Formalizing Hierarchical Scheduling for Refinement of Real-Time Systems

Chenyang Zhu, Michael Butler, Corina Cirstea

Abstract

The Event-B formalism offers a stepwise development approach for managing complexity in system design. However, the existing work that extends Event-B models with discrete timing properties inadequately represents the communication and competition between concurrent tasks in concurrent systems. In this paper, we present the semantics of the parameterized real-time trigger-response properties of Event-B models based on timing invariants. We show a method of syntactically encoding parameterized real-time trigger-response properties in Event-B machines. To capture the concurrency between tasks, we distinguish end-to-end timing properties and scheduler-based timing properties from the perspective of different system design phases. We model end-to-end timing properties as parameterized timing properties and scheduler-based timing properties as unparameterized timing properties. A nondeterministic queue-based scheduling framework is proposed to replace end-to-end timing properties with scheduler-based timing properties. Additional gluing invariants are provided for this refinement. To demonstrate the usability of the framework, we formalize a two-level hierarchical scheduling system with local resource sharing managed by a time-division multiplexing global scheduler and two alternative local schedulers refined by the nondeterministic queue-based scheduling policy. Models are proved using the Rodin tool.

Keywords: Event-B, Refinement, Timing, Concurrency, Hierarchical Scheduling

1. Introduction

With the thriving growth of cyber-physical systems (CPS), much attention has been focused on simplifying the design of systems while maintaining their usability and dependability. Timing and concurrency are two critical features of CPS. The physical world evolves with time, and timing properties should be specified and verified together with the system to guarantee that the CPS is interacting with the physical world correctly [1]. Besides, with the advanced processors of CPS, multiple real-time tasks execute concurrently to achieve the goal of computation as a whole. A hard real-time system requires that all time-critical tasks meet their specified deadlines [2]. Formal modeling is used to manage the precision of specifications that describe system properties, including functional behavior, timing behavior, performance characteristics, and internal structure [3]. However, real systems are difficult to specify and verify. A stepwise modeling approach with abstraction and refinement can be adapted to master the complexity of specifications and requirements [4]. Abstraction helps to reason about a system with high-level goals while refinements add implementation details to the model and verify the consistency between different refinement levels.

Event-B is a formal method for system-level modeling and theorem-based analysis with refinements [5]. However, explicit notions of time and liveness are not supported in Event-B. Existing work that extends Event-B models with timing properties uses a trigger-response pattern to model discrete time [6]. The pattern sets timestamps for trigger and response events and uses a tick event to prevent the global clock from proceeding to a point at which time constraints between trigger and response events would be violated. This pattern, however, does not distinguish timing properties for different system design phases and cannot show the communication and competition between concurrent tasks in concurrent or distributed systems.

In this paper, we build on the results already presented in [7] that distinguish end-to-end timing properties and scheduler-based timing properties from differ-

ent system design phases. End-to-end timing properties are defined as timing properties from the system requirements, and they place discrete-time properties on individual tasks. However, in real-time systems, there are always several tasks running concurrently. The concurrent execution of the whole system must satisfy the timing properties of each task. To model the behavior of these concurrent tasks, we defined scheduler-based timing properties as concrete timing properties for the system design phase, which place discrete-time constraints on the scheduler that schedules the concurrent tasks. The scheduler-based timing properties should meet the end-to-end timing properties of each task. We propose a nondeterministic queue-based scheduling framework to model the behavior of the schedulers. Tasks are placed in a nondeterministic position in the queue, and once a task enters the queue, it cannot be postponed forever. Additional gluing invariants are provided to use the framework to replace end-to-end timing properties with scheduler-based timing properties.

This paper improves upon previous work [7] in several ways: (a) a formal definition of parameterized timing properties is presented together with the primary and auxiliary invariants required to prove that an Event-B model satisfies specific timing properties; (b) there is a refinement pattern for generating auxiliary invariants to replace parameterized timing properties to unparameterized timing properties; and (c) a two-level hierarchical scheduling system that allows compositional scheduling policies is formalized. As shown in Figure 1, [7] replaced end-to-end timing properties with scheduler-based timing properties by using a nondeterministic scheduling framework. Then, this framework is refined to two alternative scheduling policies, namely, first-in-first-out (FIFO) and deferrable priority-based (DPB) scheduler with an aging technique. This paper not only introduces time-division multiplexing (TDM) as the global scheduler but also combines the two scheduling policies into one refinement to show that the local schedulers are compatible with different scheduling policies.

The paper is organized as follows: Section 2 provides background definitions of Event-B and real-time trigger-response properties. Section 3 introduces the requirements and refinement strategy for the hierarchical system. Mutual exclu-

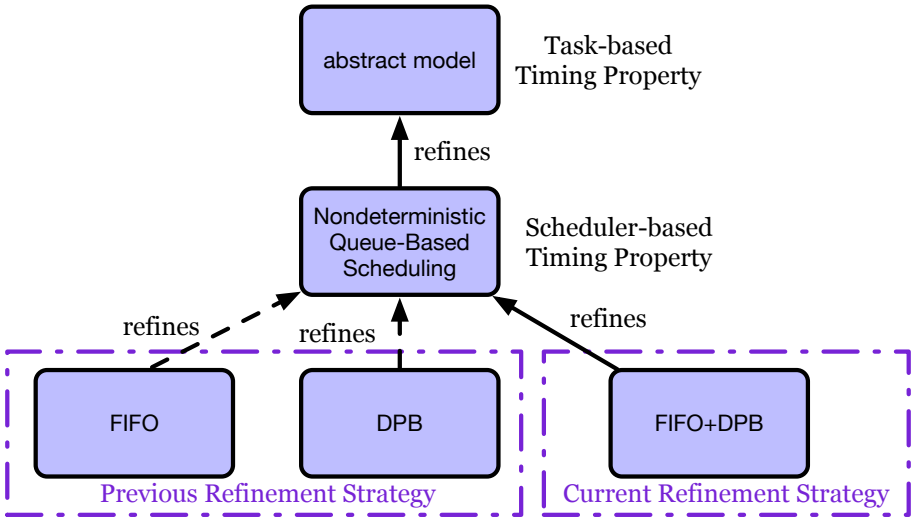

Figure 1: Improvement in the Current Refinement Structure

sion is used to model the time-division multiplexing and local resource sharing. Section 4 presents formalization and proofs for parameterized real-time trigger-response properties. In Section 5 we define end-to-end timing properties and scheduler-based timing properties based on parameterized and unparameterized timing properties. A two-level hierarchical scheduling system is formalized to demonstrate the usage of nondeterministic queue-based scheduling framework to replace timing properties. A general rule is shown in this section to replace parameterized timing properties with unparameterized timing properties. Section 6 summarizes some related work on modeling discrete-time in concurrent situations as well as refinement of timed systems. Section 7 summarizes the paper results and outlines future work.

2. Background Definitions

2.1. Event-B

Event-B [5] is a formal modeling method based on set theory, which is usually used for system-level modeling and analysis for discrete systems. Event-B, an evolution of the B-Method developed by Abrial [8], is greatly inspired by

the notion of action systems [9] and guarded commands [10]. A discrete model is made of *contexts* and *machines*. A *context* describes the static part of the model, which is specified with carrier sets s and constants c. *Constants* are defined with their properties and relationships by *axioms* and *theorems* [5]. A *machine* describes the dynamic behavior of the discrete model, which is specified with *variables* v and *events*. An *event* is described using guards and actions. The guards define the enabling condition under which the event can occur and the actions denote the way that the *variables* are modified by the event. The *variables* of a *machine* are defined by *invariants* $I(s, c, v)$ and *theorems*. *Machines* may see one or more *contexts* [5].

In general, an event *evt* can be represented by the form:

$$evt \triangleq \textbf{any} \quad p \quad \textbf{where} \quad G_{evt}(p, v) \quad \textbf{then} \quad S_{evt}(p, v, v') \tag{1}$$

An event might have a number of parameters p that are local to the event. The guard predicate $G_{evt}(p, v)$ defines the conditions under which the event *evt* can occur and the before-after action predicate $S_e(p, v, v')$ defines the way that *evt* changes the state in the system. The initial event does not have any parameters or guards, we write the event as:

$$init \triangleq \quad \textbf{then} \quad S_{init}(v) \tag{2}$$

Formal modeling is used to manage the precision of specifications, but real systems are difficult to specify and verify without abstractions [3]. Abstraction and refinement can help to master the complexity of requirements [4]. Abstraction makes it easier to control the complexity of the high-level model. Refinements add implementation details to the different levels of abstraction and guarantee that each refined model is consistent with the model being refined. Refinement of a system usually involves changing the variables of the system [11]. In Event-B machines, gluing invariants are used to link the variables in the refined model to the variables in the abstract model [12].

Given abstract event A with guards $G(p, v)$ and before-after predicates $S(p, v, v')$ and concrete event C with guards $H(p, w)$ and before-after predi-

cates $R(p, w, w')$, Abrial defines rule (3) to verify that C is a refinement of A: $A \sqsubseteq C$ [5] provided:

$$A \sqsubseteq C \triangleq I(v) \wedge J(v, w) \wedge H(p, w) \wedge R(p, w, w') \vdash G(p, v) \wedge \exists v' \cdot (S(p, v, v') \wedge J(v', w')) \tag{3}$$

where v and w are variables of A and C, respectively. $J(v, w)$ are the gluing invariants that relate the abstract and concrete variables. The rule expressed in equation (3) can be split into several proof obligations to verify the correctness of a refinement step, namely, invariant preservation (INV), feasibility (FIS), guard strengthening (GRD) and so on. These proof obligations guarantee that the proved properties in the abstract model are preserved in the concrete model. They can be generated and proved by the Rodin [12] platform. Here, we present the formalism of invariant preserving and guard strengthening in proof obligation (4) and proof obligation (5), respectively.

$$I(v) \wedge G(p, v) \wedge S(p, v, v') \vdash I(v') \tag{4}$$

$$I(v) \wedge J(v, w) \wedge H(p, w) \vdash G(p, v) \tag{5}$$

2.2. Semantics of Real-Time Trigger-Response Properties in Event-B

Event-B is a modeling language that supports modeling refinement but lacks explicit support for expressing and verifying timing and liveness properties [13]. Given a machine M with event labels E that represent the names of the events in M, our previous work provides the syntax of real-time trigger-response properties as Definition 2.1 [14] based on a trigger-response pair (T, R), where $T \subseteq E$ are trigger events, $R \subseteq E$ are response events, and $T \cap R = \varnothing$. We use $\mathbf{timing}(T, R, w, d)$ to define the timing properties between trigger events T and response events R, where w is the delay constraint and d is the deadline constraint. Multiple timing properties can be added to the machine. Figure 2 depicts the behavioral definition of timing. A machine that is extended with $\mathbf{timing}(T, R, w, d)$ contains a special *Tick* event to update the global clock.

The *Tick* events are constrained by the deadline specification so that it is disabled when one of the responses is missing its deadline.

Definition 2.1 (Real-Time Trigger-Response Property [14])**.** *A machine M with event labels E is extended with a real-time trigger-response property* ***timing***(T, R, w, d) *that consists of*

- *trigger events $T \subseteq E$;*
- *response events $R \subseteq E \wedge T \cap R = \varnothing$;*
- *a delay $w \in \mathbb{N}$ and a deadline $d \in \mathbb{N}$.*

[14] asserted that the behaviors of a machine with **timing**(T, R, w, d) should satisfy two properties: 1) the number of *Tick* events between trigger events T and response events R is bounded by the delay time w and deadline time d, and 2) the response event eventually occurs after the trigger event, and the trigger event does not recur within the trigger-response pair to avoid the recurring delay of response events. To formalize timing properties in discrete systems, Sarshogh and Butler proposed an approach that categorizes timing constraints by using a trigger-response pattern where trigger and response events are modeled as events in Event-B [6]. Inspired by their work, we formalize the untimed Event-B machine M with timing property **timing**(T, R, w, d) as Figure 2 by using three new variables. τ_T and τ_R are used to refer to the time at which the trigger or response events occurs, respectively. We also use the variable *clk* to denote the global clock. Additional guards and before-after predicates are added to the events to model real-time behaviors. The guard $clk \geq \tau_T + w$ of the response event guarantees that the response is disabled when the global clock has not passed the delay period. τ_T and τ_R are set to the current *clk* with the before-after predicate in trigger and response events. A new *Tick* event is added to the machine to increment the global clock. Guard $G_R(v) \Rightarrow clk + 1 - \tau_T \leq d$ of the *Tick* event constrains the global clock not to tick when the response event is about to miss its deadline.

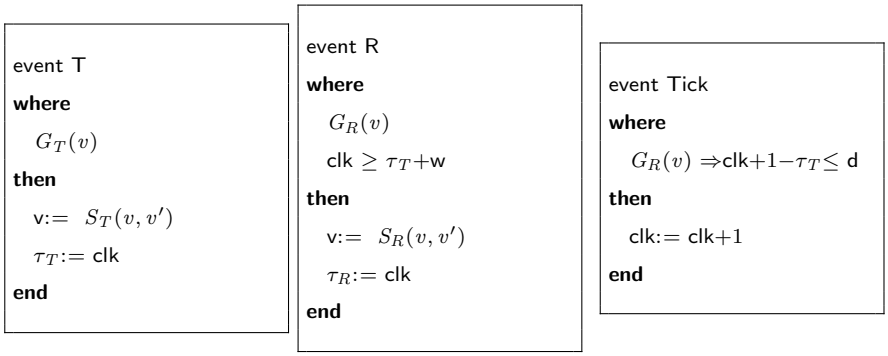

Figure 2: Formalization that Extends the Machine with timing property **timing**(T, R, w, d)

3. Formalizing a Hierarchical Scheduling System

In this section, we describe our requirements on the hierarchical scheduling system with local resource sharing managed by a time division multiplexing global scheduler and two alternative local schedulers.

3.1. Hierarchical Scheduling

With the emerging trend in real-time systems toward implementing functionalities in different levels on a shared platform, hierarchical scheduling (HS) systems are designed to use compatible schedulers to allocate CPU time so that all real-time applications meet their deadlines [15]. In this paper, we use a two-level HS system to illustrate the replacement from end-to-end timing properties to scheduler-based timing properties under the assumption of local resource sharing and no global resource sharing. Figure 3 shows a two-level HS that composes existing applications with different timing characteristics by using time-division multiplexing (TDM) as the global scheduler. The global scheduler decides which application should proceed and for how long. Then, each application uses its local scheduler to select which task to execute next. The TDM scheduler partitions a period into several time slots and assigns each of them to a single application. Take Figure 3 as an example. The TDM scheduler partitions the period of $50s$ into five time slots with $10s$ for each application. Then each application uses its local scheduler to schedule the concurrent tasks that

might have dependencies on each other. In this paper, we do not address the cases of global resource sharing in HS systems. Therefore tasks of the same application might have a shared code segment that accesses shared variables. We use critical sections to present the code segment, which have to be executed as an atomic action. Tasks of different applications do not share critical sections. Our main system requirements (SRs) are as follows:

SR-1 Tasks of different applications do not share critical sections.

SR-2 No more than one application can be in the same time slot at any time.

SR-3 No more than one task of the same application can be in its critical section at any time.

SR-4 Each application is assigned a time slot to run the tasks.

SR-5 If a task wishes to enter its critical section when the application is running, it will enter the critical section within a certain deadline.

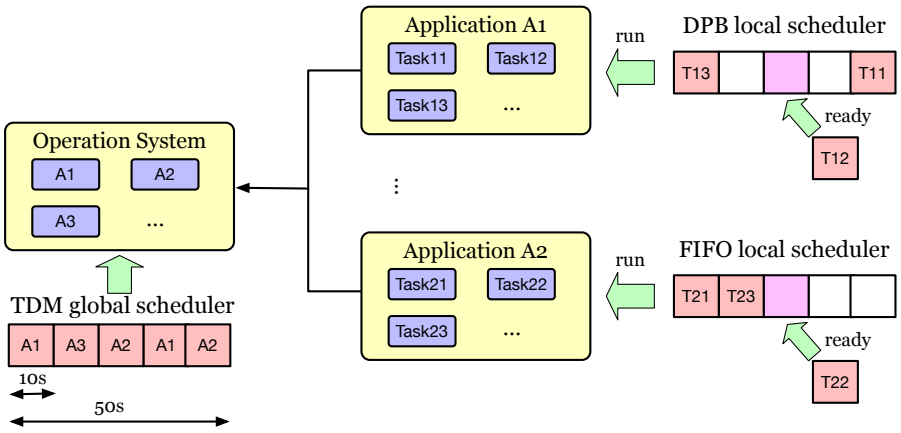

Figure 3: Two-level Hierarchical System

3.2. Refinement Strategy

In this section, we describe our refinement strategy to formalize the two-level HS system.

M_0 specifies the local resource sharing system.

M_1 introduces end-to-end timing properties for individual tasks and allocates applications with time slots.

M_2 formalizes the sequence order of tasks with a nondeterministic queue-based scheduling framework

M_3 replaces the end-to-end timing property into scheduler based timing property with nondeterministic queue-based scheduling framework.

M_4 refines the nondeterministic queue-based scheduling framework into two alternative local schedulers.

In the rest of the section, we explain these models and refinements in more detail and present part of our formalization.

3.3. Formalizing TDM and Local Resource Sharing with Mutual Exclusion

In the most abstract level M_0, a mutual exclusion model is proposed to address the dependencies within applications and tasks. The model guarantees that no two applications can be in the same time slot simultaneously and that no two tasks of the same application can be in the critical section simultaneously. We begin by defining the initial context in Figure 4. In the context, we define the carrier set *APPS* and *TASKS* of all applications and tasks in the HS system. We define *apps* as a total surjection from *TASKS* to *APPS* as a task cannot belong to two different applications. Additionally, the set of tasks of each application is finite, and its value is less than N.

In our initial model in Figure 5, we formalize the mutual exclusion model by using *app_run* and *task_run* variables. The variable *app_run* represents the set of applications that are currently running, whereas the variable *task_run* represents the set of tasks that are currently being executed. Variables *app_wait* and *task_wait* denote the set of applications and tasks that are ready to run. The invariants **inv0_1-inv0_4** formalize the relationships between the ready or waiting applications and tasks to the whole set. The invariants **inv0_5-inv0_6**

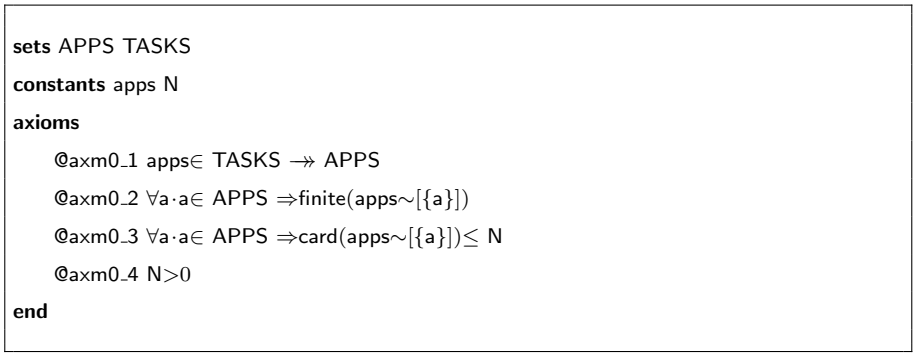

Figure 4: Initial Context c0

guarantee that only one application and one task can be executed at any time. **inv0_7** guarantees that the tasks that are waiting or being executed belong to the application that is being executing.

In the abstract model, we use three events, namely, *ready*, *run* and *finish*, to model the behaviors of applications and tasks ready to run, execute and finish executing, respectively. As the mutual exclusion model is similar for applications and tasks, we take the events of applications as an example to illustrate the model. Figure 5 gives the abstract mutual exclusion model for applications. Here, we use quantified variable a to represent an application. The event $APP\_READY(a)$ models the point at which application a is ready to run. Event $APP\_RUN(a)$ models the point at which the application starts running, while event $APP\_FINISH(a)$ models the point at which the application finishes running. We use the same approach to model the behaviors of tasks of an application. Since the ready and running tasks should belong to the application that is being executed, we add an additional guard $apps(t) \in app\_run$ to the $TASK\_READY$ event to guarantee **inv0_7**. Take Figure 3 as an example. Applications A_1, A_2 and A_3 wish to run, and the TDM global scheduler assigns each application a time slot to run. When application A_2 is running, it executes different tasks with the DPB scheduling policy. For example, tasks T_{21}, T_{23} and T_{22} wish to be executed with the $TASK\_READY$ event. Then the $TASK\_RUN(T_{21})$ event is executed since T_{21} is at the head of the queue.

After T_{21} finishes, $TASK\_RUN(T_{23})$ is executed.

Scheduling is used to allocate the processing time for concurrent tasks to maximize real-time performance [1]. To guarantee that all the high-level timing requirements of individual tasks are satisfied by the system, we begin by formalizing high-level timing properties with the parameterized real-time trigger-response properties defined in Section 4. Then, we use different scheduling policies to replace the end-to-end timing properties with scheduler-based timing properties. Additional gluing invariants are provided based on the proposed refinement rules.

4. Parameterized Real-Time Trigger-Response Properties

In Event-B models, parameters of events can be used to treat the concurrency of the system [16]. An Event-B machine first executes the initialization, followed by nondeterministically executing some enabled event. When modeling a concurrent system, instead of having separate atomic events for each task of the concurrent system, parameterized events can be used to model the atomic steps of each task. Additionally, in concurrent systems, parameterized trigger-response timing properties can be used to specify the timing properties of each task.

Given an Event-B machine M with event labels E, which contains a trigger-response pair (T, R), we extend our definition for the real-time trigger-response properties to parameterized trigger-response properties as in Definition 4.1. In this definition, we use X to denote the parameter value set. Assume that event e has a parameter p, $e.x$ is defined as the semantic label used to represent occurrence of event e with parameter p instantiated with value x. The semantics of a parameterized event e, with parameter p and operating on state variable v is represented by guard predicate $G_e(p, v)$ and before-after predicate $S_e(p, v, v')$. The semantic labels $T.x$ and $R.x$ are used to denote the occurrence of trigger events T or response events R with parameter p instantiated with value x,

invariants

@inv0_1 app_wait ⊆APPS

@inv0_2 app_run ⊆APPS

@inv0_3 task_wait ⊆TASKS

@inv0_4 task_run ⊆TASKS

@inv0_5 finite(app_run) ∧card(app_run)≤ 1

@inv0_6 finite(task_run) ∧card(task_run)≤ 1

@inv0_7 ∀t·t∈ (task_wait ∪task_run)⇒tasks(t)∈ app_run

event APP_READY
any a
where
  @grd1 a∈ APPS\app_wait
  @grd2 a∈ APPS\app_run
then
  @act1 app_wait:= app_wait∪{a}
end

event APP_RUN
any a
where
  @grd1 a∈ app_wait
  @grd2 app_run=∅
then
  @act1 app_wait:= app_wait\{a}
  @act2 app_run:= app_run ∪{a}
end

event APP_FINISH
any a
where
  @grd1 a∈ app_run
  @grd2 task_wait=∅ ∧task_run=∅
then
  @act1 app_run:= app_run\{a}
end

event TASK_READY
any t
where
  @grd1 t∈ TASKS\task_wait
  @grd2 t∈ TASKS\task_run
  @grd4 apps(t)∈ app_run
then
  @act1 task_wait:= task_wait∪{t}
end

event TASK_RUN
any t
where
  @grd1 t∈ task_wait
  @grd2 task_run=∅
then
  @act1 task_wait:= task_wait\{t}
  @act2 task_run:= task_run ∪{t}
end

event TASK_FINISH
any t
where
  @grd1 t∈ task_run
then
  @act1 task_run:= task_run\{t}
end

Figure 5: Initial Model with TDM and Local Resource Sharing

which are formally presented as 6a and 6b.

$$T.x \triangleq \textbf{where}\ \ G_T(x,v)\ \ \textbf{then}\ \ S_T(x,v,v') \tag{6a}$$

$$R.x \triangleq \textbf{where}\ \ G_R(x,v)\ \ \textbf{then}\ \ S_R(x,v,v') \tag{6b}$$

Definition 4.1 (Parameterized Trigger-Response Ordering Property)**.** *Given an Event-B machine M with events E and invariants $I(v)$, a parameterized trigger-response pair has the form (T, R, X) where $T \subseteq E$ are trigger events with a parameter p, $R \subseteq E$ are response events with a parameter p, and $T \cap R = \varnothing$. Given each $t \in T$ and $r \in R$, M satisfies sequential ordering for (T, R, X) provided the following conditions hold.*

1. $G_R(p,v) \vdash \neg G_T(p,v)$
2. $I(v) \wedge S_{init}(p,v,v') \vdash \neg G_R(p,v')$
3. $I(v) \wedge G_t(p,v) \wedge S_t(p,v,v') \vdash \neg G_t(p,v') \wedge G_R(p,v')$
4. $\forall e \cdot e \in E \setminus (T \cup R) \wedge I(v) \wedge G_e(v) \wedge S_e(v,v') \wedge G_R(p,v) \vdash G_R(p,v')$
5. $I(v) \wedge G_r(p,v) \wedge S_r(p,v,v') \vdash \neg G_r(p,v')$

In Definition 4.1, we use $G_T(p,v)$ to denote the disjunction of all $G_t(p,v)$ with $t \in T$ and $G_R(p,v)$ to denote the disjunction of all $G_r(p,v)$ with $r \in R$. When extending Event-B models with timing properties, the five conditions of Definition 4.1 are required to guarantee the sequential order of parameterized trigger-response pairs. Condition 1 specifies that when a trigger event is enabled, the response events must be disabled. Condition 2 requires that the initial event disables the response events. Condition 3 requires that once a trigger event $t.x$ occurs, it disables itself and enables some response event $r.x$. Condition 4 requires that each $e \in E \setminus (T \cup R)$ preserves the predicate $G_R(p,v)$. Condition 5 requires that each response event r disables itself.

Similar to Definition 2.1, we define parameterized real-time trigger-response properties in Definition 4.2. w and d are total functions from the parameter value set X to the natural number set $\mathbb{N}$, which define the delay and deadline for each specific parameter instance.

Definition 4.2 (Parameterized Real-Time Trigger-Response Property)**.** *A parameterized real-time trigger-response property has the form:*

$$\textbf{\textit{all}}\ x\ \textbf{\textit{in}}\ X\ \ \textbf{\textit{timing}}(T.x, R.x, w(x), d(x))$$

where delay $w \in X \to \mathbb{N}$ *and deadline* $d \in X \to \mathbb{N}$.

Similar to behaviors of a machine with unparameterized timing properties, the behaviors of a model with parameterized real-time trigger-response property **all** x **in** X **timing**$(T.x, R.x, w(x), d(x))$ satisfies two properties: 1) the number of *Tick* events between each trigger event $T.x$ and its corresponding response event $R.x$ is bounded by the delay time $w(x)$ and deadline time $d(x)$; 2) no two occurrences of $T.x$ are allowed without an occurrence of $R.x$ in between. In Event-B models, we construct Inv 1 to capture the fact that each unique trigger-response pair $(T.x, R.x)$ is bounded by the deadline time $d(x)$. Inv 1 formalizes the safety property that when a response event occurs, the time between trigger and response event should be bounded by $d(x)$.

$$\forall x \cdot x \in X \land \tau_T(x) \leq \tau_R(x) \Rightarrow \tau_R(x) - \tau_T(x) \leq d(x) \qquad \text{(Inv 1)}$$

Figure 6 shows the formalization we use that extends the untimed machine with (T, R, X) to timed machine with parameterized timing properties **all** x **in** X **timing**$(T.x, R.x, w(x), d(x))$ for each trigger-response pair $(t.x, r.x)$ where $t \in T$ and $r \in R$. The formalism is similar to the one used to extend the machine with unparameterized timing properties. The timestamp variable $\tau_t \in X \to \mathbb{N}$ and $\tau_r \in X \to \mathbb{N}$ are set by the before-after predicate in the trigger and response events respectively. Additional constraints relating to $\tau_t(x)$ and $\tau_r(x)$ are imposed on the response event R and *Tick* event. Here $G_{Tick}(v)$ in equation (7) denotes the guard of *Tick* event. In the $G_t(p, v)$ and $G_r(p, v)$ predicate, we substitute the parameter p with x.

$$G_{Tick}(v) \triangleq \forall x \cdot x \in X \land G_R(x, v) \Rightarrow clk + 1 \leq \tau_T(x) + d(x) \qquad (7)$$

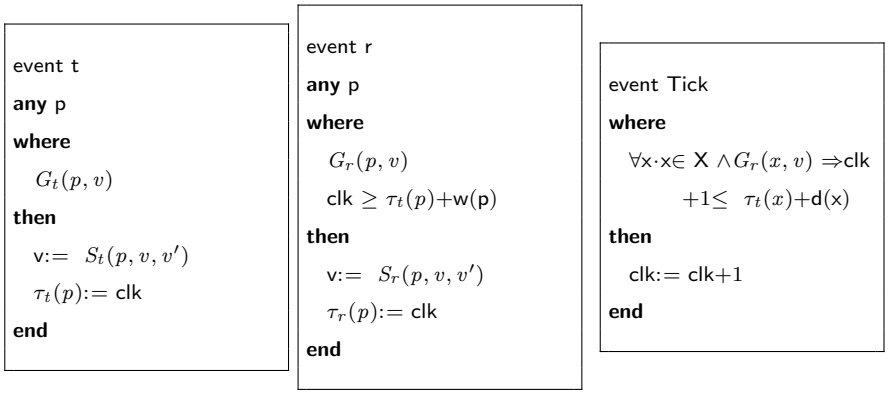

Figure 6: Formalization that Extends the Machine with parameterized timing properties **all** x **in** X **timing**($T.x, R.x, w(x), d(x)$) for each trigger-response pair ($t.x, r.x$) where $t \in T$ and $r \in R$

With the formalization shown in Figure 6, the response is constrained by the guard $clk \geq \tau_t(x) + w(x)$, which guarantees that response event $R.x$ can occur only after the delay time $w(x)$ has passed. No additional gluing invariants are required for the delay constraint. Thus, in this paper we focus on deadlines and not the delays. Inv 2 is an auxiliary invariant that can be used to prove Inv 1, which defines the state where the trigger event occurred while the response event has not occurred, and the time between the current time and the timestamp of the trigger event should also be bounded by $d(x)$. We first construct Lemma 4.3 to prove that Inv 2 is preserved by all the events of the machine with timing encoded.

$$\forall x \cdot x \in X \Rightarrow (G_R(x, v) \Rightarrow clk - \tau_T(x) \leq d(x)) \qquad \text{(Inv 2)}$$

Lemma 4.3. *Given the formalization in Figure 6 that extends the machine with* ***all*** *x* ***in*** *X* ***timing***($T.x, R.x, w(x), d(x)$)*, Inv 2 is preserved by all the events in the extended machine.*

Proof. Given Inv 2, the only events that change the variables of the invariant are trigger events $t \in T$ and *Tick* event. Thus in the proof we mainly examine these two events.

For the trigger events, preservation of Inv 2 is represented as follows:

$$\frac{\begin{array}{c} H_0 : \forall x \cdot x \in X \Rightarrow (G_R(x, v) \Rightarrow clk - \tau_T(x) \leq d(x)) \\ H_1 : I(v) \wedge G_t(p, v) \wedge S_t(p, v, v') \end{array}}{G : \forall x \cdot x \in X \Rightarrow (G_R(x, v') \Rightarrow clk' - \tau'_T(x) \leq d(x))}$$

Assume $x \in X$ and $G_R(x, v')$, we have to prove $clk' - \tau'_T(x) \leq d(x)$.

$$\begin{array}{ll} & clk' - \tau'_T(x) \leq d(x) \\ & \quad \langle \textbf{Case } \text{x=p} \rangle \\ & clk' - \tau'_T(p) \leq d(p) \\ \equiv & \quad \langle S_t(p, v, v') : clk' = clk; \tau'_T = (\tau_T \lhd\!\!+ \{p \mapsto clk\}) \rangle \\ & clk - (\tau_T \lhd\!\!+ \{p \mapsto clk\})(p) \leq d(p) \\ \equiv & \quad \langle (\tau_T \lhd\!\!+ \{p \mapsto clk\})(p) = clk \rangle \\ & clk - clk \leq d(p) \\ \equiv & \quad \langle 0 \leq d(p) \rangle \\ & \top \\ & \quad \langle \textbf{Case } \ x \neq p \rangle \\ & clk' - \tau'_T(x) \leq d(x) \\ \equiv & \quad \langle S_t(p, v, v') : clk' = clk; \tau'_T = (\tau_T \lhd\!\!+ \{p \mapsto clk\}) \rangle \\ & clk - \tau_T(x) \leq d(x) \\ \Leftarrow & \quad \langle H_0 \rangle \\ & \top \end{array}$$

For the *Tick* event, the preservation of Inv 2 is represented by:

$$\frac{\begin{array}{c} H_0 : \forall x \cdot x \in X \Rightarrow (G_R(x, v) \Rightarrow clk - \tau_T(x) \leq d(x)) \\ H_1 : \forall x \cdot x \in X \wedge G_R(x, v) \Rightarrow clk + 1 - \tau_T(x) \leq d(x) \\ H_2 : clk' = clk + 1 \end{array}}{G : \forall x \cdot x \in X \Rightarrow G_R(x, v') \Rightarrow clk' - \tau'_T(x) \leq d(x)}$$

$$
\begin{aligned}
& \forall x \cdot x \in X \Rightarrow G_R(x, v') \Rightarrow clk' - \tau'_T(x) \leq d(x) \\
\equiv \quad & \langle clk' = clk + 1; \tau'_T(p) = \tau_T(p) \rangle \\
& \forall x \cdot x \in X \Rightarrow G_R(x, v') \Rightarrow clk + 1 - \tau_T(x) \leq d(x) \\
\equiv \quad & \langle H_1 \rangle \\
& \top
\end{aligned}
$$

□

In the formalization, *clk* is increased by the *Tick* event only. The difference between $\tau_T(p)$ and $\tau_R(p)$ stands for the number of *Tick* events between $T.x$ and $R.x$. When $\tau_T(p) \leq \tau_R(p)$, the response event $R.x$ must have occurred after the trigger event. Therefore, when Inv 1 is preserved by the model, the behavior of the model satisfies the properties of parameterized timing properties. Hence, we construct Theorem 4.4 to prove that Inv 1 is preserved by all the events in a machine, which shows that the behavior of the model satisfies the parameterized real-time trigger-response properties.

Theorem 4.4. *Given the formalization in Figure 6 that extends machine M with* ***all*** *x* ***in*** *X* ***timing***$(T.x, R.x, w, d)$*, invariant Inv 1 is preserved by all the events in the extended machine.*

Proof. Given Inv 1, the only events that changes the τ_T variables and τ_R variables are t and r events where $t \in T$ and $r \in R$. Thus in the proof we only examine these two events.

For the trigger events, the preservation of Inv 1 is represented by:

$$
\frac{\begin{array}{c} H_0 : \forall x \cdot x \in X \wedge \tau_T(x) \leq \tau_R(x) \Rightarrow \tau_R(x) - \tau_T(x) \leq d(x) \\ H_1 : I(v) \wedge G_t(p, v) \wedge S_t(p, v, v') \end{array}}{G : \forall x \cdot x \in X \wedge \tau'_T(x) \leq \tau'_R(x) \Rightarrow \tau'_R(x) - \tau'_T(x) \leq d(x)}
$$

$$
\begin{aligned}
& \forall x \cdot x \in X \wedge \tau'_T(x) \leq \tau'_R(x) \Rightarrow \tau'_R(x) - \tau'_T(x) \leq d(x) \\
\Leftarrow \quad & \langle x \text{ is not free in } H_0 \rangle \\
& \tau'_T(x) \leq \tau'_R(x) \Rightarrow \tau'_R(x) - \tau'_T(x) \leq d(x) \\
& \langle \textbf{Case } \text{x=p} \rangle \\
& \tau'_T(p) \leq \tau'_R(p) \Rightarrow \tau'_R(p) - \tau'_T(p) \leq d(p) \\
\equiv \quad & \langle S_t(p, v, v') : \tau'_R = \tau_R; \tau'_T = (\tau_T \lhd\!\!+ \{p \mapsto clk\}) \rangle \\
& \tau'_T(p) \leq \tau'_R(p) \Rightarrow \tau_R(p) - clk \leq d(p) \\
\equiv \quad & \langle 0 \leq d(p) \wedge \tau_R(p) \leq clk \rangle \\
& \tau'_T(p) \leq \tau'_R(p) \Rightarrow \top \equiv \top \\
& \langle \textbf{Case } x \neq p \rangle \\
& \tau'_T(x) \leq \tau'_R(x) \Rightarrow \tau'_R(x) - \tau'_T(x) \leq d(x) \\
\equiv \quad & \langle S_t(p, v, v') : \tau'_R = \tau_R; \tau'_T = (\tau_T \lhd\!\!+ \{p \mapsto clk\}) \rangle \\
& \tau_T(x) \leq \tau_R(x) \Rightarrow \tau_R(x) - \tau_T(x) \leq d(x) \\
\Leftarrow \quad & \langle H_0 \rangle \\
& \top
\end{aligned}
$$

315

For the response events, the preservation of Inv 1 is represented by:

$$
\frac{\begin{array}{c} H_0 : \forall x \cdot x \in X \wedge \tau_T(x) \leq \tau_R(x) \Rightarrow \tau_R(x) - \tau_T(x) \leq d(x) \\ H_1 : I(v) \wedge G_R(p, v) \wedge S_r(p, v, v') \end{array}}{G : \forall x \cdot x \in X \wedge \tau'_T(x) \leq \tau'_R(x) \Rightarrow \tau'_R(x) - \tau'_T(x) \leq d(x)}
$$

$$
\begin{array}{ll}
 & \forall x \cdot x \in X \wedge \tau'_T(x) \leq \tau'_R(x) \Rightarrow \tau'_R(x) - \tau'_T(x) \leq d(x) \\
\Leftarrow & \quad \langle x \text{ is not free in } H_0 \rangle \\
 & \tau'_T(x) \leq \tau'_R(x) \Rightarrow \tau'_R(x) - \tau'_T(x) \leq d(x) \\
 & \quad \langle \textbf{Case} \quad x = p \rangle \\
 & \tau'_T(p) \leq \tau'_R(p) \Rightarrow \tau'_R(p) - \tau'_T(p) \leq d(p) \\
\equiv & \quad \langle S_t(p, v, v') : \tau'_T = \tau_T; \tau'_R = (\tau_R \lhd\!\!+ \{p \mapsto clk\}) \rangle \\
 & \tau_T(p) \leq \tau_R(p) \Rightarrow clk - \tau_T(p) \leq d(p) \\
\Leftarrow & \quad \langle \text{strengthen predicate} \rangle \\
 & clk - \tau_T(p) \leq d(p) \\
\Leftarrow & \quad \langle \text{ Inv 2:} G_R(p, v) \Rightarrow clk - \tau_T(p) \leq d(p) \rangle \\
 & G_R(p, v) \\
\Leftarrow & \quad \langle H_1 \rangle \\
 & \top \\
 & \quad \langle \textbf{Case} \quad x \neq p \rangle \\
 & \tau'_T(x) \leq \tau'_R(x) \Rightarrow \tau'_R(x) - \tau'_T(x) \leq d(x) \\
\equiv & \quad \langle S_t(p, v, v') : \tau'_T = \tau_T; \tau'_R = (\tau_R \lhd\!\!+ \{p \mapsto clk\}) \rangle \\
 & \tau_T(x) \leq \tau_R(x) \Rightarrow \tau_R(x) - \tau_T(x) \leq d(x) \\
\Leftarrow & \quad \langle H_0 \rangle \\
 & \top
\end{array}
$$

□

5. Formalizing Hierarchical Scheduling with Timing Properties

5.1. End-to-end Timing Properties

In this paper, we define end-to-end timing properties as high-level timing properties to specify the time constraints of individual tasks in the HS system. Based on the abstract model that specifies no two tasks of the same application can be in the critical section simultaneously, we introduce parameterized

real-time trigger-response properties to the model as end-to-end timing properties (8a) and (8b) in the first refinement. Timing property (8a) ensures that each application occupies the CPU resource for *app_ddl*. For each a in the $APPS$ set, the time between the occurrence of $APP\_READY.a$ and $APP\_RUN.a$ is bounded below by 0 and above by *app_ddl*. Timing property (8b) guarantees that if a task wishes to enter its critical section, it will enter the critical section within the specified timing property *task_ddl*. Figure 7 shows the refinement with end-to-end timing property for each application. The timing property of each task can be modeled with the same pattern. We use $@axm1\_3$ to guarantee that all tasks of one application finish executing within the application execution time. $at(a)$ models the timestamp at which application a wishes to occupy CPU time. $ar(a)$ models the timestamp at which application a gets the CPU to run tasks. $tt(t)$ models the timestamp at which task t wishes to enter the critical section. $tr(t)$ models the timestamp task entering the critical section. $@inv1\_6$ and $@inv1\_8$ capture the end-to-end timing property based on parameterized real-time trigger-response property semantics. Since timed machine of Figure 7 is constructed according to the approaches of Section 4, from Theorem 4.4 we have that $@inv1\_6$ to $@inv1\_9$ are preserved.

$$\textbf{all}\ \text{a}\ \textbf{in}\ APPS\ \textbf{timing}(APP\_READY.a, APP\_RUN.a, 0, app\_ddl) \quad (8a)$$

$$\textbf{all}\ \text{t}\ \textbf{in}\ TASKS\ \textbf{timing}(TASK\_READY.t, TASK\_RUN.t, 0, task\_ddl) \quad (8b)$$

5.2. Replacing End-To-End Timing Property with Scheduler-Based Timing Properties

In concurrent computing, concurrent tasks are executed by interleaving the execution steps of each task, which models tasks in the outside world that happen concurrently. In real-time systems, scheduling is used to make sure that all tasks meet their deadlines [1]. A scheduler is used to allocate the resource to a task for some time.

constants app_ddl task_ddl

axioms

@axm1_1 app_ddl>0

@axm1_2 task_ddl>0

@axm1_3 N $*$ task_ddl$\leq$ app_ddl

invariants

@inv1_6 $\forall$a$\cdot$at(a)$\leq$ ar(a)$\Rightarrow$ ar(a)$-$at(a)$\leq$ app_ddl

@inv1_7 $\forall$a$\cdot$a$\in$ app_wait$\Rightarrow$clk$-$at(a)$\leq$ app_ddl

@inv1_8 $\forall$t$\cdot$tt(t)$\leq$ tr(t)$\Rightarrow$ tr(t)$-$tt(t)$\leq$ task_ddl

@inv1_9 $\forall$t$\cdot$t$\in$ task_wait$\Rightarrow$clk$-$tt(t)$\leq$ task_ddl

event APP_READY **extends** APP_READY

then

@act2 at(a):$=$ clk

end

event APP_RUN **extends** APP_RUN

then

@act3 ar(a):$=$ clk

end

event TICK

where

@grd1 $\forall$a$\cdot$a$\in$ app_wait $\Rightarrow$clk$+1-$at(a) $\leq$ app_ddl

@grd2 $\forall$t$\cdot$t$\in$ task_wait $\Rightarrow$clk$+1-$tt(t) $\leq$ task_ddl

then

@act1 clk:$=$ clk$+1$

end

Figure 7: First Refinement with End-to-end Timing Properties

In the next refinement of the case study, we specify two scheduler-based timing properties with unparameterized timing properties (9a) and (9b). Property (9a) requires that when the system is idle, one of the requesting tasks will enter the critical section within *idletime*. Specifically, there are two cases that trigger the scheduling of the enter event: 1) a task wishes to enter, and both the queue and the critical section are empty, and 2) some task leaves the critical section, and there is some other task waiting in the queue. Observe here that events can act as timing triggers only under certain conditions; e.g., the *wish* event is only a timing trigger when the queue and critical section are empty. To address such conditional triggers, we split the event into separate refinements representing separate cases. We refine the *TASK_READY* event into a *TASK_READY_EMPTY* event, enabled when the first condition holds, and a *TASK_READY_NONEMPTY* event, enabled when the second condition holds. Similarly, we split the *TASK_FINISH* event into a *TASK_FINISH_NONEMPTY* event, enabled when the second condition holds, and a *TASK_FINISH_IDLE* event, enabled when the last task in the queue finish executing. The events *TASK_READY_EMPTY* and *TASK_FINISH_NONEMPTY* are therefore used as trigger events in (9a), whereas the response event *TASK_RUN* is the event modeling entering the critical section.

(9b) requires that once a task enters the critical section, it will leave the critical section within *runtime*. Therefore, the trigger event is the *TASK_RUN* event, whereas response events should correspond to leaving the critical section. As the latter is now captured by *two* events, there are two response events in (9b). Here, *TASK_FINISH_NONEMPTY* implies that when some tasks finish executing, others are still waiting in the queue. *TASK_FINISH* denotes the situation in which the last task in the queue finishes executing.

$$\begin{cases} \textbf{timing}(\mathit{TASK\_READY\_EMPTY}, \mathit{TASK\_RUN}, 0, \mathit{idletime}) \\ \textbf{timing}(\mathit{TASK\_FINISH\_NONEMPTY}, \mathit{TASK\_RUN}, 0, \mathit{idletime}) \end{cases} \tag{9a}$$

$$\begin{cases} \textbf{timing}(\mathit{TASK\_RUN},\ \mathit{TASK\_FINISH\_NONEMPTY}, 0, \mathit{runtime}) \\ \textbf{timing}(\mathit{TASK\_RUN},\ \mathit{TASK\_FINISH\_IDLE}, 0, \mathit{runtime}) \end{cases} \tag{9b}$$

Parameterized timing properties describe time bounds over quantified trigger-response pairs. This method lacks an adequate representation of the conflicts of timing properties resulting from the competition between concurrent trigger-response pairs. Global schedulers are used to schedule the tasks to execute. We employ unparameterized timing properties $\textbf{timing}(P, Q, 0, wt)$ to represent the timing properties of global schedulers, which can be used to replace end-to-end timing property **all** x **in** X $\textbf{timing}(T.x, R.x, 0, d(x))$ with additional constraints. In this paper, we present a pattern for replacing an end-to-end timing property with a collection of scheduler-based timing properties.

In the refinement, the timing properties at the abstract level could be replaced by other timing properties at the concrete level. One end-to-end timing property **all** x **in** X $\textbf{timing}(T.x, R.x, 0, d(x))$ could be replaced by n scheduler-based timing properties presented in (10). Take HS system as an example, the abstract timing property (8b) is replaced by a set of concrete timing properties (9a) and (9b). We construct a pattern to replace the timing properties so that the refinement is preserved.

$$\begin{cases} \textbf{timing}(P_1, Q_1, 0, wt_1) \\ \textbf{timing}(P_2, Q_2, 0, wt_2) \\ ... \\ \textbf{timing}(P_n, Q_n, 0, wt_n) \end{cases} \tag{10}$$

As presented in Figure 2 and Figure 6, the guard of *Tick* event is determined by the timestamps of trigger events and the predicate G_R. Thus we use the relation between the timestamps of P and T and the relation between abstract event set R and concrete event sets $Q_1, Q_2, \ldots, Q_n$ to construct the gluing invariants that relate the end-to-end timing properties and scheduler-based timing properties.

When replacing the end-to-end timing properties with scheduler-based timing properties, the schedulers determine the waiting time of each task based on its scheduling policy. Thus, for each scheduler-based timing property **timing**$(P_i, Q_i, 0, wt_i)$, where $i \in 1 \dots n$, that replaces the end-to-end timing property **all** x **in** X **timing** $(T.x, R.x, 0, d(x))$, we use a function $g_i(x)$ to denote the maximum waiting time of each task x for other tasks to run under the specific scheduling policy. This $g_i(x)$ can be provided by modelers based on different real-time scheduling specifications, which can be used to relate the timestamps of P and T. We use τ_T and τ_R to represent the timestamps of events T and R, and τ_P and τ_Q to represent the timestamps of events P and Q respectively. As shown in the formalism presented in Figure 2 and Figure 6, the timestamps are updated when the corresponding trigger and response events occur. We require that $g_i(x)$ of each x is updated by the trigger event P simultaneously with τ_P. It is obvious that $g_i(x)$ should satisfy the condition presented in Equation 11, which requires that maximum waiting time of each task x should be less than its deadline $d(x)$.

$$\forall x \cdot x \in X \Rightarrow 0 \leq g_i(x) \leq d(x) \tag{11}$$

Figure 8 shows the time diagram of an example that replaces the parameterized timing property **all** x **in** X **timing**$(T.x, R.x, 0, d(x))$ with a single unparameterized timing property **timing**$(P, Q, 0, wt)$. Assume that tasks are being executed in the order $p_1 \rightarrow p_2 \rightarrow p_3$ and that the worst-case execution time is $wt \in \mathbb{N}$. p_3 has to wait for p_1 and p_2 to finish being executed before it is executed. In the refined model, some unparameterised event can be used to replace the parameterized events. For example, in the abstract model we use $T.x$ to denote the event that some task x wishes to run. In the refined model, we use event P to denote the event at which the scheduler allows some task to run and Q to denote the event that stops executing a running task. When τ_P is initially updated, the maximum waiting time for p_2 is wt and the maximum waiting time for p_3 is $2 * wt$. Thus $g(p_2) = wt$ and $g(p_3) = 2 * wt$. After p_1 finishes executing, τ_P is updated again by the scheduler. In this case p_2 does

not need to wait and the maximum waiting time for p_3 is wt. After p_2 finishes executing, $g(p_3) = 0$.

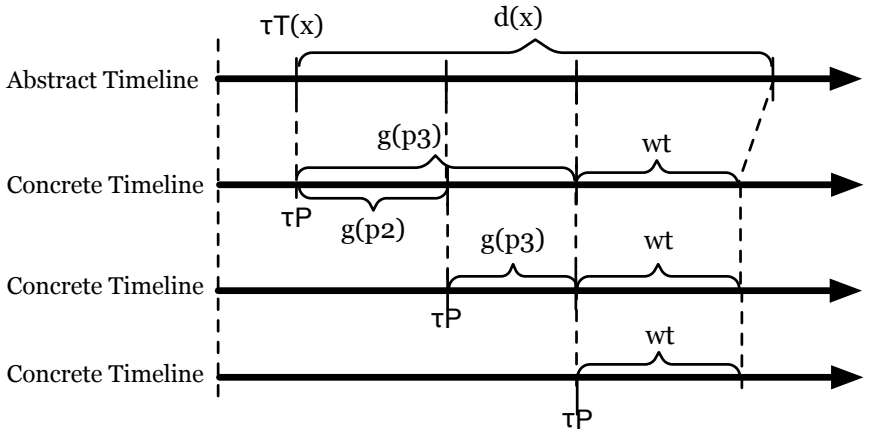

Figure 8: Time Diagram of Replacement of Parameterized Timing Properties to Unparameterized Timing Properties

Besides the relation between timestamps of trigger events, we also show the relation between the guards of the response events. For each task $x \in X$, we want to show the response events $R.x$ could be represented by some response event Q_i of the scheduler where $i \in 1 \ldots n$. Based on the above assumptions, we construct Theorem 5.1 to provide gluing invariants that relate the scheduler-based timing properties and end-to-end timing properties. We mainly show that the GRD proof obligation of *Tick* event is discharged by the additional conditions provided in Theorem 5.1.

Theorem 5.1. *Given a machine M with end-to-end timing property **all** x **in** X **timing** $(T.x, R.x, 0, d(x))$ to be refined with a machine N with scheduler-based timing property presented in (10). Given that a function $g_i(x)$ that represents the waiting time of each task x under the specific scheduling policy is provided for each **timing**$(P_i, Q_i, 0, wt_i)$ where $i \in 1 \ldots n$. The scheduler-based timing properties replace the end-to-end timing property when:*

1. $\forall x \cdot x \in X \Rightarrow (\tau_{P_i} + g_i(x) + wt_i \leq \tau_T(x) + d(x))$ *is a valid invariant for each **timing**$(P_i, Q_i, 0, wt_i)$ where $i \in 1 \ldots n$;*

2. $\forall p \cdot G_R(p, v) \Rightarrow H_{Q_1}(w) \vee H_{Q_2}(w) \ldots \vee H_{Q_n}(w)$, *where* G_R *is the guard of the response event of the end-to-end timing property, and* H_{Q_i} *is the guard of the response event of a timing property* ***timing***$(P_i, Q_i, 0, wt_i)$ *that replaces the abstract timing property.*

Proof. In this proof we want to show that the GRD proof obligation for the *Tick* event, formally presented as $I(v) \wedge J(v, w) \wedge H_{Tick}(w) \vdash G_{Tick}(v)$.

$$\frac{\begin{array}{c} H_0 : G_R(p, v) \Rightarrow H_{Q_1}(w) \vee H_{Q_2}(w) \ldots \vee H_{Q_n}(w) \\ H_1 : (H_{Q_1}(w) \Rightarrow clk + 1 \leq \tau_{P_1} + wt_1) \wedge (\forall x \cdot x \in X \Rightarrow (\tau_{P_1} + g_1(x) + wt_1 \leq \tau_T(x) + d(x))) \\ H_2 : (H_{Q_2}(w) \Rightarrow clk + 1 \leq \tau_{P_2} + wt_2) \wedge (\forall x \cdot x \in X \Rightarrow (\tau_{P_2} + g_2(x) + wt_2 \leq \tau_T(x) + d(x))) \\ \ldots \\ H_n : (H_{Q_n}(w) \Rightarrow clk + 1 \leq \tau_{P_n} + wt_n) \wedge (\forall x \cdot x \in X \Rightarrow (\tau_{P_n} + g_n(x) + wt_n \leq \tau_T(x) + d(x))) \end{array}}{G : \forall x \cdot x \in X \Rightarrow (G_R(x, v) \Rightarrow clk + 1 \leq \tau_T(x) + d(x))}$$

Assume $x \in X$ and $G_R(x, v)$. We have to show $clk + 1 \leq \tau_T(x) + d(x))$

$$\begin{array}{ll} & clk + 1 \\ \leq & \quad \langle H_0 : \exists i \cdot H_{Q_i} \text{ with } H_i \rangle \\ & \tau_{P_i} + wt_i \\ \leq & \quad \langle g_i(x) \geq 0 \rangle \\ & \tau_{P_i} + g_i(x) + wt_i \\ \leq & \quad \langle H_0 : \exists i \cdot H_{Q_i} \text{ with } H_i \text{ and } x \in X \rangle \\ & \tau_T(x) + d(x) \end{array}$$

□

5.3. Nondeterministic Queue-Based Scheduling

In our HS system that replaces the end-to-end deadline constraint with scheduler-based deadline constraints, we propose a nondeterministic queue-based scheduling framework to address the scheduling order of the sequential execution of a set of events. In this framework, a queue is used to manage the ready tasks. Each task is formally assigned a position in the queue: $queue \in wait \rightarrowtail$

$(0..N-1)$. When one task is ready, it is nondeterministically assigned a natural number that is not in the range of the queue. Only the task in the front of the queue ($min(ran(queue))$) can get the resource to run. The dequeue operation will decrease the indexes of all other tasks in the queue by the index of the front task plus one ($min(ran(queue)) + 1$) to guarantee that once a task is added to the queue, and it will eventually have the opportunity to run. In the second refinement, we use this nondeterministic queue-based scheduling framework to impose an order on the execution of the concurrent tasks. This refinement prevents a task from entering the critical section endlessly while also not allowing other tasks to enter the critical section. The second refinement is shown in Figure 9.

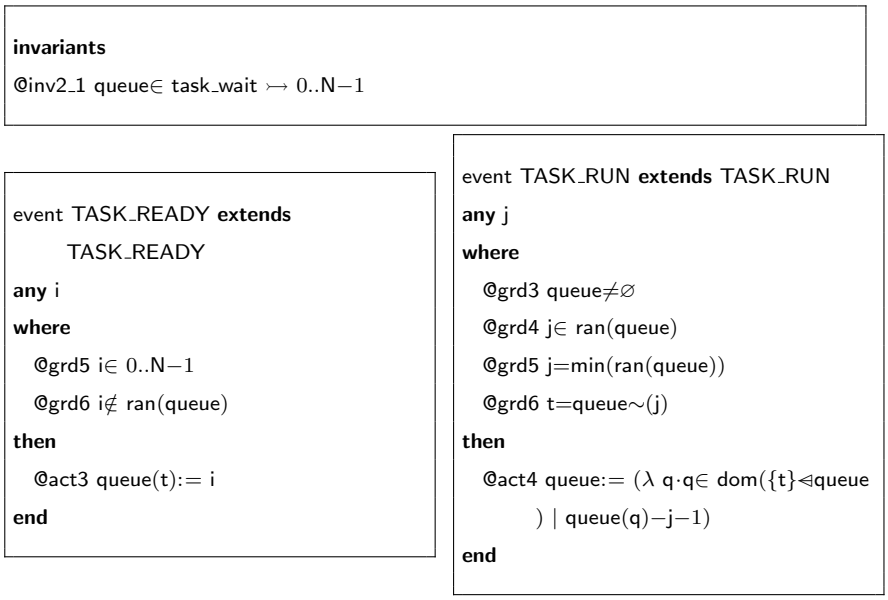

Figure 9: Second Refinement with Nondeterministic Queue Based Scheduling

Figure 10 shows the time diagram of the refinement with the scheduling framework. Assume that in the abstract machine, the trigger event of one task t occurs at timestamp $tt(t)$, and the deadline is $task\_ddl$. Additional gluing invariants are provided based on Theorem 5.1. In the refined machine, the trigger event $TASK\_READY\_EMPTY$ or $TASK\_FINISH\_NONEMPTY$

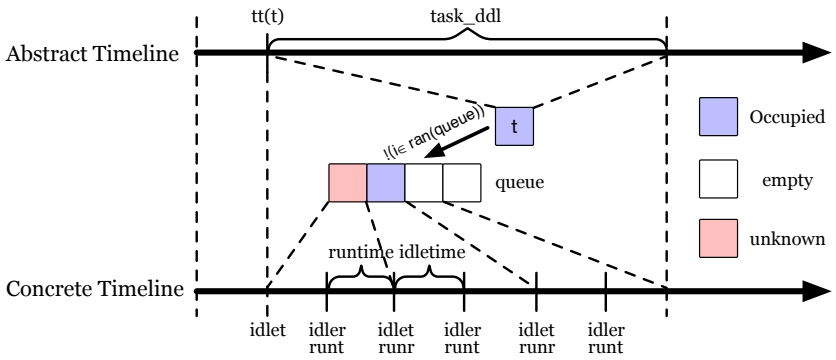

Figure 10: Time Diagram of Timing Properties' Refinement with the Scheduling Framework

starts at timestamp *idlet*, and its deadline is *idletime*. The trigger event *TASK_RUN* starts at timestamp *runt*, and its deadline is *runtime*. The task t has to wait for all the tasks ahead of it in the queue to enter and leave the critical section. The total waiting time is proportional to its index in the queue, which is $queue(t) * (idletime + runtime)$. If the critical section is empty and the time that the last task leaves the critical section is *idlet*, then $g(t) = queue(t) * (idletime + runtime)$. There might be tasks that have waited $queue(t) * (idletime + runtime)$. Then, task t should enter the critical section within $idlet + queue(t) * (idletime + runtime) + idletime$. Given that the critical section is not empty, $g(t) = queue(t) * (idletime + runtime)$. Assume that the time that the last task enters the critical section is *runt*; then, task t should enter the critical section within $runt + queue(t) * (idletime + runtime) + (idletime + runtime)$. Based on Theorem 5.1, the sum of the refined sequential deadline should be less than the abstract deadline $tt(t) + task\_ddl$, which is shown in $@inv3\_9$ and $inv3\_10$ in Figure 11. $@inv3\_9$ and $inv3\_10$ present these two conditions as required gluing invariants. Assume that there are N tasks, of which the worst case is $N - 1$ tasks in the waiting list; thus, $max(queue(t)) = N - 1$. $@axm2\_3$ and $@axm2\_4$ present the required condition. Figure 11 shows the required axioms and invariants to replace the end-to-end deadlines to scheduler-based deadlines. $@inv3\_5$ and $@inv3\_7$ present the invariant for scheduler-based

deadlines (9b), and @$inv3\_6$ and @$inv3\_8$ present the invariants for scheduler-based deadlines (9a).

5.4. Two Implementations of Nondeterministic Queue-Based Scheduling

The nondeterministic queue-based scheduling framework is a general framework that nondeterministically assigns indexes to tasks. By applying additional rules to the assignment of these indexes, the queue-based scheduling framework can be refined to some scheduling policies such as the *FIFO* and *DPB* scheduling policies. In the model, we define *FIFO* and *DPB* as two scheduling policies of the *SCHEDULING* carrier set. The refinement shows that the two scheduling policies are compatible under the HS system. Each application has either the *FIFO* or *DPB* scheduling policy based on the *scheduling* constant. As shown in Figure 12, we define the constant *scheduling* as a total function from *APPS* to *Scheduling*. Both refinements refine the nondeterministic queue base scheduling framework by restricting the position of the ready task in the queue. Details are provided in the following sections.

5.4.1. First In First Out

FIFO is one of the scheduling policies that guarantees that the resources are assigned to each task in the order that they require the resource. The FIFO scheduling policy handles all tasks without priorities. The queue-based scheduling framework assigns each task with a corresponding natural number $k \in \mathbb{N}$, and the FIFO scheduling policy limits this natural number to the current size of the queue. Moreover, when the critical section is empty, the task that is in the front of the queue leaves the queue and enters the critical section. The indexes of all the other tasks in the queue are reduced by one.

The refinement from the scheduler-based model is shown in Figure 12. Assume that the application that is running is scheduling tasks with the *FIFO* policy. Initially, the queue is empty, and *qsize* is zero. Whenever some task is added to the queue, it is assigned the number of the queue size, and the queue size increases by one when the scheduling policy is *FIFO*. When the critical section

constants idletime runtime

axioms

@axm2_1 idletime>0

@axm2_2 runtime>0

@axm2_3 ((N−1)∗(idletime+runtime))+idletime≤ task_ddl

@axm2_4 N∗(idletime+runtime)≤ task_ddl

invariants

@inv3_5 idler≥ idlet⇒ idler−idlet≤ idletime

@inv3_6 runr≥ runt⇒runr−runt≤ runtime

@inv3_7 queue≠∅ ∧task_run=∅ ⇒clk−idlet≤ idletime

@inv3_8 task_run≠ ∅⇒clk−runt≤ runtime

@inv3_9 ∀t·task_run=∅∧t∈ task_wait⇒idlet+(queue(t)∗(idletime+runtime))+idletime≤ tt(t)+task_ddl

@inv3_10 ∀t·task_run≠∅∧t∈ task_wait⇒runt+(queue(t)∗(idletime+runtime))+(idletime +runtime)≤ tt(t)+task_ddl

TASK_READY_EMPTY
extends
TASK_READY
where
@grd7 task_wait=∅∧ task_run=∅
then
@act4 idlet:= clk
end

TASK_READY_NONEMPTY
extends
TASK_READY
where
@grd7 task_wait≠∅ ∨task_run≠∅
end

TASK_RUN **extends** TASK_RUN
then
@act5 runt:= clk
@act6 idler:= clk
end

TASK_FINISH_NONEMPTY
extends
TASK_FINISH
where
@grd2 queue≠∅
then
@act2 runr:= clk
@act3 idlet:= clk
end

TASK_FINISH_IDLE
extends
TASK_FINISH
where
@grd2 queue=∅
then
@act2 runr:= clk
end

TICK **refines** TICK
where
@grd1 ∀a·a∈ app_wait ⇒clk+1−at(a)≤ app_ddl
@grd2 task_run=∅ ∧ task_wait≠∅ ⇒ clk+1−idlet≤ idletime
@grd3 task_run≠∅⇒clk +1−runt≤ runtime
then
@act1 clk:= clk+1
end

Figure 11: Replace Task-based Timing Properties with Scheduler-based Timing Properties

is empty, the task in the front of queue $queue(0)$ is removed from the queue, and the indexes of all the other tasks in the queue are reduced by one. The queue size also decreases by one. In the nondeterministic queue-based scheduling policy, the guard for $TASK\_READY$ is $i \notin ran(queue)$. $@inv4\_1$ and $@inv4\_2$ are used to to prove that $i = qsize \Rightarrow i \notin ran(queue)$. $TASK\_READY\_NONEMPTY$ uses the same refinement strategy as $TASK\_READY\_EMPTY$.

5.4.2. Deferrable Priority Based Scheduling with Aging Technique

Fixed priority scheduling policies assign tasks with fixed priorities. In the model, we use $pindex \in TASKS \rightarrow 0..N-1$ to denote the queue position of tasks with different priorities. Tasks with higher priorities have lower indexes in the queue. The scheduler will select the tasks with higher priorities to access the system resources before those with lower priorities. However, there is a disadvantage of these scheduling policies: tasks with lower priorities may be starved when the tasks with higher priorities keep coming and jumping the queue. An aging technique is used to ensure that tasks with lower priorities are eventually executed. The general way to implement an aging technique is to increase the priorities of tasks with lower priorities while they are waiting in the ready queue. However, with the increasing priorities of some tasks, aging will allow tasks with lower priorities to occupy the positions of other tasks. In contrast, deferrable priority-based scheduling allows a task to be deferred with a random position after its assigned position when some other tasks occupy the position of that task.

To avoid the starving problem of tasks with lower priorities, we add a rule to priority-based scheduling: when some other task with lower priority occupies the position of some high-priority tasks, which means that the lower priority one has waited some time in the queue, the high-priority one is deferred by some higher random index. Specifically, the indexes of the tasks are decreasing by $min(ran(queue)) + 1$ when the task at the front queue, whose index is $min(ran(queue))$, leaves the queue and enters the critical section. The *enqueue* operation will assign the task its corresponding index in the queue.

sets SCHDULING

constants FIFO DPB scheduling

axioms

@axm3_2 partition(SCHDULING,{FIFO},{DPB})

@axm3_3 scheduling∈ APPS →SCHDULING

invariants

@inv4_1 qsize∈ 0..N

@inv4_2 ∀i·i≥ qsize∧app_run⊆ scheduling∼[{FIFO}]⇒i∉ ran(queue)

TASK_READY_NONEMPTY_FIFO
refines
TASK_READY_NONEMPTY

any t i

where

@grd1 t∈ TASKS\task_wait

@grd2 t∈ TASKS\task_run

@grd4 apps(t)∈ app_run

@grd5 i=qsize

@grd6 qsize≤ N−1

@grd7 task_wait≠∅ ∨task_run≠∅

@grd8 app_run⊆ scheduling∼[{FIFO}]

then

@act1 task_wait:= task_wait∪{t}

@act2 tt(t):= clk

@act3 queue(t):= i

@act4 idlet:= clk

@act5 qsize:= qsize+1

end

TASK_RUN_FIFO **refines** TASK_RUN

any t j

where

@grd1 t∈ task_wait

@grd2 task_run=∅

@grd3 queue≠∅

@grd4 j∈ ran(queue) ∧j=0

@grd5 t=queue∼(j)

@grd6 app_run⊆ scheduling∼[{FIFO}]

then

@act1 task_wait:= task_wait\{t}

@act2 task_run:= task_run ∪{t}

@act3 tr(t):= clk

@act4 queue:= (λ q·q∈ dom({t}⩤ queue)|queue(q)−j−1)

@act5 runt:= clk

@act6 idler:= clk

@act7 qsize:= qsize−1

end

Figure 12: First In First Out Scheduling Policy

However, this operation would cause a conflict, as it will make some tasks occupy the spaces of other tasks. For example, task a's level is 3, and task b's level is 2. One task c is at the front of the queue. When c leaves the queue, the index of a is reduced to 2. When b wishes to enter the queue, its position is taken by a. Here, we choose the next available space available in the queue $i = min(k \mid k \in ran(pindex) \wedge k \notin ran(queue) \wedge k > pindex(t))$. When other tasks do not take the position, the task takes its assigned position $pindex(t)$. The dequeue operation is the same as the basic queue-based scheduling framework. Figure 13 shows the refinement from the scheduler-based model with a deferrable priority-based scheduling policy with aging technique.

constants pindex
@axm3_1 pindex$\in$ TASKS $\rightarrow$0..N$-$1

TASK_READY_NONEMPTY_DPB **refines** TASK_READY_NONEMPTY
any t i
where
@grd1 t$\in$ TASKS\task_wait
@grd2 t$\in$ TASKS\task_run
@grd4 apps(t)$\in$ app_run
@grd5 {k|k$\in$ ran(pindex)$\wedge$k$\notin$ ran(queue)$\wedge$k$\geq$ pindex(t)}$\neq\varnothing$
@grd6 i=min({k|k$\in$ ran(pindex)$\wedge$k$\notin$ ran(queue)$\wedge$k$\geq$ pindex(t)})
@grd7 task_wait$\neq\varnothing$ $\vee$task_run$\neq\varnothing$
@grd8 app_run$\subseteq$ scheduling$\sim$[{DPB}]
then
@act1 task_wait:= task_wait$\cup${t}
@act2 tt(t):= clk
@act3 queue(t):= i
end

TASK_RUN_DPB **extends** TASK_RUN
where
@grd7 app_run$\subseteq$ scheduling$\sim$[{DPB}]
end

Figure 13: Deferrable Priority Based Scheduling Policy with Aging Technique

Table 1: Proof Statistics

Machine	Generated PO	Automatically Proved	Automatically Proved %
m0	14	12	85.7
m1	34	34	100
m2	6	5	83.3
m3	49	43	87.8
m4	27	27	100

5.5. Proof Statistics

Table 1 shows the proof statistics of the model. In $m3$, six proof obligations, all of which are related to @$inv3\_9$ and @$inv3\_10$, cannot be discharged automatically. As mentioned in Theorem 5.1, the modeler needs to provide the function $g(x)$ that denotes the maximum waiting time of each task x based on different scheduling policies. Thus, the modeler needs to verify that these two invariants are consistent with the whole model. Then, @$inv3\_9$ and @$inv3\_10$ can be used to discharge the *GRD* proof obligation of the *Tick* event. In $m4$, we provide @$inv4\_2$ to capture the property that given the *FIFO* scheduling policy, any i larger or equal to *qsize* is not in the range of *queue*. @$inv4\_2$ helps with the proof obligations to refine the nondeterministic queue-based scheduling policy to the *FIFO* policy. Therefore, all proof obligations in $m4$ are discharged. The model and the proofs supporting this study are openly available from the University of Southampton repository at [17].

6. Related Work

Timing issues are critical in cyber-physical systems. Timing analysis should be carried out together with the development of the system to improve the real-time performance as well as guarantee the safety of the whole system. Timed automata [18] that are supported by the UPPAAL [19] model checker has been

used in industrial modeling of real-time systems. It is challenging to model a complex system with the timed automata formalism and UPPAAL as it does not support the refinement of the model. Some approaches, such as counterexample-guided abstraction refinement have been brought up to add abstraction and refinement when modeling a complex system [20]. This approach uses a model checker to get the counterexamples from the abstract model and uses these counterexamples as guides to refine the system. However, it is difficult to find the missing part from the model just from counterexamples. Abadi and Lamport started to specify and reason about real-time systems by representing time as an ordinary variable in Temporal Logic of Actions (TLA) [21]. Their work put time bounds on individual actions by using *timers* to restrict the increase of the global clock. Based on their work, Zhang et al. specified the time interval between two actions with TLA [22]. The time specifications are expressed by the TLA+ language [23], which is supported by the TLC model checker [24]. However, the TLC model checker does not support parameterized specification. Also, refinement of time specifications is not adapted to the verification support of TLA+. Our work not only presents parameterized timing properties but also provide refinement patterns to replace parameterized timing properties to unparameterized timing properties.

Event-B supports modeling refinement but lacks explicit support for expressing and verifying timing constraints [13]. Rehm proposed to add a timestamp set that links events to different timestamps in Event-B models. When the global clock has reached some timestamp, then the linked event would be triggered [25]. However, their approach cannot specify timing properties such as delay and deadline on actions or between actions. Influenced by Abadi and Lamport work that put time bounds on individual actions, Butler and Falampin proposed an approach to model and refine timing properties in classical B [8], which adds a clock variable representing the current time and an operation which advances the clock [26]. Additional constraints are added to the clock so that the global clock can not advance to a point where deadlines would be violated. Based on this approach, work has been done to extend Event-B models with timing

properties and refinement patterns [6, 13]. Sarshogh and Butler developed a trigger-response pattern to extend Event-B models with discrete timing properties such as deadline, delay, and expiry [6]. Their approach sets timestamps for trigger and response events and uses a *Tick* event to prevent the global clock from proceeding to a point where time constraints between trigger and response events would be violated. Sulskus et al. presented the notion and Event-B semantics for the interval timing properties by using an interrupt event between trigger and response events [13]. However, Sarshogh and Sulskus's work did not specify the relations between trigger and response events formally. Our work provides the semantics and syntax with proofs.

There are several patterns developed by Sarshogh to refine deadlines, delay, and expiry. For example, to refine an abstract deadline D to sequential sub-deadlines $D_1..D_n$, there should be invariants to ensure the order of sequential sub-deadlines and the sum of the duration of sub-deadlines should be less than the abstract deadline duration [6]. Sarshogh's approach only handles the system with trigger and response pattern without specifying some possible interrupt events from the environment. Sulskus et al. extended their work by constructing a set of refinement transformations with Event-B code templates to verify and validate interval timing properties [27]. Their work provides soundproof to refine abstract time intervals to alternative or sequent sub-time intervals. However, they lack refinement patterns to replace parameterized time intervals to unparameterized time intervals.

The trigger-response pattern only models discrete-time constraints, while the real-world events do not always happen at integer-value times. Continuous-time can be modeled approximated by choosing the granularity of the global clock, which models the timed system with an approximate sense. Banach et al. present the Hybrid Event-B extension, which accommodates continuous behaviors in between discrete transitions [28]. Based on this extension, Butler et al. outline an approach to modeling and reasoning about hybrid systems which uses continuous functions over real intervals to model the evolution of continuous values over time [29].

7. Conclusion and Future Work

Based on a trigger-response approach to modeling timing properties in Event-B, we present the syntax of parameterized real-time trigger-response properties with Event-B formalization and proofs. Rules and proofs are provided to replace parameterized timing properties with unparameterized timing properties. To distinguish timing properties from the perspective of different system design phases, we define parameterized timing properties that place discrete-time constraints on individual tasks as end-to-end timing properties, which describe high-level timing properties from the system requirement specification phase. These end-to-end timing properties cannot precisely describe the concurrent behavior of tasks. In real-time systems, schedulers are used to schedule concurrent tasks. To model the behavior of these concurrent tasks, we define scheduler-based timing properties with unparameterized timing properties as concrete timing properties for the system design phase, which places discrete-time constraints on the scheduler that schedules the concurrent tasks. To replace end-to-end timing properties with scheduler-based timing properties, we introduce a nondeterministic queue-based scheduling policy with some additional gluing invariants. We formalize a two-level hierarchical scheduling system that refines the nondeterministic queue-based scheduling policies to two compatible local scheduling policies to illustrate the pattern that replaces end-to-end timing properties with scheduler-based timing properties.

This paper addresses the safety properties but not the liveness properties and possible *Zeno* behavior, whereby an infinite number of events occur within a limited period of time [30]. In our setting, if there is an infinite number of intermediate events occurring between the trigger and response pair, then the response events are infeasible, and the *Tick* event cannot increment the global clock. Fairness assumptions are required in the model to guarantee that when the trigger event occurs, the response event will eventually occur. Additional refinement rules are required to guarantee that the refined model also avoids *Zeno* behavior. In this paper, we only treat the safety properties of a system,

which guarantee that the response event does not occur earlier than the delay time or later than the deadline time. So we assume that the trigger-response pair (T, R) satisfies the liveness property which guarantees that a trigger event is followed eventually by a response event. [14] dealt with liveness properties by imposing weak fairness as well as conditional convergence to prove the eventual occurrence of the response event and the global tick event. [31] extended the work by introducing refinement patterns of timing properties with weak fairness assumptions. However, the work in [14, 31] did not deal with the parameterized trigger-response timing properties of this paper. Future work could also explore the conditions for the liveness properties of the parameterized trigger-response properties.

In the cases that the system does not require an explicit mention of time, the notion of bounded fairness and finitary fairness allows one to express the eventual occurrence of a set of events. Some work has been done to model fairness in Event-B [32, 33]. Bounded fairness modeling, as well as finitary fairness modeling, can be researched further with some addition prove rules and refinement frameworks.

To explicitly represent the timing properties in a cyber-physical system, there are three typical time constraints to consider: period, deadline, and worst-case execution time. More work can be done to apply scheduling policies such as rate-monotonic (RM) scheduling and the priority inheritance protocol based on the queue-based scheduling framework to analyze the real-time performance of CPS together with the mentioned time constraints in Event-B. Hoang et al. proposed to reuse simple models as patterns to construct larger models [34]. Therefore, our work of refining real-time properties with the scheduling framework can also be used as a pattern to refine complicated real-time systems with concurrent tasks.

Acknowledgment

Our sincere thanks go to anonymous reviewers who provided helpful suggestions to improve the quality of the paper. This work is supported in part by the scholarship from China Scholarship Council (CSC) under the Grant CSC NO.201708060147.

[1] R. Alur, Principles of Cyber-Physical Systems, The MIT Press, 2015.

[2] H. Kopetz, Real-time systems: design principles for distributed embedded applications, Springer Science & Business Media, 2011.

[3] E. M. Clarke, J. M. Wing, Formal methods: State of the art and future directions, ACM Computing Surveys (CSUR) 28 (4) (1996) 626–643.

[4] M. Butler, Mastering System Analysis and Design through Abstraction and Refinement, IOS Press, 2013.
URL http://eprints.soton.ac.uk/349769/

[5] J.-R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.

[6] M. R. Sarshogh, M. Butler, Specification and refinement of discrete timing properties in Event-BEvent Dates: September 2011.
URL https://eprints.soton.ac.uk/272480/

[7] C. Zhu, M. Butler, C. Cirstea, Refinement of timing constraints for concurrent tasks with scheduling, in: M. Butler, A. Raschke, T. Hoang, K. Reichl (Eds.), Abstract State Machines, Alloy, B, TLA, VDM, and Z: ABZ 2018, Vol. 10817, Springer, 2018, pp. 219–233.
URL https://eprints.soton.ac.uk/419024/

[8] J.-R. Abrial, The B-book: assigning programs to meanings, Cambridge University Press, 2005.

[9] R.-J. Back, F. Kurki-Suonio, Distributed cooperation with action systems, ACM Transactions on Programming Languages and Systems (TOPLAS) 10 (4) (1988) 513–554.

[10] E. W. Dijkstra, Guarded commands, nondeterminacy, and formal derivation of programs, in: Programming Methodology, Springer, 1978, pp. 166–175.

[11] R.-J. Back, Refinement calculus, part ii: Parallel and reactive programs, in: Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems), Springer, 1989, pp. 67–93.

[12] M. Jastram, P. Butler, Rodin User's Handbook: Covers Rodin V.2.8, 2.8covers Rodin, Createspace Independent Pub, 2014.
URL https://books.google.co.uk/books?id=ws2WoAEACAAJ

[13] G. Sulskus, M. Poppleton, A. Rezazadeh, An interval-based approach to modelling time in Event-B, Fundamentals of Software Engineering 9392 (2015) 292–307.
URL http://eprints.soton.ac.uk/377201/

[14] C. Zhu, M. Butler, C. Cirstea, Semantics of real-time trigger-response properties in Event-B, in: 2018 International Symposium on Theoretical Aspects of Software Engineering (TASE), 2018, pp. 150–155. doi:10.1109/TASE.2018.00028.

[15] J. D. Regehr, J. A. Stankovic, Using hierarchical scheduling to support soft real-time applications in general-purpose operating systems, University of Virginia, 2001.

[16] M. Butler, Incremental design of distributed systems with Event-B, in: M. Broy, W. Sitou, T. Hoare (Eds.), Engineering Methods and Tools for Software Safety and Security - Marktoberdorf Summer School 2008, IOS Press, 2009, pp. 131–160, chapter: 4.
URL https://eprints.soton.ac.uk/266910/

[17] C. Zhu, M. Butler, C. Cirstea, Formalizing hierarchical scheduling for refinement of real-time systems (September 2019).
URL https://eprints.soton.ac.uk/434252/

[18] R. Alur, D. L. Dill, A theory of timed automata, Theoretical computer science 126 (2) (1994) 183–235.

[19] K. G. Larsen, P. Pettersson, W. Yi, Uppaal in a nutshell, International journal on software tools for technology transfer 1 (1-2) (1997) 134–152.

[20] H. Dierks, S. Kupferschmid, K. G. Larsen, Automatic abstraction refinement for timed automata, in: International Conference on Formal Modeling and Analysis of Timed Systems, Springer, 2007, pp. 114–129.

[21] M. Abadi, L. Lamport, An old-fashioned recipe for real time, ACM Trans. Program. Lang. Syst. 16 (5) (1994) 1543–1571. doi:10.1145/186025.186058.
URL http://doi.acm.org/10.1145/186025.186058

[22] H. Zhang, M. Gu, X. Song, Specifying time-sensitive systems with tla+, in: 2010 IEEE 34th Annual Computer Software and Applications Conference, IEEE, 2010, pp. 425–430.

[23] L. Lamport, Specifying systems: the TLA+ language and tools for hardware and software engineers, Addison-Wesley Longman Publishing Co., Inc., 2002.

[24] Y. Yu, P. Manolios, L. Lamport, Model checking tla+ specifications, in: Advanced Research Working Conference on Correct Hardware Design and Verification Methods, Springer, 1999, pp. 54–66.

[25] J. Rehm, Proved development of the real-time properties of the ieee 1394 root contention protocol with the Event-B method, International journal on software tools for technology transfer 12 (1) (2010) 39–51.

[26] M. Butler, J. Falampin, An approach to modelling and refining timing properties in B (January 2002).
URL https://eprints.soton.ac.uk/256235/

[27] G. Sulskus, M. Poppleton, A. Rezazadeh, Modelling complex timing requirements with refinement, in: 2016 IEEE 17th International Conference on Information Reuse and Integration (IRI), IEEE, 2016, pp. 118–125.

[28] R. Banach, M. Butler, S. Qin, N. Verma, H. Zhu, Core hybrid Event-B i: Single hybrid Event-B machines, Science of Computer Programming 105 (2015) 92 – 123.

[29] M. Butler, J.-R. Abrial, R. Banach, Modelling and refining hybrid systems in Event-B and rodin, in: L. Petre, E. Sekerinski (Eds.), From Action System to Distributed Systems: The Refinement Approach, Taylor & Francis, 2016.
URL https://eprints.soton.ac.uk/376053/

[30] N. A. Lynch, F. W. Vaandrager, Forward and backward simulations, ii: Timing-based systems, Inf. Comput. 128 (1996) 1–25.

[31] C. Zhu, M. Butler, C. Cirstea, Towards refinement semantics of real-time trigger-response properties in event-b, in: 13th International Symposium on Theoretical Aspects of Software Engineering (01/08/19), 2019.
URL https://eprints.soton.ac.uk/430321/

[32] E. Sekerinski, T. Zhang, Finitary fairness in Event-B, in: Dagstuhl Seminar on Refinement Based Methods for the Construction of Dependable Systems (Dagstuhl, Germany, 2009.

[33] E. Sekerinski, T. Zhang, Finitary fairness in action systems, in: Z. Liu, J. Woodcock, H. Zhu (Eds.), Theoretical Aspects of Computing – ICTAC 2013, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 319–336.

[34] T. S. Hoang, A. Fürst, J.-R. Abrial, Event-B patterns and their tool support, Software & Systems Modeling 12 (2) (2013) 229–244.