>0} \left\{ t | e(v) + \sum_{\pi \in \sigma} rf_\pi(t) \leq t \right\} \right\} \tag{3}$$

where $\mathbb{D}(\Gamma)$ denotes the set of all possible path combinations (one path from each digraph task in Γ), and σ denotes one such path combination in $\mathbb{D}(\Gamma)$. A digraph task is schedulable *if and only if* all its vertices v satisfy $R(v, \Gamma) \leq d(v)$. As an alternative, we can replace RF with IF in Eq. (3) to calculate the exact response time (Guan et al. 2014). This schedulability condition is necessary and sufficient (or *exact*) for digraph tasks with *arbitrary offsets*, but its verification is extremely complicated since it requires to check all the possible path combinations from higher priority digraph tasks.

The concepts of *request bound function* (RBF) and *interference bound function* (IBF) are introduced to avoid the path combination problem in the exact analysis, as each digraph task is characterized with a single RBF or IBF function (as opposed to one for each possible combination of action paths in the exact analysis) (Guan et al. 2014).

Definition 6 (*Baruah* 2003; *Guan et al.* 2014; *Zeng and Natale* 2015) For a digraph task τ, the maximum *cumulative execution time request* by its jobs that have their release times within any time interval of length t is defined as its *request bound function* $\tau.rbf(t)$, formally $\tau.rbf(t) = \max_{\pi \in \mathcal{D}(\tau)} rf_\pi(t)$.

Definition 7 (*Guan et al.* 2014) For a digraph task τ, the maximum amount of *executed load* by its jobs that have their release times within any time interval of length t is defined as its *interference bound function* $\tau.ibf(t)$, formally $\tau.ibf(t) = \max_{\pi \in \mathcal{D}(\tau)} if_\pi(t)$.

Similarly, the approximate response times for a vertex v with an interference digraph task set Γ can be expressed by

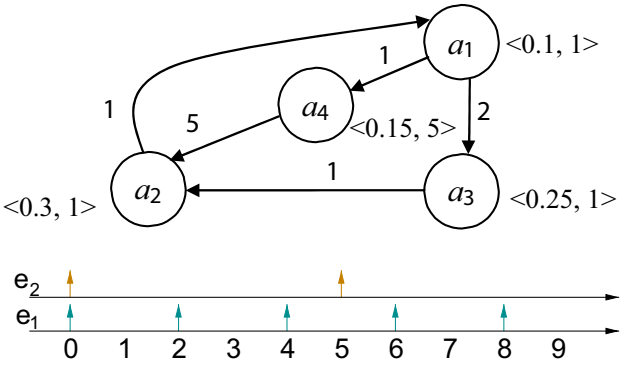


Fig. 4 A digraph task model for the FSM in Fig. 2 and the event arrivals in the hyperperiod

$$\begin{cases} R_{RBF}(v,\Gamma) = \min\limits_{t>0}\left\{t \mid e(v) + \sum\limits_{\tau\in\Gamma} \tau.rbf(t) \le t\right\} \\ R_{IBF}(v,\Gamma) = \min\limits_{t>0}\left\{t \mid e(v) + \sum\limits_{\tau\in\Gamma} \tau.ibf(t) \le t\right\} \end{cases} \tag{4}$$

Hence, a digraph system is schedulable *if* all its vertices v satisfy $R_{RBF}(v,\Gamma) \le d(v)$ (or $R_{IBF}(v,\Gamma) \le d(v)$). The condition is sufficient but not necessary. We adopt the notations from Guan et al. (2014): If the response time calculated by method A (denoted as R_A) is always upper bounded by the one derived from method B (denoted as R_B), i.e., $R_A \le R_B$, we say that A dominates B, written as A $\succcurlyeq$ B. If R_A and R_B are always the same, we denote the equivalent relation as A = B. These analysis methods have the following property (Guan et al. 2014):

$$\begin{array}{ccc} \mathsf{IF} & \succcurlyeq & \mathsf{IBF} \\ \| & & \curlyvee \\ \mathsf{RF} & \succcurlyeq & \mathsf{RBF} \end{array}$$

In the following, we discuss how to construct the corresponding digraph task for an FSM. In synchronous FSMs, each action is characterized by its WCET. The structure of the FSM and its activation events constrain the possible sequences of action executions. Two representations are possible: one with actions (i.e., to treat each action as a job in the digraph); the other with action instances (i.e., each action instance in the hyperperiod is a job). Without loss of generality, *all event offsets in the FSM are assumed to be 0.*

4.2 Modeling the FSM as a digraph with actions

We use the FSM in Fig. 2 as an example. Fig. 4 shows the event stream pattern in the first hyperperiod, and the corresponding digraph task model based on actions.

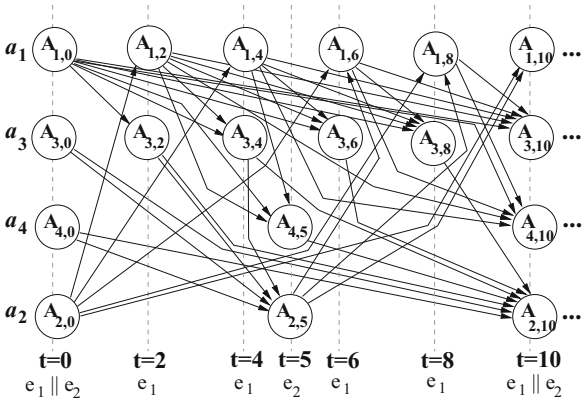


Fig. 5 A more accurate digraph task model for the FSM in Fig. 2

The minimum inter-arrival time between e_1 and e_2 is 1, the greatest common divisor of the periods of the two events. Thus, the edges (a_1, a_4), (a_2, a_1), and (a_3, a_2) are labeled with 1. a_1 and a_3 are both triggered by e_1, thus the edge (a_1, a_3) is labeled with 2, the period of e_1. Similarly, (a_4, a_2) is labeled with 5, the period of e_2. Because of the synchronous assumption of SR models, the deadline of an action is defined as the minimum distance to the next trigger event, which is equal to the minimum label among all the outgoing edges. For example, the ⟨WCET, deadline⟩ pair for vertex a_2 is $\langle 0.3, 1\rangle$.

4.3 Modeling the FSM as a digraph with action instances

The task digraph in Fig. 4 is a pessimistic approximation of the FSM action activation model, as the periodic pattern of event streams is not adequately captured. For example, consider the three actions a_2, a_1, and a_4 activated at time 0, 1, and 2 respectively, which implies that the event sequence e_2, e_1, e_2 occurs at the times 0, 1, 2. This is clearly impossible considering the event pattern in Fig. 4.

A more accurate task model can be obtained by identifying each action occurrence according to the time when its trigger events happen in the hyperperiod. For each action a_i, we generate an infinite sequence of vertices in a digraph model, where each vertex $A_{i,t_{i,k}}$ corresponds to the instance of a_i triggered by an event at time $t_{i,k}$, i.e., it represents an *action instance* $(a_i, t_{i,k})$. We denote it as $A_{i,t_{i,k}} = (a_i, t_{i,k})$ and *use them interchangeably*. As its trigger event $e(a_i)$ is periodic, $t_{i,k} = k \times T_{e(a_i)}, \forall k \in \mathbb{N}$. For example, in Fig. 5 illustrating the more accurate digraph task model for the FSM in Fig. 2, there will be a set of vertices $A_{2,t_{2,k}}$, $t_{2,k} = 0, 5, 10, \ldots$ representing the possible executions of a_2 at times $t_{2,k} = 5 \times k$. The next action following a_2 is a_1. Since events may also be inactive (the machine may stutter), following $A_{2,0}$ (action a_2 executed at time 0), action a_1 could then be executed at 2, 4, 6, and so on. Thus, in this task graph there should be an edge from $A_{2,0}$ to $A_{1,j}$ for $j = 2, 4, 6, \ldots$ Generally, for each $A_{i,t_{i,k}}$, the action a_{i+1} possibly following a_i can be triggered by the event $e(a_{i+1})$ at time $t_{i+1} = \left(k + \left\lfloor \frac{t_i}{T_{e(a_{i+1})}} \right\rfloor\right) \times T_{e(a_{i+1})}$, for all positive integers k ($\forall k \in \mathbb{N}^+$).

The digraph model with action instances is shown in Fig. 5. The minimum inter-arrival time label on each edge $(A_{i,k} - A_{j,l})$ is omitted, but can be easily computed as $(l - k)$. Since there is an infinite number of possible action instances, schedulability analysis requires the reduction of the digraph.

The digraph model can be simplified by removing the edges that are not critical to the schedulability analysis, and by folding vertices according to the periodic pattern of the event arrivals in the hyperperiod. We first introduce the concept of *tight* and *loose* edges, and prove that the removal of loose edges does not affect the schedulability analysis.

Definition 8 An edge $(A_{i,t_{i,k}}, A_{j,t_{j,q}})$ in the digraph model is *tight* if there is no other edge $(A_{i,t_{i,k}}, A_{j,t_{j,p}})$ in the model such that $t_{i,k} < t_{j,p} < t_{j,q}$ (with $p < q$). This means that $t_{j,q}$ is the earliest time a_j can be triggered following a_i at time $t_{i,k}$. All other edges are defined as *loose*.

Definition 9 An *action sequence* σ is defined as a sequence of action instances. In each action instance $A_{i,t_i} = (a_i, t_i)$ a_i is an action in the FSM, and t_i is the time event $e(a_i)$ occurs. For notational convenience, we assume the action instances in sequence σ are indexed incrementally from 1, i.e., $\sigma = [A_{1,t_1}, \ldots]$.

Definition 10 An action sequence $\sigma = [A_{1,t_1}, \ldots]$ is *legal* if

- $\text{snk}(a_i) = \text{src}(a_{i+1})$; and
- a_{i+1} is triggered by $e(a_{i+1})$ at time $t_{i+1} > t_i$, i.e., $t_{i+1} = (k + \lfloor \frac{t_i}{T_{e(a_{i+1})}} \rfloor) \times T_{e(a_{i+1})}$, $\forall k = \mathbb{N}^+$.

A legal action sequence $\sigma = [A_{1,t_1}, \ldots]$ is a path in which a *tight* edge $(A_{i,t_i}, A_{i+1,t_{i+1}})$ connects any two consecutive action instances (a_i, t_i) and (a_{i+1}, t_{i+1}).

As mentioned, the schedulability analysis relies on the quantification of functions like request (bound) function. We now demonstrate how the digraph can be reduced by removing all the loose edges without affecting the exact and approximate schedulability analysis. We first introduce the dominance relationship (Stigge and Yi 2013, 2015a) for two paths.

Definition 11 Given two paths π and π', π dominates π', denoted as $\pi \succcurlyeq \pi'$, if and only if

$$\forall t \geq 0, rf_{\pi}(t) \geq rf_{\pi'}(t)$$

or, in an equivalent way

$$\forall t \geq 0, if_{\pi}(t) \geq if_{\pi'}(t)$$

A path π is a *critical path*, if there exists no other path π' such that $\pi' \succcurlyeq \pi$.

In Stigge and Yi (2013, 2015a) the authors demonstrate that for the exact schedulability analysis we only need to focus on these critical paths (or critical request functions). For any path π including some loose edges, we demonstrate that there

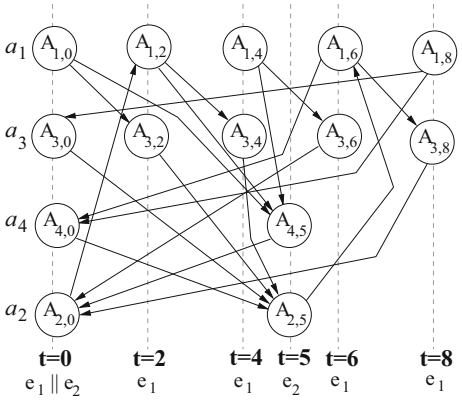


Fig. 6 The simplified digraph task model of Fig. 5

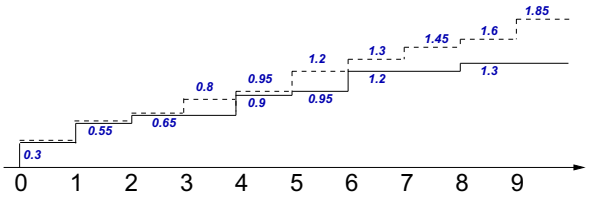


Fig. 7 Request bound functions of the digraph models in Fig. 4 (dotted line) and in Fig. 6 (solid line)

always exists another path π' constructed by replacing the loose edges with the corresponding tight edges, e.g., $\pi = (A_{1,0}, A_{3,4}, A_{2,5})$ and $\pi' = (A_{1,0}, A_{3,2}, A_{2,5})$ in Fig. 5, such that $\pi' \succcurlyeq \pi$. Hence, any path π with loose edges is not critical and removing all the loose edges is safe for the purpose of schedulability analysis. In addition, since the RBF (IBF) function searches the maximum value among the request (interference) functions of all the paths, it does not change when all the loose edges in the digraph task model are removed.

The digraph obtained after removing all loose edges repeats every hyperperiod. Thus, it is sufficient to reason about the possible transitions and actions triggered by the events in one hyperperiod. For each event, we generate a set of vertices in the digraph, each vertex corresponds to the action triggered by the specific event. In general, there are $\sum_{a_i} \frac{H_{\mathcal{F}}}{T_{e(a_i)}}$ vertices in the digraph model. For example, for the two possible arrivals of event e_2 at time 0 and 5, we create two vertices $A_{2,0}$ and $A_{2,5}$ to represent the possible executions of a_2 within the hyperperiod ($H_{\mathcal{F}} = 10$ in the example). Figure 6 shows the reduced digraph model for the FSM $\mathcal{F}$ in Fig. 2. In the figure, the back edges to $A_{2,0}$ and $A_{4,0}$ are clearly connected to instances in the next hyperperiod. The minimum inter-arrival time labeled on edge $(A_{i,k}, A_{j,l})$ is $(l - k)$ mod $H_{\mathcal{F}}$.

4.4 Comparisons on the RBF, IBF, and DBF

The action instance digraph in Fig. 6 can give more accurate RBF/IBF than the action-based graph in Fig. 4, as shown in Figs. 7 and 8 respectively. For example,

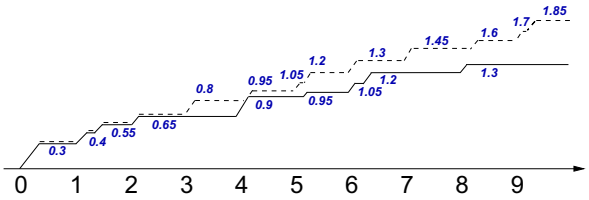


Fig. 8 Interference bound functions of the digraph models in Fig. 4 (dotted line) and in Fig. 6 (solid line)

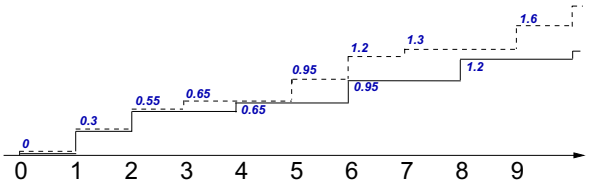


Fig. 9 Demand bound functions of the digraph models in Fig. 4 (dotted line) and in Fig. 6 (solid line)

rbf(10) (or *ibf(10)*) is 1.3, against a pessimistic estimate of 1.85 using the model of Fig. 4.

In addition, the digraph of Fig. 6 allows a more accurate definition of deadlines for the actions. In a synchronous FSM the deadline of an action should be no larger than the trigger time of the following action. Consider action a_2: in Fig. 4 its deadline is always 1, but there are instances such as $A_{2,0}$ with looser deadlines. The concept of *demand bound function* quantifies the difference that originates from different deadline assignments.

Definition 12 (*Baruah* 2003; *Zeng and Natale* 2015) The *demand bound function* (DBF) $\tau.dbf(t)$ of a digraph task τ is the maximum cumulative execution times by its jobs that have their release times *and deadlines* within any time interval of length t.

Intuitively, the RBF captures the maximum cumulative execution request, while the DBF quantifies the amount of execution time from jobs that are released and must be completed within a given time interval. The RBF (Guan et al. 2014) and DBF (Stigge et al. 2011) functions can be computed by a pseudo-polynomial algorithm. In Fig. 6, two instances of the same action have different deadlines: $A_{2,5}$ has a deadline of 1, and the deadline of $A_{2,0}$ is 2 as the next action is triggered 2 time units later. The DBF for the digraph in Fig. 6 is lower than the one in Fig. 4, as shown in Fig. 9. For example, *dbf(10)* is 1.3 for the model in Fig. 6, against a value of 1.85 using the model of Fig. 4.

Once the digraph model is generated, we can check the system schedulability under a conventional static-priority schedulability analysis. Equation (4) requires that the RBF/IBF functions are available. The calculations of $rbf(t)$ and $ibf(t)$ have pseudo-polynomial complexity using the technique proposed in Guan et al. (2014). However, this technique may be not efficient for large t. The computation can be made more efficient by leveraging the periodicity of the RBF function (Zeng and Di Natale 2013, 2015).

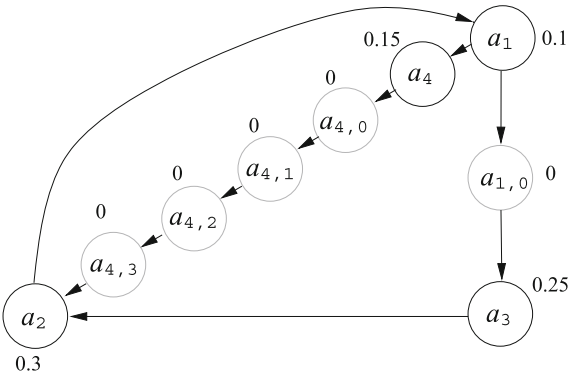


Fig. 10 The transformed unit digraph for the digraph of Fig. 4. The deadlines are omitted as they are irrelevant for execution matrix

4.5 Execution matrix for digraph tasks

The concept of execution matrix is introduced in Zeng and Di Natale (2015) to express the amount of execution time that a given set of tasks or actions may request in one time unit. The execution matrix can express the time that is required for transitioning between any two digraph locations (for a digraph model) or any two states (in a state execution model). To compute the execution matrix for a given digraph, it is first necessary to transform the digraph into a Unit Digraph (Zeng and Di Natale 2015) in which each job is separated by the predecessor and the successor by *edges with unit delay*. The transformation rule that guarantees the equivalence of the transformed graph with respect to the original graph is described in Zeng and Di Natale (2015).

Using the digraph models in Fig. 4 as an example, the transformed unit digraph is in Fig. 10. Each activation interarrival is now one unit of time and the execution model can be represented by an execution matrix $\mathbf{X}^{(1)}$ in which each element $x_{i,j}$ represents the execution time to go from job i to job j in one unit of time. The worst case execution time that may be required in any time interval Δ_d of integer length d can now be computed as follows. Assume the worst case execution times to go from any job i to any job j for any intervals of length p and q (with $p+q=d$) are known and stored in execution matrices $\mathbf{X}^{(p)}$ and $\mathbf{X}^{(q)}$ respectively, the execution matrix $\mathbf{X}^{(d)}$ can be computed as

$$x^d_{i,j} = \max\{x^p_{i,k} + x^q_{k,j}\} \tag{5}$$

which is the matrix product $\mathbf{X}^{(p)} \otimes \mathbf{X}^{(q)}$ in max-plus algebra. By recursive reasoning, the worst case execution time in any interval of length d can be obtained as the matrix power of $\mathbf{X}^{(1)}$ in max-plus algebra (Baccelli et al. 1992).

The digraph of Fig. 10, and consequently of Fig. 4, has a larger utilization (0.1625) than the original FSM (0.13). This is consistent with the findings in Sect. 4.4, that the model in Fig. 4 overestimates the function $rbf(t)$, especially for large t. Its execution request matrix is a 9×9 irreducible matrix, as there are 9 nodes in the unit digraph.

$$\mathbf{X}^{(1)} = \begin{bmatrix} -\infty & 0.1 & -\infty & 0.1 & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & 0 & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & 0.25 \\ -\infty & -\infty & -\infty & -\infty & 0.15 & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & 0 & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & 0 & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & 0 \\ 0.3 & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \end{bmatrix} \tag{6}$$

Once the worst case execution time request is represented using an execution matrix, several results of max-plus algebra theory can be applied. If the digraph is strongly connected, then the values of the matrix powers are periodically recurring. In the example of Fig. 4 the digraph is strongly connected. Thus, $\forall k \in \mathbb{N}$ and $k \geq 11$,

$$\mathbf{X}^{(k+4)} = \mathbf{X}^{(k)} + 4 \times 0.1625 \tag{7}$$

The digraph in Fig. 6 has the same utilization as the original FSM. However, unlike Fig. 4, it is not strongly connected since removing all the loose edges results in some unreachable vertices (e.g., $A_{1,0}$). To compute the periodicity parameters, we first modify it to be strongly connected by adding loose edges, e.g., $(A_{2,0}, A_{1,4})$, $(A_{2,5}, A_{1,0})$ and $(A_{2,5}, A_{1,8})$. This addition does not change the RBF and DBF functions (by the definition of loose edge). Its execution request matrix $\bar{\mathbf{X}}^{(1)}$ is a 50×50 irreducible matrix. Thus, $\forall k \in \mathbb{N}$ and $k \geq 23$,

$$\bar{\mathbf{X}}^{(k+10)} = \bar{\mathbf{X}}^{(k)} + 10 \times 0.13 \tag{8}$$

5 State-based schedulability analysis

We now discuss the schedulability analysis based on the study on the state transitions in the FSMs triggered by events with static offsets. We emphasize that all the analysis techniques presented in Sect. 4 (including the "exact analysis" in Stigge and Yi 2013, 2015a) are possibly inaccurate: they assume that task activation offsets are arbitrary and unknown. However, in synchronous FSMs tasks have the same static offset, as events are all synchronized (because of the SR semantics) even if they trigger different FSMs. This requires the quantification of execution requests on a given interval (with given start time and end time) (Zeng and Natale 2012), as opposed to request and interference functions that are only specified for a generic interval of a given length (Stigge and Yi 2013, 2015a; Guan et al. 2014). Without loss of generality, we assume that *all event offsets are 0*.

5.1 Exact response time analysis

The exact response time analysis follows the general framework of systems with static offsets (Tindell 1994), where the feasibility of each action instance is checked

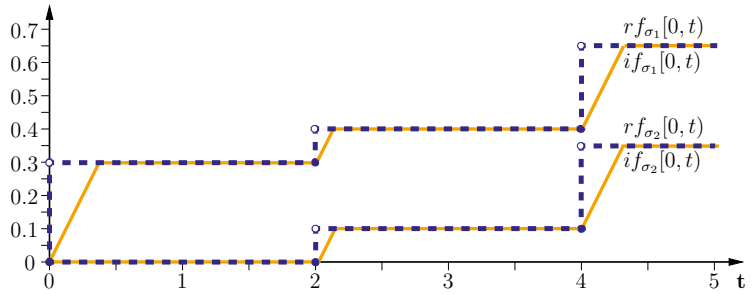


Fig. 11 Illustration of the RF and IF functions, for the two action sequences $\sigma_1 = [(a_2, 0), (a_1, 2), (a_3, 4)]$ and $\sigma_2 = [(a_1, 2), (a_3, 4)]$ of the FSM in Fig. 2

by considering a set of representative busy periods. We denote the worst case response time of an action instance $A_{k,t_k} = (a_k, t_k)$ as $R(A_{k,t_k})$, i.e., the maximum time between the release time (t_k) and the finish time of A_{k,t_k}. To calculate $R(A_{k,t_k})$, it is necessary to quantify the interferences from higher priority tasks. Like the analysis with arbitrary offsets (Stigge and Yi 2013), the accurate analysis requires the definition over a given path in the digraph task (or equivalently, an action sequence in the FSM). We refine the definitions of request and interference functions for systems with static offsets as below.

Definition 13 For an FSM $\mathcal{F}$, the request function (RF) of a legal action sequence $\sigma = [A_{1,t_1}, \ldots]$ during a time interval $\Delta = [s, f)$ (s *inclusive* and f *exclusive*), denoted as $rf_\sigma(\Delta)$, is defined as the maximum cumulative *execution time requests* from action instances of σ that have their trigger times within Δ. That is,

- $t_l = \min\{t_k | t_k \geq s\}$ and $t_m = \max\{t_k | t_k < f\}$, with $t_l \leq t_m$;
- $rf_\sigma(\Delta) = \sum_{k=l}^{m} C_{a_k}$.

If there exists no such a pair of t_l and t_m, $rf_\sigma(\Delta) = 0$.

Definition 14 For an FSM $\mathcal{F}$, the interference function (IF) of a legal action sequence $\sigma = [A_{1,t_1}, \ldots]$ during a time interval $\Delta = [s, f)$, denoted as $if_\sigma(\Delta)$, is defined as the maximum *executed loads* from action instances of σ that have their trigger times within Δ. That is,

- $t_l = \min\{t_k | t_k \geq s\}$ and $t_m = \max\{t_k | t_k < f\}$, with $t_l \leq t_m$;
- $if_\sigma(\Delta) = \sum_{k=l}^{m-1} C_{a_k} + \min(C_{a_m}, f - t_m)$.

The request and interference functions for systems with arbitrary offsets (Stigge and Yi 2013; Guan et al. 2014), are defined over any time interval of a given length t. They can be expressed as

$$rf_\sigma(t) = rf_\sigma[t_1, t_1 + t), \quad if_\sigma(t) = if_\sigma[t_1, t_1 + t) \tag{9}$$

where t_1 denotes the release time of the first action instance in σ.

We illustrate the two functions in Fig. 11 with example action sequences from the FSM in Fig. 2. As in the figure, given the start time s, $rf_\sigma[s, t)$ ($if_\sigma[s, t)$) is a

non-decreasing discontinuous staircase (continuous slanted staircase) function with respect to t. The horizontal segments of $rf_\sigma[s,t)$ are left-open and right-closed, and the slope of the segments of $if_\sigma[s,t)$ is either 0 (horizontal segments) or 1 (slanted segments).

The RF and IF functions allow to compute the worst case response time of an action instance $A_{k,t_k} = (a_k, t_k)$ by considering all the possible combinations of the legal action sequences from the tasks Γ with priority higher than a_k. Only higher priority tasks need to be considered since the scheduler is preemptive (hence no blocking delay from lower priority tasks), and there is no delay caused by other action instances from the same FSM (since all the actions must complete before the next one is activated).

Let $\Omega(\mathcal{F})$ denote the set of all the legal action sequences in the FSM $\mathcal{F}$, and $\Omega(\Gamma) := \Omega(\mathcal{F}_1) \times \Omega(\mathcal{F}_2) \times \cdots$ indicate the set of all combinations of legal action sequences from the higher priority FSMs $\Gamma = \{\mathcal{F}_1, \mathcal{F}_2, \ldots\}$. A combination of action sequences $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots\} \in \Omega(\Gamma)$ is an element of $\Omega(\Gamma)$, where $\sigma_i \in \Omega(\mathcal{F}_i)$. Let $R(A_{k,t_k}, \bar{\sigma})$ denote the response time of the action instance A_{k,t_k} under the interferences from $\bar{\sigma}$, we have

$$R(A_{k,t_k}) = \max_{\bar{\sigma} \in \Omega(\Gamma)} \{R(A_{k,t_k}, \bar{\sigma})\} \tag{10}$$

That is, the exact response time of A_{k,t_k} can be calculated by searching for the maximum value among $R(A_{k,t_k}, \bar{\sigma})$ for all the possible path combinations $\bar{\sigma}$. In the following, we focus on the calculation of $R(A_{k,t_k}, \bar{\sigma})$. For convenience, the combination of $\bar{\sigma}$ and $[A_{k,t_k}]$, the action sequence containing only A_{k,t_k}, is denoted as $\bar{\sigma}^+$. We first extend the concept of busy period (Lehoczky 1990; Tindell 1994) and define it over $\bar{\sigma}^+$.

Definition 15 A *busy period* $[s, f)$ of $\bar{\sigma}^+$ is defined as follows:

- It starts at some time s when an action instance originated from $\bar{\sigma}$ and A_{k,t_k} is triggered. All the instances from $\bar{\sigma}$ that are triggered strictly before s are ignored (i.e., assumed to have finished their executions).
- It is a continuous time interval during which no other tasks with lower priority can start execution.
- It ends at the earliest time f when there are no instances triggered in $[s, f)$ waiting to be executed.

The start times of all the busy periods that are relevant for the calculation of $R(A_{k,t_k}, \bar{\sigma})$ can be easily determined. The set includes any trigger time s for any action instance from $\bar{\sigma}$ that occurs earlier than t_k, plus t_k itself.

Given a start time s, the corresponding finish time f can be calculated using the RF function:

$$f = \min_{t>s} \{t | rf_{\bar{\sigma}^+}[s,t) \leq t - s\} \tag{11}$$

where $rf_{\bar{\sigma}^+}(\Delta) = \sum_{\sigma_i \in \bar{\sigma}^+} rf_{\sigma_i}(\Delta)$. Alternatively, f can be calculated using the IF function (where $if_{\bar{\sigma}^+}(\Delta) = \sum_{\sigma_i \in \bar{\sigma}^+} if_{\sigma_i}(\Delta)$):

$$f = \min_{t>s}\{t | if_{\bar{\sigma}^+}[s,t) \leq t - s\} \tag{12}$$

Similar to the case of arbitrary offsets (Guan et al. 2014), RF and IF functions are equivalent for the purpose of computing the finish time f (and the exact response time of A_{k,t_k}). Simply speaking, $rf_\sigma(\Delta)$ and $if_\sigma(\Delta)$ only differ at the slanted segment of $if_\sigma(\Delta)$ (with slope 1). Thus, the time instant f, at the intersection of $if_{\bar{\sigma}^+}(\Delta)$ and the line with slope 1, can only lie in the horizontal segments of $if_{\bar{\sigma}^+}(\Delta)$. Such segments must coincide with those of $rf_{\bar{\sigma}^+}(\Delta)$.

Given a start time s of the busy period of $\bar{\sigma}^+ = \{\bar{\sigma}, [A_{k,t_k}]\}$, the finish time f can be computed with Eq. (11) or (12), and the response time of A_{k,t_k} is

$$R(A_{k,t_k}, \bar{\sigma}, s) = f - t_k \tag{13}$$

The equation is a trivial extension of Eq. (1) in Tindell (1994): intuitively A_{k,t_k} can only finish at f due to the higher priority task instances from $\bar{\sigma}$ triggered in $\Delta = [s, f)$.

The response time $R(A_{k,t_k}, \bar{\sigma})$ is the maximum over all the possible busy periods that start no later than t_k.

$$R(A_{k,t_k}, \bar{\sigma}) = \max_{s \leq t_k}\{R(A_{k,t_k}, \bar{\sigma}, s)\} \tag{14}$$

For example, consider an action instance A_{k,t_k} where $C_{a_k} = 0.4$ and $t_k = 2.1$. We assume its interfering combination $\bar{\sigma} = \{\sigma_1, \sigma_2\}$ where σ_1 and σ_2 refer to the two interfering FSMs, for which the RF and IF functions are shown in Fig. 11. The trigger times are $\{0, 2, 4\}$ and $\{2, 4\}$ for the two higher priority action sequences σ_1, σ_2 respectively. Hence, the set of possible start times to be considered is $\{0, 2, 2.1\}$ (4 is not included since it is larger than the trigger time 2.1 of a_k). By Eq. (13)

$$\begin{cases} R(A_{k,t_k}, \bar{\sigma}, 0) = 0.3 - 2.1 = -1.8 \\ R(A_{k,t_k}, \bar{\sigma}, 2) = 2.6 - 2.1 = 0.5 \\ R(A_{k,t_k}, \bar{\sigma}, 2.1) = 2.5 - 2.1 = 0.4 \end{cases}$$

The negative response time -1.8 indicates that the corresponding busy period finishes before A_{k,t_k}. The response time $R(A_{k,t_k}, \bar{\sigma})$ is 0.5, the maximum over those of the busy periods.

To calculate the response time, we must consider all the possible combinations of action sequences from the interfering task set Γ, and generate the corresponding request functions. This computation has exponential complexity due to the exponential number of possible combinations of action sequences from each FSM (or paths of the corresponding digraph task) (Stigge and Yi 2013). The optimization techniques presented in Stigge and Yi (2013, 2015a) can be used to speed up the analysis, but the worst case complexity remains the same.

We first refine the concept of dominance relation on the request functions (Stigge and Yi 2013, 2015a) with a given time interval.

Definition 16 For two request functions rf_σ and $rf_{\sigma'}$ (where σ and σ' are two legal action sequences), rf_σ *dominates* $rf_{\sigma'}$ on the time interval $\Delta = [s, f)$, written as $rf_\sigma \succcurlyeq_\Delta rf_{\sigma'}$, if and only if

$$\forall [s', f') \subseteq [s, f) : \quad rf_\sigma[s', f') \geq rf_{\sigma'}[s', f')$$

The *critical request function set* (or $\mathcal{F}.CRF_\Delta$) for an FSM $\mathcal{F}$ is the maximum set of request functions rf_σ ($\sigma \in \mathcal{F}$) such that there exits no other action sequence σ' with $rf_{\sigma'} \succcurlyeq_\Delta rf_\sigma$.

Let $CRF_\Delta(\mathcal{F})$ denote the set of critical request functions of the FSM $\mathcal{F}$, and $CRF_\Delta(\Gamma) := CRF_\Delta(\mathcal{F}_1) \times CRF_\Delta(\mathcal{F}_2) \times \cdots$ indicate the set of all combinations of critical request functions from the task set Γ. Then Eq. (10) for calculating the response time of A_{k,t_k} can be simplified without losing any accuracy

$$R(A_{k,t_k}) = \max_{\bar{\sigma} \in CRF_{[0,R(A_{k,t_k}))}(\Gamma)} \{R(A_{k,t_k}, \bar{\sigma})\} \tag{15}$$

This means that we only need to generate the critical request functions within the time interval $\Delta = [0, R(A_{k,t_k}))$, with a significant reduction in time complexity.

In practice, the finish time $R(A_{k,t_k})$ of Δ, the time interval for checking dominance relationship, can be safely replaced with the upper bound on $R(A_{k,t_k})$. For example, the one developed in Sect. 5.2 can be used. Furthermore, since any action must be finished before the next action, the start time of the time interval Δ can use any release time of A_{j,t_j}, as long as the end state of a_j is the start state of a_k.

In addition, the combinatorial abstraction refinement technique proposed in Stigge and Yi (2013, 2015a) can be used to simplify the computation of Eqs. (10) and (15). The idea is based on an iterative procedure that refines the abstraction of paths until the schedulability condition is satisfied.

Although the above techniques, similar to those in Stigge and Yi (2013, 2015a), can help reduce the runtime, the complexity remains to be strongly coNP-hard (Stigge and Yi 2012). In the next subsection, we present an approximate analysis that avoids the exhaustive enumeration of all the combinations of the action sequences.

5.2 Approximate response time analysis

The approximate analysis uses one request bound function or interference bound function for each FSM to abstract the workload for all the action sequences. Such an abstraction results in pseudo-polynomial complexity. We first define the RBF and IBF for an FSM over a time interval with given start and finish times.

Definition 17 The *request bound function* (RBF) of an FSM $\mathcal{F}$ during a time interval $\Delta = [s, s + t)$, denoted as $\mathcal{F}.rbf(\Delta)$, is the maximum sum of execution times by the actions of $\mathcal{F}$ that have their trigger time within Δ. That is

$$\mathcal{F}.rbf(\Delta) = \max_{\sigma \in \Omega(\mathcal{F})} \{\mathcal{F}.rf_\sigma(\Delta)\}. \tag{16}$$

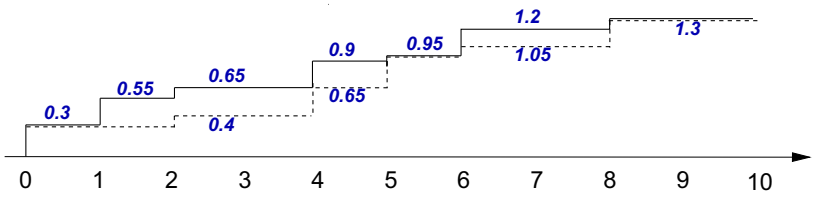


Fig. 12 $rbf[0, 10)$ (dotted line) and $rbf(10)$ (solid line) of the FSM in Fig. 2

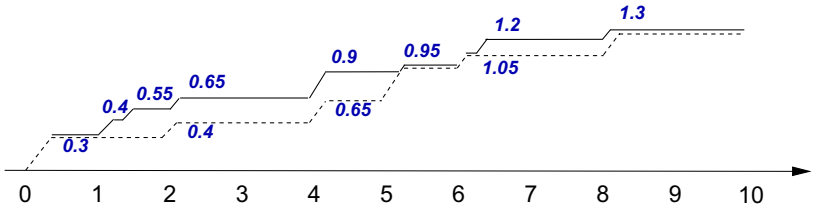


Fig. 13 $ibf[0, 10)$ (dotted line) and $ibf(10)$ (solid line) of the FSM in Fig. 2

Definition 18 The *interference bound function* (IBF) of an FSM $\mathcal{F}$ during a time interval $\Delta = [s, s+t)$, denoted as $\mathcal{F}.ibf(\Delta)$, is the maximum amount of executed workload from the actions of $\mathcal{F}$ that have their trigger time within Δ. That is

$$\mathcal{F}.ibf(\Delta) = \max_{\sigma \in \Omega(\mathcal{F})} \{if_\sigma(\Delta)\}. \tag{17}$$

Similar to $rbf(t)$ and $ibf(t)$ defined over intervals of given length t, $rbf[s, s + t)$ and $ibf[s, s + t)$ are both monotonically increasing with respect to t. By their definitions, these functions satisfy the following relationships:

$$rbf(t) = \max_s \{rbf[s, s + t)\}, \quad ibf(t) = \max_s \{ibf[s, s + t)\} \tag{18}$$

Figure 12 shows the $rbf[0, 10)$ function in the first hyperperiod of the FSM in Fig. 2. As a comparison, $rbf(10)$ for any time interval of length 10 is also shown in the figure. Fig. 13 shows a similar comparison between $ibf[0, 10)$ and $ibf(10)$.

We now use these functions to derive safe upper bounds on the response times of an action instance A_{k,t_k}. Similar to that with the IF and RF functions, the analysis with the IBF and RBF functions requires to check a set of representative busy periods. Considering the set of higher priority tasks Γ, the set of start times can be safely estimated as the union of $\mathbb{T}_\mathcal{F}$, $\forall \mathcal{F} \in \Gamma$. The finish time f of the busy period for a given start time s is

$$\begin{aligned} f_{RBF} &= \min_{t>s} \left\{ t \mid \sum_{\forall \mathcal{F} \in \Gamma} \mathcal{F}.rbf[s, t) + \delta(s, t, t_k) \cdot C_{a_k} \le t - s \right\} \\ f_{IBF} &= \min_{t>s} \left\{ t \mid \sum_{\forall \mathcal{F} \in \Gamma} \mathcal{F}.ibf[s, t) + \delta(s, t, t_k) \cdot C_{a_k} \le t - s \right\} \end{aligned} \tag{19}$$

where the function $\delta(s, t, x)$ is defined as

$$\delta(s,t,x) = \begin{cases} 1, & \text{if } x \in [s,t); \\ 0, & \text{otherwise.} \end{cases} \tag{20}$$

Using the RBF, the response time of A_{k,t_k} during the busy period starting at s and the approximate response time of A_{k,t_k} are

$$\begin{aligned} R_{RBF}(A_{k,t_k}, s) &= f_{RBF} - t_k \\ R_{RBF}(A_{k,t_k}) &= \max_{s \le t_k}\{R_{RBF}(A_{k,t_k}, s)\} \end{aligned} \tag{21}$$

Likewise, the response time estimates using IBF are

$$\begin{aligned} R_{IBF}(A_{k,t_k}, s) &= f_{IBF} - t_k \\ R_{IBF}(A_{k,t_k}) &= \max_{s \le t_k}\{R_{IBF}(A_{k,t_k}, s)\} \end{aligned} \tag{22}$$

We now study the relationship between the exact and approximate response times. For any action sequence combination $\bar{\sigma}$ in $\Omega(\Gamma)$, it is $rf_{\bar{\sigma}}(\Delta) \le \sum_{\mathcal{F}\in\Gamma} \mathcal{F}.rbf(\Delta)$. The finish times in Eq. (11) and, consequently, the response time estimates of A_{k,t_k}, are monotonically increasing with the RF function. Hence, $\forall s$, $\forall \bar{\sigma}$, we have $R_{RF}(A_{k,t_k}, \bar{\sigma}, s) \le R_{RBF}(A_{k,t_k}, s)$. This implies $R_{RF}(A_{k,t_k}) \le R_{RBF}(A_{k,t_k})$. If we indicate the state based analysis methods with the suffix-*SO* to distinguish them with respect to the digraph based analysis, the following dominance relations apply

$$\begin{array}{ccc} \text{IF-SO} & \succcurlyeq & \text{IBF-SO} \\ \| & & \curlyvee \\ \text{RF-SO} & \succcurlyeq & \text{RBF-SO} \end{array}$$

In order to verify schedulability, we must consider all the possible busy periods of $\bar{\sigma}^+$. Because of the periodicity of the trigger events, a busy period starts at an integer multiple of the event period within the system hyperperiod H. For example, consider a system including 2 blocks: a high priority Dataflow block with period 4, and the low priority Stateflow block in Fig. 2 (with event periods $\{2, 5\}$). The start times of the busy periods to be considered are $\{0, 2, 4, 5, 6, 8, 10, 12, 14, 15, 16, 18\}$.

The functions $rbf[s, f)$ and $ibf[s, f)$ can be computed by Eqs. (16) and (17) respectively, which require to enumerate all the action sequences. However, there exists a more efficient solution based on dynamic programming, as discussed in the next subsection.

5.3 Calculation of *rbf*() and *ibf*()

In this subsection, we show how to compute $rbf(\Delta)$ and $ibf(\Delta)$. Compared to Stigge et al. (2011), there are two possible improvements for synchronous FSMs. First, the possible event pattern repeats every hyperperiod, and the same applies to the RBF and

IBF functions. This leads to the concept of execution request matrix, and to the definition of the RBF and IBF functions for one hyperperiod (with the refinement by a pair of start and end states). Second, the algorithm for computing RBF and IBF can be further improved by leveraging the periodicity of the execution request matrix. Because of the periodicity of the trigger events, if we move (left or right) the time interval Δ by an integer number of hyperperiods, the functions $rbf(\Delta)$ and $ibf(\Delta)$ remain the same. Formally, $\forall k \in \mathbb{N}$,

$$\begin{cases} \mathcal{F}.rbf[s + kH_{\mathcal{F}},\ f + kH_{\mathcal{F}}) = \mathcal{F}.rbf[s,\ f) \\ \mathcal{F}.ibf[s + kH_{\mathcal{F}},\ f + kH_{\mathcal{F}}) = \mathcal{F}.ibf[s,\ f) \end{cases} \tag{23}$$

Furthermore, the trigger times and absolute deadlines for the action instances of $\mathcal{F}$ can only belong to the *ordered* set $\mathbb{T}_{\mathcal{F}}$. Hence, we only need to consider the calculations of $rbf[s,\ f)$ and $ibf[s,\ f)$ for all $s,\ f \in \mathbb{T}_{\mathcal{F}}, s \in [0,\ H_{\mathcal{F}})$.

We now refine the concepts of $rbf(\Delta)$ and $ibf(\Delta)$ for a given pair of start and end states.

Definition 19 Given the start and end states s_i and s_j, the *request bound function* $\mathcal{F}.rbf_{i,j}(\Delta)$, is defined as the maximum cumulative execution times of any legal action sequence $[(a_k, t_k), k = 1, \ldots, n]$ of $\mathcal{F}$ such that

- the source state of the first transition is s_i; and
- the sink state of the last transition is s_j; and
- $t_1 \geq s$ and $t_n < f$; and
- $\mathcal{F}.rbf_{i,j}(\Delta) = \max\{\sum_{k=1}^{n} C_{a_k}\}$.

If s_j is not reachable from s_i, then $\mathcal{F}.rbf_{i,j}(\Delta)$ is defined as $-\infty$.

Definition 20 Given the start and end states s_i and s_j, the *interference bound function* $\mathcal{F}.ibf_{i,j}(\Delta)$, is defined as the maximum amount workload of any legal action sequence $[(a_k, t_k), k = 1, \ldots, n]$ of $\mathcal{F}$ such that

- the source state of the first transition is s_i; and
- the sink state of the last transition is s_j; and
- $t_1 \geq s$ and $t_n < f$; and
- $\mathcal{F}.ibf_{i,j}(\Delta) = \max\left\{\sum_{k=1}^{n-1} C_{a_k} + \min(C_{a_n},\ f - t_n)\right\}$.

If s_j is not reachable from s_i, then $\mathcal{F}.ibf_{i,j}(\Delta)$ is defined as $-\infty$.

The domain for the possible RBF and IBF values is $\mathbb{R}^* = \mathbb{R} \bigcup \{-\infty\}$. By these definitions,

$$\mathcal{F}.rbf(\Delta) = \max_{i,j}\{\mathcal{F}.rbf_{i,j}(\Delta)\}, \quad \mathcal{F}.ibf(\Delta) = \max_{i,j}\{\mathcal{F}.ibf_{i,j}(\Delta)\} \tag{24}$$

Also, $rbf_{i,j}(\Delta)$ is additive, i.e. $\forall i,\ j, \forall t \in [s,\ f]$,

$$\mathcal{F}.rbf_{i,j}[s,\ f) = \max_{m}\{\mathcal{F}.rbf_{i,m}[s, t) + \mathcal{F}.rbf_{m,j}[t,\ f)\} \tag{25}$$

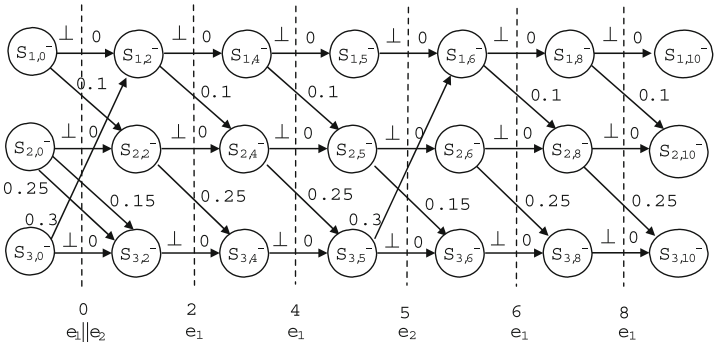


Fig. 14 Reachability graph in the first hyperperiod of the FSM in Fig. 2

However, $ibf_{i,j}(\Delta)$ is additive only for some specific points of t, i.e., $\forall i, j, \forall t \in \{t | t \in \mathbb{T}_{\mathcal{F}}, s \leq t \leq f\}$,

$$\mathcal{F}.ibf_{i,j}[s, f) = \max_m \{\mathcal{F}.ibf_{i,m}[s, t) + \mathcal{F}.ibf_{m,j}[t, f)\} \tag{26}$$

For $t \in \mathbb{T}_{\mathcal{F}}$, it is also $ibf_{i,m}[s, t) = rbf_{i,m}[s, t)$, and the computation of $ibf_{i,j}(\Delta)$ can be derived from the combination $rbf_{i,m}[s, t)$ and $ibf_{m,j}[t, f)$.

Thus, $rbf_{i,j}[s, f)$ for a long interval $[s, f)$ can be computed from its values for shorter intervals $[s, t)$ and $[t, f)$, and dynamic programming can be used for an efficient calculation of $rbf_{i,j}(\Delta)$. The computation of $rbf_{i,j}(\Delta)$ requires searching the reachable states within the possible sequences of events. We build a *reachability graph* where each state s_i corresponds to an infinite sequence of vertices. Each vertex S_{i,t_i^-} corresponds to the source state s_i before any action triggered by the event at time t_i, $\forall t_i \in \mathbb{T}_{\mathcal{F}}$. The edge $(S_{i,t_i^-}, S_{j,t_{i+1}^-})$ is added when:

- t_{i+1} is the next time instant after t_i in the set $\mathbb{T}_{\mathcal{F}}$;
- there is a transition $\theta_k = \{s_i, s_j, e_{\theta_k}, g_k, a_k, p_k\} \in \mathbb{A}$ from s_i to s_j in the FSM, and the edge is labeled with C_{a_k};
- t_i is an integer multiple of $T_{e(a_k)}$.

Intuitively, edge $(S_{i,t_i^-}, S_{j,t_{i+1}^-})$ corresponds to a transition from s_i to s_j at time t_i. Furthermore, stuttering edges from S_{i,t_i^-} to S_{i,t_{i+1}^-} (denoted by the symbol $\perp$ and labeled with 0) are added to represent the possible stuttering behavior. As an example, Fig. 14 shows the reachability graph of the FSM in Fig. 2.

Every path from S_{i,t_i^-} to S_{j,t_j^-} in the reachability graph corresponds to a possible legal action sequence from s_i to s_j during the interval $[t_i, t_j)$. Thus, $rbf_{i,j}[t_i, t_j)$ can be computed as the longest path from S_{i,t_i^-} to S_{j,t_j^-}. This problem can be solved in cubic time to the size of the graph (i.e., cubic to the length of the time interval) by negating the weight of the edges and leveraging the classic Floyd-Warshall algorithm (Floyd 1962) for solving the all-pairs shortest path problem.

Let $n_s = \lceil \frac{s}{H} \rceil$ and $n_f = \lfloor \frac{f}{H} \rfloor$. $rbf(\Delta)$ can be decomposed as

$$\begin{aligned} rbf_{i,j}[s, f) &= \max_{k,l}\{rbf_{i,k}[s, n_s H) + rbf_{k,l}[n_s H, n_f H) + rbf_{l,j}[n_f H, f)\} \\ &= \max_{k,l}\{rbf_{i,k}[s, n_s H) + rbf_{k,l}[n_s H, n_f H) + rbf_{l,j}[0, f - n_f H)\} \end{aligned} \tag{27}$$

This equation is a special case of (25), i.e., the RBF function can be decomposed into functions defined over three intervals, namely $[s, n_s H)$, $[n_s H, n_f H)$, and $[n_f H, f)$. $rbf_{i,k}[s, n_s H)$ and $rbf_{l,j}[n_f H, f)$ can be computed directly by finding the longest path between the two states in the reachability graph, since the intervals $[s, n_s H)$ and $[n_f H, f)$ are within one hyperperiod. Clearly, it is inefficient to build the reachability graph for multiple hyperperiods to calculate directly $rbf_{k,l}[n_s H, n_f H)$.

For simplicity, we denote the request bound function within k hyperperiods for a pair of given start and end states as $x_{i,j}^{(k)} = rbf_{i,j}[0, kH)$, and the $n \times n$ matrix $\mathbf{X}^{(k)}(\mathcal{F}) = (x_{i,j}^{(k)}, i, j = 1, \ldots, n)$ where $n = |\mathbb{S}|$ is the number of states in the FSM. We define $\mathbf{X}^{(1)}(\mathcal{F})$ as *the execution request matrix* of the FSM $\mathcal{F}$, and simply denote it as $\mathbf{X}$. As a special case of Equation (25), we have

$$\forall i, j, \forall 1 \leq l < k, \quad x_{i,j}^{(k)} = \max_m \left\{ x_{i,m}^{(l)} + x_{m,j}^{(k-l)} \right\} \tag{28}$$

Equation (28) can be used to compute the RBF function for intervals spanning multiple hyperperiods instead of building large reachability graphs. This results in a significant improvement, typically one to two orders of magnitudes. Hence, Eq. (27) can be expressed as the following:

$$rbf_{i,j}[s, f) = \max_{k,l} \left\{ rbf_{i,k}[s, n_s H) + x_{k,l}^{(n_f - n_s)} + rbf_{l,j}[0, f - n_f H) \right\} \tag{29}$$

Given the execution request matrix, we can use the periodicity property introduced in Sect. 3 (e.g., Theorem 1) to further speedup the calculation of Eq. (28).

6 Experimental results

In this section, we use randomly generated systems of synchronous finite state machines to evaluate the schedulability analysis techniques and compare their accuracy and runtime. The list of analysis techniques is as follows:

- *RF/IF-SO* the state-based exact analysis using RF/IF, where tasks have static offsets (Sect. 5.1).
- *RF/IF-AO1* the analysis using RF/IF with arbitrary offsets for the digraph with action instances (Sect. 4.3).
- *RF/IF-AO2* the analysis using RF/IF with arbitrary offsets for the digraph with actions (Sect. 4.2).
- *IBF-SO* the state-based approximate analysis using IBF with static offsets (Sect. 5.2).

- *RBF-SO* the state-based approximate analysis using RBF with static offsets (Sect. 5.2).
- *IBF-AO1* the approximate analysis using IBF with arbitrary offsets. In this case, three approaches of different complexity can be used to compute the exact $ibf(t)$:
  - *IBF-AO1-(18)* using Eq. (18), which requires to first compute $ibf[s, s+t)$ for any $s \geq 0$.
  - *IBF-AO1-Digraph* using the algorithm in Guan et al. (2014) on the digraphs with action instances.
  - *IBF-AO1-FSM* using a modification of the algorithm in Guan et al. (2014) on reachability graphs (e.g., Fig. 14).
- *RBF-AO1* the approximate analysis using RBF with arbitrary offsets. Similar to *IBF-AO1*, three approaches are used to compute the $rbf(t)$: *RBF-AO1-(18)*, *RBF-AO1-FSM* and *RBF-AO1-Digraph*.
- *IBF-AO2* the approximate analysis using IBF with arbitrary offsets for the digraph with actions. The algorithm in Guan et al. (2014) is used to compute the IBF function.
- *RBF-AO2* the approximate analysis using RBF with arbitrary offsets for the digraph with actions using the algorithm in Guan et al. (2014).

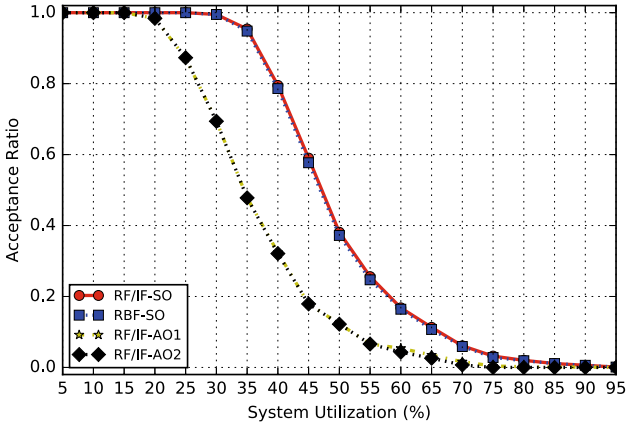
For the *RF/IF-AO1* and *RF/IF-AO2* analyses using digraphs, we used the techniques proposed in Stigge and Yi (2013, 2015a) to improve the efficiency of the computations. The same techniques are also used for *RF/IF-SO*. Dynamic programming techniques are applied to improve the efficiency of the analysis methods with prefix *IBF-* or *RBF-*.

We first use a relatively simple set of FSMs, to allow for the evaluation of a significant number of cases using the computationally expensive analysis (such as *IF-SO* and *RF-SO*), as in Sects. 6.1 and 6.2. The periodicity property of the execution request matrix on the state machines or the digraph tasks is used in subsequent experiments with a much larger hyperperiod and more complex set of activation events (Sect. 6.3). Finally, we study the scalability of the approximate analysis methods (Sect. 6.4).

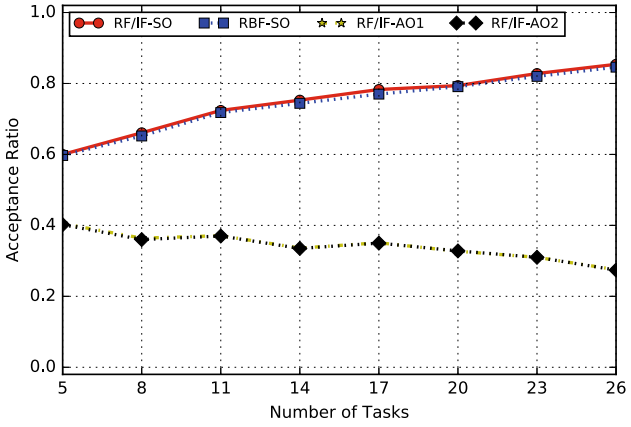
6.1 Comparing RBF-SO with exact analysis methods

The first set of experiments compares *RF/IF-SO* with *RF/IF-AO1* and *RF/IF-AO2*, to evaluate the effect on different digraph models and different task offset settings. In addition, *RBF-SO* is compared to evaluate the impact of IBF/RBF approximations. Due to the complexity of the analyses with IF/RF, we generate relatively simple random systems. For each FSM, the number of states is randomly distributed as follows: 20% of the FSMs has only one state, 20% of them has two states, 20% with three states, 20% with five states, 10% with ten states, 5% with fifteen states, and the remaining 5% has twenty states. Transitions are randomly connected between states such that the average number of outgoing edges is 3. Two types of event periods are used for this set of experiments:

Type-I Event periods are randomly drawn from the set {10, 20, 25, 50}.
Type-II Event periods are randomly drawn from the set {10, 20, 30, 40, 50, 60}.



(a) Acceptance ratio vs. system utilization ($n = 20$)



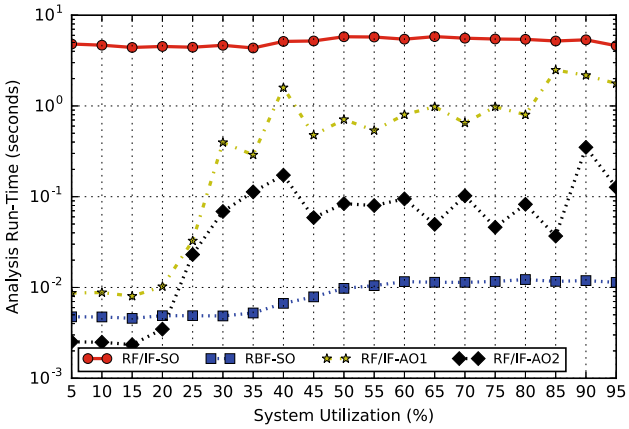
(b) Acceptance ratio vs. number of tasks ($U = 40\%$)

Fig. 15 The acceptance ratios of *RF/IF-SO*, *RF/IF-AO1*, and *RF/IF-AO2* with *Type-I* event periods

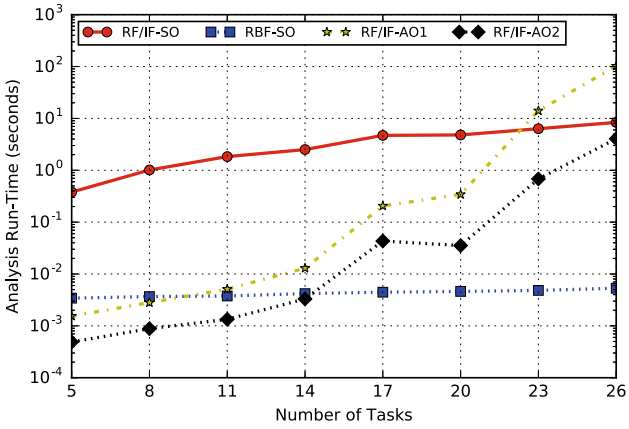
The WCETs of actions are assigned such that the utilization of the FSM is uniformly distributed. The priorities of FSMs are assigned using the rate monotonic policy based on their base period (*gcd* of the event periods in the FSM). Two parameters are used as variables in the generation of the task systems: the total utilization U and the number of tasks n. 1000 random system are generated for each configuration and a timeout of 200 seconds was applied to all runs.

The acceptance ratio between the number of schedulable systems and the total number of generated systems is used to compare the accuracy of the analyses. The experiments in Figs. 15 and 16 use *Type-I* event periods, while the ones in Figs. 17 and 18 adopt *Type-II* event periods.

We first set the number of tasks to $n = 20$ and vary the system utilization U from 5 to 95%, as in Figs. 15a and 17a. Then the number of tasks n is varied from 5 to 26 with



(a) Runtime vs. system utilization ($n = 20$)

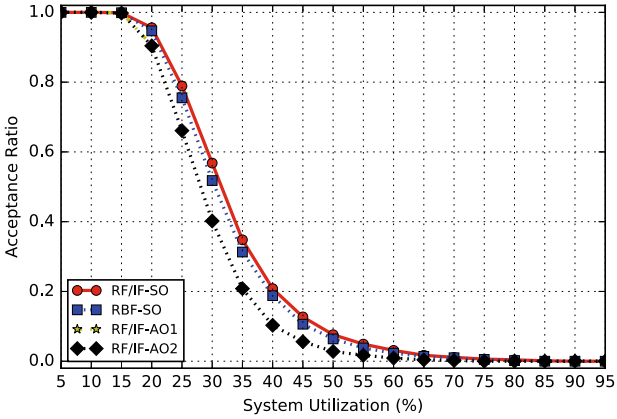


(b) Runtime vs. number of tasks ($U = 40\%$)

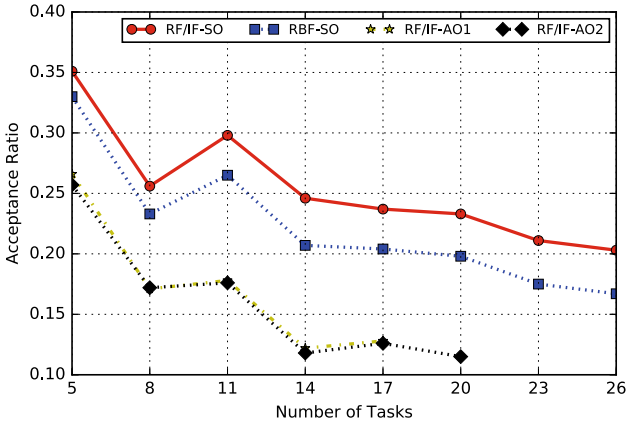
Fig. 16 The analysis runtimes of *RF/IF-SO*, *RF/IF-AO1*, and *RF/IF-AO2* with *Type-I* event periods

a 40% system utilization, as in Figs. 15b and 17b. The results indicate that *RF/IF-SO* and *RBF-SO* have similar accuracy: the maximum difference between them is less than 5%. For *Type-I* event periods *RBF-SO* is almost as accurate as *RF/IF-SO* since the maximum difference is less than 1%. Compared with *RF/IF-AO1* and *RF/IF-AO2*, *RBF-SO* has a noticeably higher acceptance ratio across all the settings. As observed in these graphs, the relative comparison for the analysis accuracy among these methods remains the same regardless of the value of the system parameters.

Figures 16 and 18 illustrate the corresponding average runtime of the four analysis methods. *RF/IF-SO* is about two orders of magnitude slower than *RBF-SO*, since it needs to perform the schedulability analysis by checking all the possible combinations of action sequences. In addition, *RF/IF-AO1* and *RF/IF-AO2* have lower accuracy than *RBF-SO*, with much worse scalability (the two analysis methods have significantly



(a) Acceptance ratio vs. system utilization ($n = 20$)



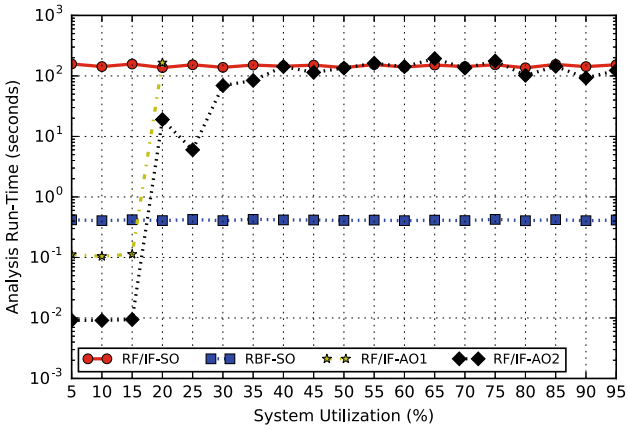
(b) Acceptance ratio vs. number of tasks ($U = 40\%$)

Fig. 17 The acceptance ratios of *RF/IF-SO*, *RF/IF-AO1*, and *RF/IF-AO2* with *Type-II* event periods

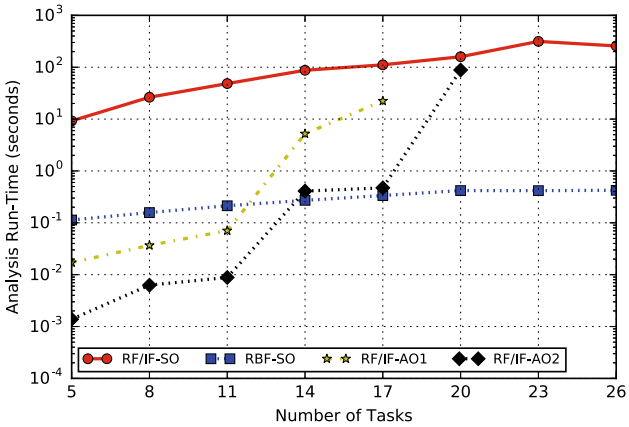
larger runtime than *RBF-SO* for larger task sets). In most cases, *RBF-SO* has the smallest runtime as it only needs to consider one RBF function for each FSM instead of a larger number of action sequence (or path) combinations. Compared to *Type-I*, the runtimes for *Type-II* event periods are significantly larger for the same system parameters, since the system hyperperiod is much longer.

6.2 Comparing RBF/IBF based approximate analyses

In this set of experiments, the systems are generated as in Sect. 6.1, but the FSMs have a larger number of states and a larger set of possible event periods. Each FSM has a random number of states such that 20% of the FSMs have five states, 20% ten states,



(a) Runtime vs. system utilization ($n = 20$)



(b) Runtime vs. number of tasks ($U = 40\%$)

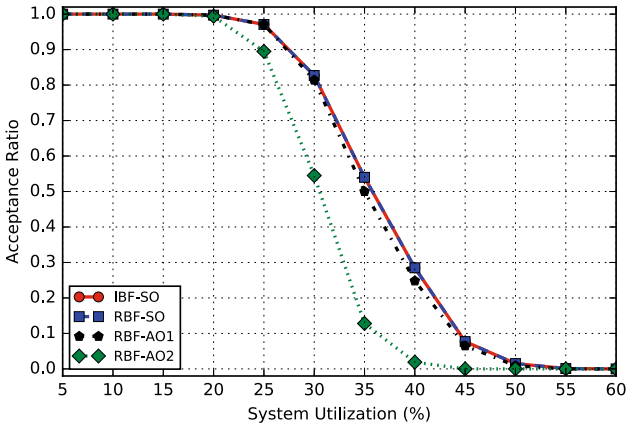
Fig. 18 The analysis runtimes of *RF/IF-SO*, *RF/IF-AO1*, and *RF/IF-AO2* with *Type-II* event periods

20% fifteen states, 20% twenty states, 10% twenty-five states, and the remaining 10% fifty states. Two types of event periods are used in the set of experiments:

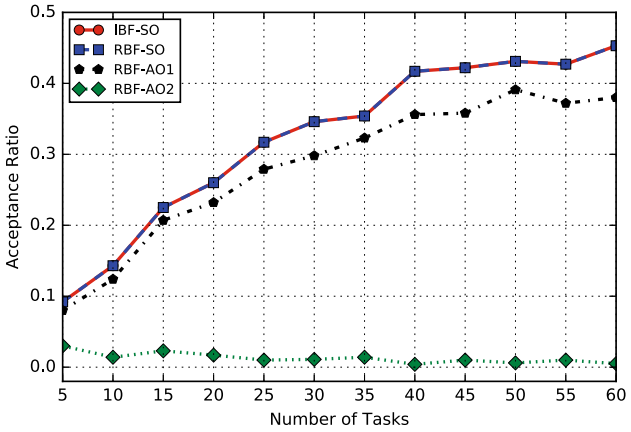
Type-III The base period of each FSM is randomly extracted from the set {1, 2, 5, 10, 20, 25, 50, 100}, and the event periods within the same FSM are computed as the product of the base period and a factor randomly chosen from the set {1, 2, 5, 10}.

Type-IV Similar to *Type-III* but the base period of each FSM is generated by the product of one to *two* factors, each randomly drawn from three harmonic sets (2,4), (6,12), (5,10).

We compare the approximate analysis techniques: *IBF-SO*, *RBF-SO*, *IBF-AO1*, *RBF-AO1*, *IBF-AO2*, and *RBF-AO2*. In the case of the last four analysis methods, the analysis runtime is computed for all the possible approaches for calculating IBF or RBF.



(a) Acceptance ratio vs. system utilization ($n = 20$)

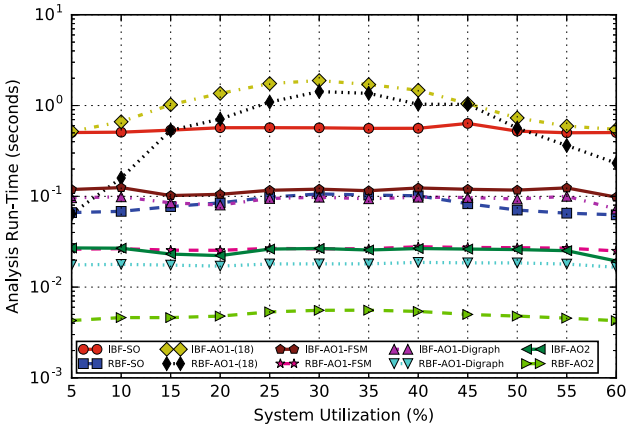


(b) Acceptance ratio vs. number of tasks ($U = 40\%$)

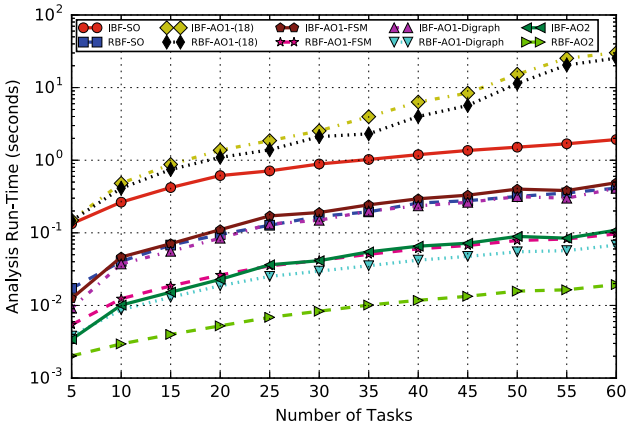
Fig. 19 The acceptance ratios of approximate analysis methods with *Type-III* event periods

Figures 19, 20, 21, and 22 illustrate the comparison on their acceptance ratios and runtimes with the two types of event periods. In Figs. 19 and 21 we only show *IBF-SO* and omit the other IBF based analysis (*IBF-AO1* and *IBF-AO2*), as they are practically indistinguishable with respect to their RBF based counterparts (*RBF-SO*, *RBF-AO1*, and *RBF-AO2*, respectively).

The runtime of *IBF-SO* is noticeably larger than *RBF-SO*. This is a result of the higher complexity to calculate IBF than RBF, due to the need to compute the intersection of the slanted segment of one IF with the horizontal segment of another. There is a very limited accuracy improvement when using *IBF-SO* (as expected). *RBF-SO* provides results that are very close to the exact analysis, as was the case for the first set of experiments (Sect. 6.1). *IBF-AO1* and *RBF-AO1* provide results that are very close in terms of acceptance ratio. Likewise for *IBF-AO2* and *RBF-AO2*. These results



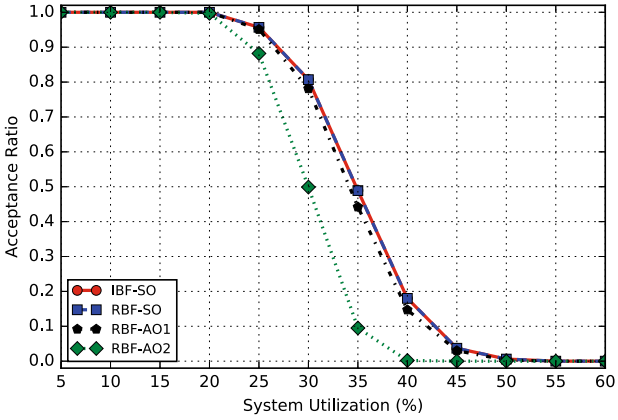
(a) Runtime vs. system utilization ($n = 20$)



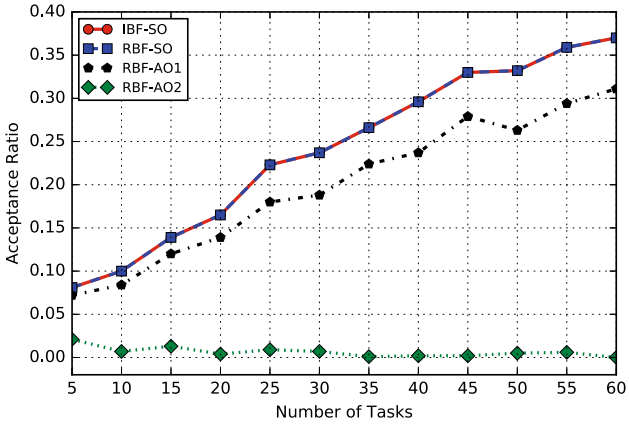
(b) Runtime vs. number of tasks ($U = 40\%$)

Fig. 20 The analysis runtimes of approximate analysis methods with *Type-III* event periods

indicate that for such randomly generated systems, the analyses based on IBF and RBF have similar quality, but the former always require a longer runtime. In Fig. 20, the methods with arbitrary offsets *IBF-AO1*, *RBF-AO1*, *IBF-AO1-(18)* and *RBF-AO1-(18)* respectively have the longer runtimes than the corresponding analyses with static offsets since they need to first calculate the IBF and RBF with static offsets before obtaining the functions with arbitrary offsets by Eq. (18). On the contrary, in Fig. 22, *IBF-AO1-(18)* and *RBF-AO1-(18)* respectively have the smaller runtimes than the corresponding analyses with static offsets since *IBF-SO* and *RBF-SO* have to verify a much larger number of action instances for the *Type-IV* setting. Clearly, this is not an efficient approach, as the direct calculation of the FSM reachability graph (as with *IBF-AO1-FSM* and *RBF-AO1-FSM*) is much faster. Among the three approaches for computing *IBF-AO1* and *RBF-AO1*, *IBF-AO1-Digraph* and *RBF-AO1-Digraph* have



(a) Acceptance ratio vs. system utilization ($n = 20$)

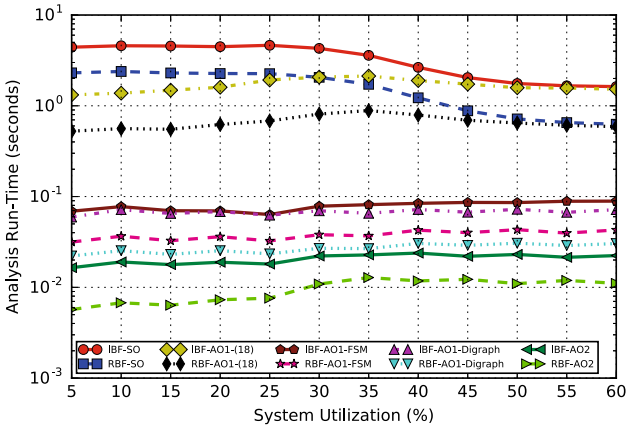


(b) Acceptance ratio vs. number of tasks ($U = 40\%$)

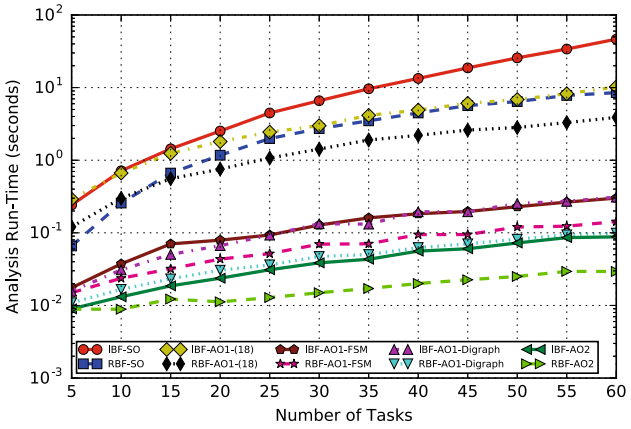
Fig. 21 The acceptance ratios of approximate analysis methods with *Type-IV* event periods

the smallest runtimes, that is, they provide the most efficient way to calculate IBF and RBF by using the digraph with action instances. However, *IBF-AO1-FSM* (*RBF-AO1-FSM*) is almost as efficient as *IBF-AO1-Digraph* (*RBF-AO1-Digraph*).

Compared to *RBF-SO*, both *RBF-AO1* and *RBF-AO2* use the digraph model with arbitrary offsets, and have larger pessimism. However, their runtimes are shorter. *RBF-AO2* has the lowest accuracy but the smallest runtime because of the simplest graph model. Moreover, the results show that a digraph with action instances allows for more accurate results, since *RBF-AO1* gives a significantly higher acceptance ratio than *RBF-AO2*.



(a) Runtime vs. system utilization ($n = 20$)

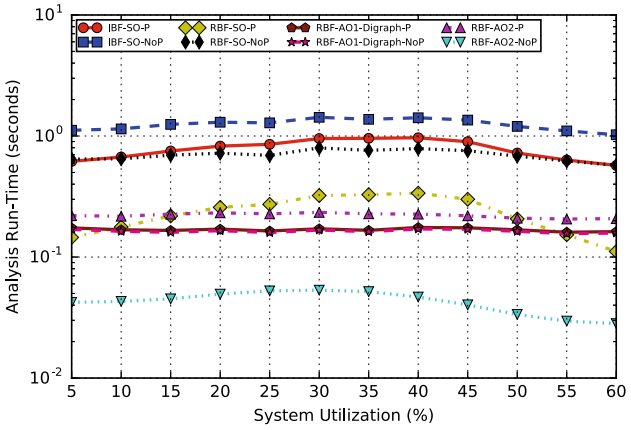


(b) Runtime vs. number of tasks ($U = 40\%$)

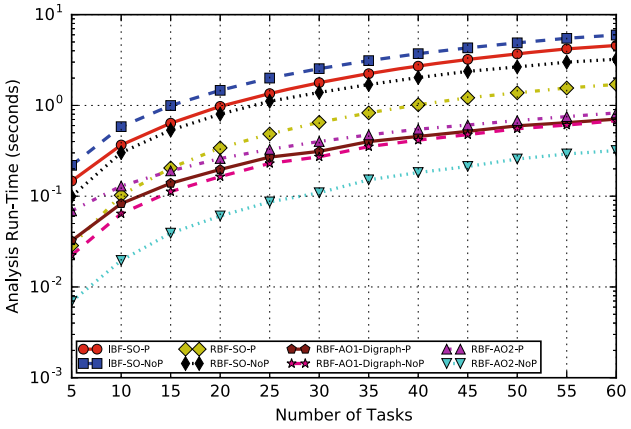
Fig. 22 The analysis runtimes of approximate analysis methods with *Type-IV* event periods

6.3 Analysis of large busy periods leveraging the periodicity

The periodicity of the request and interference bound functions computed from the execution matrix can be used to improve the runtime of the exact and approximate analysis methods when the hyperperiod is large and the event periods differ significantly. In this set of experiments, we evaluate the runtime of the analysis when the periodicity parameters are computed according to four methods: *IBF-SO*, *RBF-SO*, *RBF-AO1-Digraph* and *RBF-AO2*. The suffix denotes the use of the periodicity property (*-P*) or a standard analysis (*-NoP*). The experiment setting is similar to the previous set of experiments, but the base period of each FSM has a larger range. Two types of event periods are used in this set of experiments:



(a) Runtime vs. system utilization ($n = 20$)



(b) Runtime vs. number of tasks ($U = 40\%$)

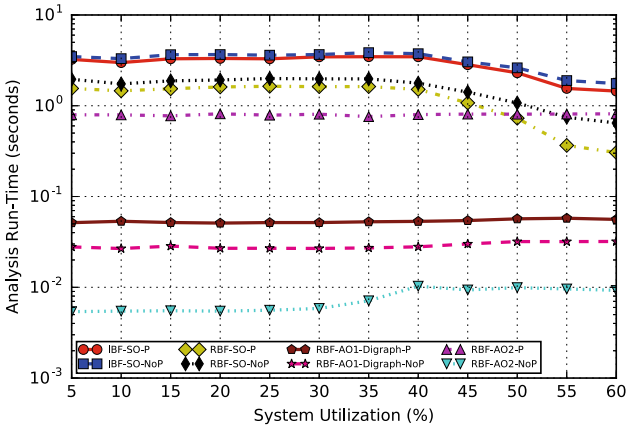
Fig. 23 The analysis runtimes of analysis methods with and without periodicity property where the event periods are set as *Type-V*

Type-V Similar to *Type-III*, but the base period of each FSM is randomly extracted from the set {1, 2, 5, 10, 20, 25, 50, 100, 1000, 2000}.

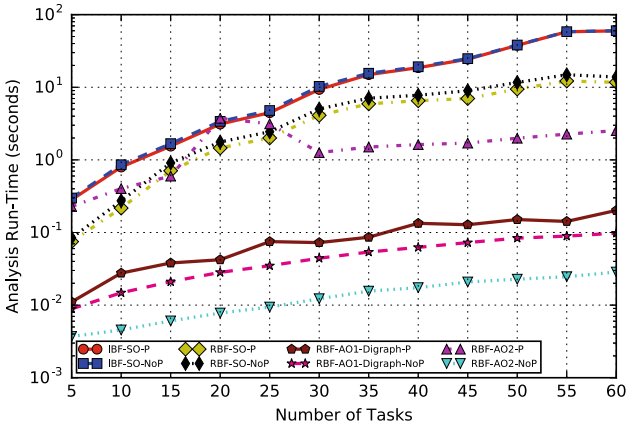
Type-VI Similar to *Type-III*, but the base period of each FSM is generated by the product of one to *three* factors, each randomly drawn from three harmonic sets (2,4), (6,12), (5,10).

Finally, transitions are connected such that the execution matrices are irreducible since the periodicity property can only be exploited for irreducible matrices (i.e., strongly connected digraphs).

Figures 23 and 24 show the runtimes of these analysis methods with and without the use of the periodicity property. The runtimes of *IBF-SO-NoP* and *RBF-SO-NoP* are higher than *IBF-SO-P* and *RBF-SO-P* respectively, which means that the calculation



(a) Runtime vs. system utilization ($n = 20$)



(b) Runtime vs. number of tasks ($U = 40\%$)

Fig. 24 The analysis runtimes of analysis methods with and without periodicity property where the event periods are set as *Type-VI*

of the IBF and RBF functions with static offsets can indeed be made more efficient. In contrast, the runtime of *RBF-AO2-NoP* is smaller than *RBF-AO2-P*, which means that the computation of the periodicity parameters is unlikely to improve the runtime of *RBF-AO2* when all the transformed digraphs are strongly-connected. We believe that this happens since in general the transformed digraph models contain a very large number of vertices and edges which result in very long computation times for the periodicity parameters. *RBF-AO1-Digraph-P* and *RBF-AO1-Digraph-NoP* have very close runtimes because only a small percentage (0.25%) of the transformed digraphs in *RBF-AO1-Digraph* are strongly-connected. Even if the digraphs in *RBF-AO1-Digraph* are modified by adding loose edges (see Sect. 4.5) to make them strongly connected,

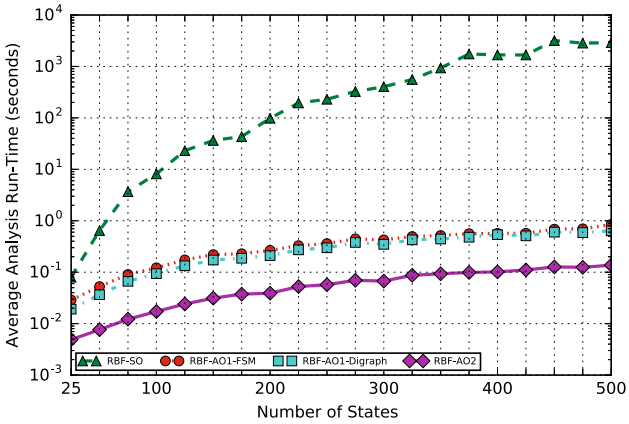


Fig. 25 Runtime versus number of states ($n = 20$ and $U = 40\%$)

the runtime of the periodicity analysis on these strongly-connected digraphs is still high.

6.4 Scalability

In the final set of experiments, we study the scalability of the approximate analysis methods. In the second set of experiments (Sect. 6.2), the analyses with the IBF functions have very similar accuracy to those using RBF functions in terms of evaluating the feasibility conditions, but with longer runtimes. Hence, we only focus on the four analyses methods with the RBF functions: *RBF-SO*, *RBF-AO1-FSM*, *RBF-AO1-Digraph*, and *RBF-AO2*. The experiment setting is similar to the second set of experiments, but each Stateflow has a number of states that is fixed and goes from 25 to 500. The number of tasks/FSMs is 20, and the system utilization is 40%. The event periods are as in *Type-III*. Because of the very long runtimes of these experimental runs, only 10 simulations for each number of states have been performed.

Figure 25 illustrates the impact of the number of states on the runtimes of the approximate analysis methods. As expected, although at the price of the worst accuracy, *RBF-AO2* has the smallest runtime among all methods. *RBF-AO1-Digraph* has a smaller runtime compared with *RBF-AO1-FSM*, but the two methods are very close in terms of required analysis runtime. Finally, *RBF-SO* has the longest runtime, since the calculation of $rbf[s, f)$ is more complicated than the one of $rbf(t)$ on the digraph models and it is sensitive to a large number of states (for any FSM with more than 25 states).

7 Conclusion

In this work, we presented the analysis framework for calculating the exact and approximate response times of synchronous finite state machine models through existing work

on a digraph model abstraction or through new methods based on state analysis. The analysis methods applied in the digraph model are assumed to have arbitrary offsets, while the new analysis considers periodic tasks with synchronized offsets by leveraging the state information. We performed a series of experiments to compare the digraph-based and state-based methods in terms of accuracy and runtime. Moreover, we demonstrated that it is possible to utilize the periodicity property to decrease the analysis time.

Acknowledgements This paper is partially supported by NSF Grant Nos. 1739318 and 1812963.

References

Baccelli F, Cohen G, Olsder GJ, Quadrat JP (1992) Synchronization and linearity: an algebra for discrete event systems. Wiley, Hoboken

Baruah SK (2003) Dynamic-and static-priority scheduling of recurring real-time tasks. Real Time Syst 24(1):93–128

Di Natale M, Zeng H (2012) Task implementation of synchronous finite state machines. In: Proceedings of the conference on design, automation, and test in Europe, pp 206–211

Esterel Technologies (2014) The Trusted Design Chain Company: scade suite. http://www.esterel-technologies.com/products/scade-system/

Fersman E, Krcal P, Pettersson P, Yi W (2007) Task automata: schedulability, decidability and undecidability. Int J Inf Comput 205(8):1149–1172

Floyd RW (1962) Algorithm 97: shortest path. Commun ACM 5(6):345

Gavalec M (2000) Linear matrix period in max-plus algebra. Linear Algebra Appl 307(1):167–182

Guan N, Gu C, Stigge M, Deng Q, Yi W (2014) Approximate response time analysis of real-time task graphs. In: IEEE real-time systems symposium (RTSS), pp 304–313

Harel D (1987) Statecharts: a visual formalism for complex systems. Sci Comput Program 8(3):231–274

Hartmann M, Arguelles C (1999) Transience bounds for long walks. Math Oper Res 24(2):414–439

Lee EA, Varaiya P (2011) Structure and interpretation of signals and systems. Addison Wesley, Boston

Lehoczky JP (1990) Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: 11th IEEE real-time systems symposium, vol 90, pp 201–209

Mathworks (1994) The mathworks simulink and stateflow user's manuals. http://www.mathworks.com

Molnárová M (2005) Generalized matrix period in max-plus algebra. Linear Algebra Appl 404:345–366

Natale MD, Guo L, Zeng H, Sangiovanni-Vincentelli A (2010) Synthesis of multi-task implementations of simulink models with minimum delays. IEEE Trans Ind Inf 6(4):637–651

Norström C, Wall A, Yi W (1999) Timed automata as task models for event-driven systems. In: sixth international conference on real-time computing systems and applications, pp 182–189

Peng C, Zeng H (2018) Response time analysis of digraph real-time tasks scheduled with static priority: generalization, approximation, and improvement. Real Time Syst 54(1):91–131

Stigge M, Ekberg P, Guan N, Yi W (2011) The digraph real-time task model. In: 16th IEEE real-time and embedded technology and applications symposium, pp 71–80

Stigge M, Yi W (2012) Hardness results for static priority real-time scheduling. In: 24th Euromicro conference on real-time systems (ECRTS), pp 189–198

Stigge M, Yi W (2013) Combinatorial abstraction refinement for feasibility analysis. In: 34th IEEE real-time systems symposium (RTSS), pp 340–349

Stigge M, Yi W (2015) Combinatorial abstraction refinement for feasibility analysis of static priorities. Real Time Syst 51(6):639–674

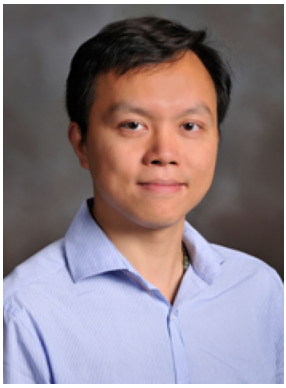
Stigge M, Yi W (2015) Graph-based models for real-time workload: a survey. Real Time Syst 51(5):602–636

Tindell K (1994) Adding time-offsets to schedulability analysis. In: Department of Computer Science, University of York, Report No. YCS-94-221

Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T, Mueller F, Puaut I, Puschner P, Staschulat J, Stenström P (2008) The worst-case execution-time problem: overview of methods and survey of tools. ACM Trans Embed Comput Syst 7(3):36:1–36:53

Young NE, Tarjant RE, Orlin JB (1991) Faster parametric shortest path and minimum-balance algorithms. Networks 21(2):205–221
Zeng H, Di Natale M (2011) Mechanisms for guaranteeing data consistency and flow preservation in autosar software on multi-core platforms. In: 6th IEEE international symposium on industrial and embedded systems, pp 140–149
Zeng H, Di Natale M (2013) Using max-plus algebra to improve the analysis of non-cyclic task models. In: 25th Euromicro conference on real-time systems (ECRTS), pp 205–214
Zeng H, Di Natale M (2015) Computing periodic request functions to speed-up the analysis of non-cyclic task models. Real Time Syst 51(4):360–394
Zeng H, Natale MD (2012) Schedulability analysis of periodic tasks implementing synchronous finite state machines. In: 24th Euromicro conference on real-time systems (ECRTS), pp 353–362
Zhao Y, Peng C, Zeng H, Gu Z (2017) Optimization of real-time software implementing multi-rate synchronous finite state machines. ACM Trans Embed Comput Syst 16(5s):175:1–175:21
Zhu Q, Deng P, Di Natale M, Zeng H (2013) Robust and extensible task implementations of synchronous finite state machines. In: Design, automation test in Europe conference exhibition (DATE), pp 1319–1324

Chao Peng received the B.E. and M.E. degrees in computer science and technology from Tsinghua University and National University of Defense Technology, China, in 2010 and 2012, respectively. He is currently working toward the Ph.D. degree in computer science and technology at National University of Defense Technology. His research interests include schedulability analysis and optimization of real-time systems, and interconnection network.



Haibo Zeng is with Department of Electrical and Computer Engineering at Virginia Tech, USA. He received his Ph.D. in Electrical Engineering and Computer Sciences from University of California at Berkeley. He was a senior researcher at General Motors R&D until October 2011, and an assistant professor at McGill University until August 2014. His research interests are embedded systems, cyber-physical systems, and real-time systems. He received four best paper/best student paper awards in the above fields.

Marco Di Natale received the Ph.D. degree from Scuola Superiore SantAnna and was a visiting researcher with the University of California, Berkeley in 2006 and 2008. He is a full professor in the Scuola Superiore SantAnna. He is currently visiting Fellow for the United Technologies corporation. He's been a researcher in the area of real-time and embedded systems for more than 20 years, being author or co-author of more than 200 scientific papers, winner of six best paper awards and one best presentation award. He is a senior member of the IEEE.