Trace Semantics and Refinement Patterns for Real-time Properties in Event-B Models

Chenyang Zhu, Michael Butler, Corina Cirstea

Abstract

Event-B is a formal method that utilizes a stepwise development approach for system-level modeling and analysis. We are interested in reasoning about real-time deadlines and delays between trigger and response events. There is existing work on treating these properties in Event-B but it lacks a semantic treatment in terms of trace behaviors. Because timing properties require fairness assumptions, we use infinite traces and develop conditions under which all infinite traces of a machine satisfy trigger-response and timing properties. We present refinement semantics of models whose behavior traces are infinite. In addition, we generalize our previous work by allowing a relation between concrete states and abstract states to simulate infinite state traces. Forward simulation, which is a proof technique for refinement, has been used to verify the consistency between different refinement levels regarding finite traces. Based on forward simulation, fairness assumptions, relative deadlock freedom, and conditional convergence are adopted as additional conditions that guarantee infinite trace refinement of timed models. The bounded retransmission protocol is used to illustrate the required proof obligations for timed traces.

Keywords: Event-B, Trace Semantics, Refinement Semantics, Discrete Time Modeling, Hiding Operator

1. Introduction

In recent years, reducing the cost of software development and improving software quality have received considerable critical attention. Formal methods are one of the solutions to improving software quality that provide formal modeling, reasoning, and verification in software and hardware development cycles such as requirement specification, system design, and code validation. Event-B is usually used for system-level modeling and analysis with a stepwise development approach to manage the complexity in system design [1]. However, it lacks explicit support for the timed systems in terms of infinite behavioral traces.

Formal modeling is used to manage the precision of specifications, but real systems are difficult to specify and verify without abstractions [2]. Abstraction and refinement can help to master the complexity of requirements and design [3]. Abstraction makes it easier to model and reason about a system with a high-level goal. Each refinement step requires that the behavior of a refined model should be consistent with the behavior of the model being refined. However, the existing refinement framework in Event-B models lacks a proper treatment of infinite behavioral traces, namely the divergence of new events in the refined machine and infeasible occurrence of events without fairness assumptions. Forward simulation is used as a proof technique to verify the consistency between different refinement levels regarding finite traces. Based on the assumption that the relation between concrete states and abstract states is functional rather than the more general relational form, our previous work in [4] explored sufficient conditions under which the refinement is valid in terms of infinite traces. Fairness assumptions, relative deadlock freedom, and conditional convergence are adopted to refine the discrete timed models.

However, the functional relation assumption does not suit many real-world refinement cases as the concrete states might correspond to one or more abstract states. Thus the current work generalizes our previous work by allowing a relation between concrete states and abstract states to simulate infinite state traces. In this paper, we present infinite state trace refinement with gluing relations between abstract and concrete state traces. Then the definition of generic refinement that allows relabelling and stuttering events is presented with event trace inclusion. We summarize infinite trace properties as well as sufficient conditions under which timing properties are preserved by refinement.

A Bounded Retransmission Protocol (BRP) case study is used to present the refinement conditions for discrete timed systems.

The paper is structured as follows: Section 2 provides definitions for trace semantics and trace properties. Section 3 introduces a bounded retransmission protocol as a case study to illustrate the timed trigger-response properties and presents the corresponding Event-B model. Section 4 applies the infinite event trace refinement to verify the refinement of models without relabelling and stuttering events. Section 5 then shows the generic version of refinement semantics in terms of infinite behavioral traces. Section 6 describes some related work regarding refinement with forward simulation as well as refinement of timed systems. Section 7 summarizes the work and outlines future work.

2. Trace Semantics and Properties

2.1. Event-B

Event-B [1] 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 B-Method developed by Abrial [5], is greatly inspired by the notion of action systems [6] and guarded commands [7]. 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* [1]. 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* [1].

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

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

The normal Event-B notion of convergence requires convergent events to become disabled eventually without any fairness assumptions. Our previous work extended the notion of convergence to conditional convergence to define a set of events are converging under certain condition $Q(v)$ in Definition 2.1 [4]. The notion of conditional convergence is a generalized version of convergence as it only requires the events to be convergent under some specific condition. If the condition is not satisfied, then the events do not need to converge.

Definition 2.1 (Conditional Convergence [4])**.** *A group of events A is defined to be conditional convergent under the condition Q when*

- *A variant $V(v)$ is defined as a natural number*
- *When the group of events is enabled, the variant $V(v)$ is a natural number.*

$$I(v) \wedge G_A(v) \Rightarrow V(v) \in \mathbb{N}$$

- *An execution of each event $e \in A$ decreases $V(v)$ provided $Q(v)$ holds.*

$$I(v) \wedge G_e(v) \wedge S_e(v, v') \wedge Q(v) \Rightarrow V(v') < V(v)$$

2.2. Trace Semantics of Event-B Machine

Event-B is a modeling approach for formalizing discrete transition systems. In this paper, we abstract away from the concrete syntax of Event-B and treat a machine as a form of labeled transition system (Definition 2.2). In the definition, the machine consists of state set S and initial states $init \subseteq S$. The set of events is defined as E, which relates a pair of states with a transition relation K.

Definition 2.2 (Machine [4])**.** *A machine M is a tuple $< S, init, E, K >$ consisting of*

- *S: a set of states, each state s is a mapping of variables of M to their values;*
- *init: a set of initial states $init \subseteq S$, which correspond to initial configurations;*

- *E: a set of event labels of the machine M;*
- *K: a transition relation $K \in E \rightarrow (S \leftrightarrow S)$ that relates pairs of states;*

Given the definition of a *machine*, we define $traces(M)$ of machine M to describe the infinite behavior of the system in terms of a set of infinite sequences of alternating states and events of the form $< u_s(0), u_e(0), u_s(1), u_e(1), ... >$ in Definition 2.3 [4].

Definition 2.3 (Infinite Trace of a Machine [4])**.** *The set of infinite traces $traces(M)$ of a machine M is defined as:*

$$traces(M) \triangleq \{(u_s, u_e) \mid u_s \in \mathbb{N} \rightarrow S \wedge u_e \in \mathbb{N} \rightarrow E \wedge u_s(0) \in init \\ \wedge \forall i \cdot i \geq 0 \Rightarrow u_s(i) \mapsto u_s(i+1) \in K(u_e(i))\}$$

The event traces (Definition 2.4) and state traces (Definition 2.5) of a machine are defined below; their properties are studied in the next section.

Definition 2.4 (The Set of Event Traces of M)**.** *A set of event traces of M is defined as:*

$$e\_traces(M) = \{u_e \mid \exists u_s \cdot (u_s, u_e) \in traces(M)\}$$

Definition 2.5 (The Set of State Traces of M)**.** *A set of state traces of M for the event trace $u_e \in e\_traces(M)$ is defined as:*

$$s\_traces(M, u_e) = \{u_s \mid (u_s, u_e) \in traces(M)\}$$

Lemma 2.6 shows that the state trace set of an event trace in machine M is not empty.

Lemma 2.6.

$$u_e \in e\_traces(M) \Leftrightarrow s\_traces(M, u_e) \neq \varnothing$$

2.3. Trace Properties

In this paper, we adopt weak fairness assumptions on infinite traces to exclude traces with *Zeno* behaviors, whereby an infinite number of events occur within a finite period. In Definition 2.7, we extend the machine definition in Definition 2.2 with a set of events $F \subseteq E$. The traces of a machine that is weakly fair on the event set F satisfy the property that if F is continuously enabled from some point, then $f \in F$ will eventually occur on the trace.

Definition 2.7 (Machine Traces with Weak Fairness)**.** *A machine M with weak fairness is a tuple $< S, init, E, K, F >$ consisting of a machine $< S, init, E, K >$ and a set of weakly fair events $F \subseteq E$. The trace $(u_s, u_e) \in traces(M)$ is defined as weakly fair on F as $WFair((u_s, u_e), F)$, formally:*

$$WFair((u_s, u_e), F) \triangleq$$

$$\forall i \cdot i \geq 0 \wedge u_s(i) \in dom(K(F)) \Rightarrow \exists j \cdot j \geq i \wedge (u_e(j) \in F \vee u_s(j) \notin dom(K(F)))$$

Based on Definition 2.1, we provide the conditional convergence trace property in Definition 2.8 such that event set A must not be executed forever under the condition Q.

Definition 2.8 (Conditional Convergence Property)**.** *Trace $(u_s, u_e) \in traces(M)$ is defined to be convergent as $(u_s, u_e) \models cov(A, Q)$, formally:*

$$(u_s, u_e) \models cov(A, Q) \triangleq \forall i \cdot (\exists j \cdot j \geq i \Rightarrow (u_s(j) \notin dom(K(A)) \vee \neg Q))$$

2.4. Trigger-Response Pattern

Abid et al. proposed a set of specification patterns that can be used to express real-time requirements in the design of reactive systems [8], namely existence patterns, absence patterns, and response patterns. Existence patterns are used to express that certain events must occur in every trace of the system; absence patterns require that certain events should never occur in every trace of the system. A trigger-response pair in Event-B models can represent these patterns. In the trigger-response pair, the trigger event is followed by its possible response events eventually. Given a machine with events E, a trigger-response pair has the form (T, R) where $T \subseteq E \wedge R \subseteq E \wedge T \cap R = \varnothing$. We define when an event trace of a machine satisfies the trigger-response property $TR(T, R)$ as

Definition 2.9. An event trace u_e satisfies $TR(T, R)$ provided that any occurrence of a trigger event $t \in T$ is followed eventually by a response event $r \in R$ and the trigger event does not recur within the trigger-response pair to avoid the recurring delay of response event.

Definition 2.9 (Trigger-Response Property). *Trace $u_e \in \mathbb{N} \to E$ is defined to satisfy the trigger-response property $TR(T, R)$ as:*

$$u_e \models TR(T, R) \triangleq$$

$$\forall i \cdot i \geq 0 \wedge u_e(i) \in T \Rightarrow (\exists j \cdot j > i \wedge u_e(j) \in R \wedge \forall k \cdot i < k < j \Rightarrow u_e(k) \notin T)$$

In a trigger-response pair, the response event is not necessarily enabled after the execution of a trigger event. Instead, there may be a group of intermediate events H that is enabled by a trigger event $t \in T$ and whose execution leads to a response event $r \in R$ being enabled. Work in [9] explored sufficient conditions under which a behavioral trace satisfies the trigger-response property. For each trigger and response pair (T, R), a set of intermediate events $H \subset E$ s.t. $T \cap H = \varnothing$ and $R \cap H = \varnothing$ is assumed. This set is chosen by the modeler and consists of events that are initiated by a trigger event and whose execution is intended to lead to a response being enabled. The intermediate events should converge towards a response event being enabled and should not be disabled unless the response event is enabled. Weak fairness assumptions on the intermediate events guarantee that the intermediate events get executed sufficiently often to lead to the response event being enabled. We require that if a response event is enabled, it cannot be disabled by any event other than a response event. Weak fairness assumptions on the response events guarantee that an enabled response event eventually gets executed. To avoid the scenario that intermediate events and response events are enabled alternatively but never get executed under the weak fairness assumption, we require that only response events can disable themselves.

To model real-time systems, we introduce a special event *Tick* in event traces to represent the progress of time. In a real-time system, one essential property requires that time should always progress. The event traces of real-time systems should always be *infinite traces* with *infinitely many Tick* events. Definition 2.10

provides the timed property of a timed trace, which guarantees that there is an infinite number of $Tick$ events occurring in the event trace. Moreover, only a finite number of $non - Tick$ events could occur between any two $Tick$ events.

Definition 2.10 (Timed Trace)**.** *Event trace u_e is defined to be a timed trace as*

$$timed(u_e) \triangleq \forall i \cdot i \in \mathbb{N} \Rightarrow (\exists j \cdot j \geq i \land u(j) = Tick)$$

Given a trigger-response pair (T, R), we use the number of $Tick$ events between the trigger event T and response event R to denote the corresponding timing property for the trigger-response pair. Definition 2.11 shows the bounded real-time property for the trigger-response pair (T, R). The number of $Tick$ events between trigger and response events is restricted by a lower bound w and an upper bound d. The response event R must occur within time d of trigger event T occurring and can only occur if the delay period w has passed. Given the event trace u_e, the *projection* $u_e \upharpoonright A$ is the event trace restricted to only those events in A. The length operator $\#(u_e)$ defines the length of the event trace. The sub-trace function $u_e[i, j]$ defines the sub-trace between index i and j as $< u_e(i), u_e(i + 1), ..., u_e(j) >$. The event trace that satisfies the bounded real-time property should satisfy the trigger-response property as well as the timed property.

Definition 2.11 (Bounded Timing Property)**.** *An infinite event trace u_e is said to satisfy the bounded timing property $u_e \models TP(T, R, w, d)$ provided the following conditions hold:*

$$\begin{aligned} u_e \models TP(T, R, w, d) \triangleq & timed(u_e) \land \forall i \cdot i \geq 0 \land u_e(i) \in T \Rightarrow \\ & \exists j \cdot j > i \land u_e(j) \in R \land (w \leq \#(u_e[i, j] \upharpoonright Tick) \leq d) \end{aligned}$$

We define the traces of Event-B models with bounded timing properties as $traces(M \land TP)$ in Definition 2.12. Here we use $M \land TP$ to present a machine M that is constrained to satisfy timing properties TP.

Definition 2.12 (Machine with Bounded Timing Property)**.**

$$traces(M \land TP) \triangleq \{(u_s, u_e) \mid (u_s, u_e) \in traces(M) \land u_e \models TP\}$$

3. Example

In this paper, we take the BRP as an example to illustrate the refinement patterns for real-time properties in Event-B models. The BRP is a file transfer protocol that deals with the fault tolerance of the system under unreliable network communications [10]. The schematic view of the transmission protocol is shown in Figure 1. The transmitter sends the file to the receiver packet by packet through the data channel. As soon as the receiver receives the packet, it sends back an acknowledgment to the transmitter through an acknowledgment channel. When the transmitter receives the acknowledgment, it confirms that the packet is sent out successfully and sends out the next packet. As the data channel and the acknowledgment channel are not reliable, the packet might be lost, or the transmitter might not receive the acknowledgment. The transmitter will resend the packet to the receiver if it has not received confirmation of the same packet within some deadline. In the case of successive losses of messages, the process of packet re-transmission can be repeated several times with a retry counter. When the counter reaches a specific limit, the transmitter and the receiver decides to abort.

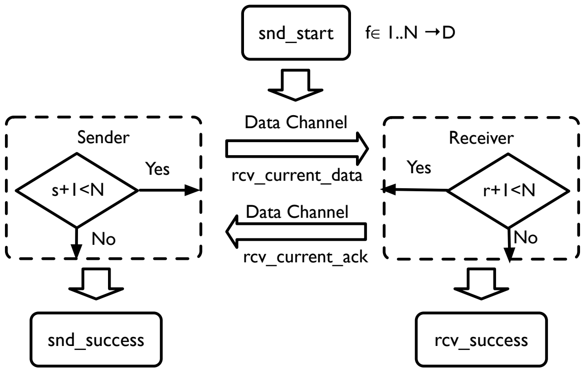


Figure 1: Scheme View of the Bounded Retransmission Protocol

The machine given in Figure 2 abstractly specifies the BRP protocol with real-time properties. The variable s presents the file pointer in the transmitter. The transmitter, receiver and channel have three states, namely *working*, *success*, and *failure*. *Success* denotes the state that a file has been transmitted successfully, and the next file is ready to transmit. *Working* represents the state that the transmitter and receiver are transmitting the file packet by packet. *Failure* means that there is something wrong with the data channel or the acknowledgment channel, and the system has to abort. To start with the *snd_start* event, the transmitter state *s_st* and receiver state *r_st* are set to *working*. Also, the channel state *c_st* is set to *working* or *failure* nondeterministically. In the case the channel is working (*rcv_current_ack* event), the file pointer is increased to show that the transmitter transfers the last packet and receives the corresponding acknowledgment successfully. The channel is set to either *working* or *failure* to transfer the next packet. In the case that the packet or acknowledgment is lost during transmission, the receiver aborts the transmission with event *rcv_abt*. We use (2) to present the trigger-response property that once a packet is transmitted from the transmitter, either it is transmitted successfully, or the receiver abort. In the model, *rcv_current_ack* enables *rcv_current_ack* or *rcv_abt* events. (2) is a valid trigger-response pair when we assume weak fairness on *rcv_current_ack* and *rcv_abt* events.

$$TR(rcv\_current\_ack, \{rcv\_current\_ack, rcv\_abt\}) \tag{2}$$

The trigger-response property is then extended with the *pkt_ddl* deadline. A new *clock* variable is added to represent the current time. And we set the timestamps of *rcv_current_ack*, *rcv_abt* as *pkt_tp*, *pkt_tc* and *abt_t* respectively. *pkt_tp* denotes the timestamp of receiving the acknowledgment from previous packet, *pkt_tc* presents the timestamp of current acknowledgment. $@inv1\_2$ and $@inv1\_4$ show the timing property (3). $@inv1\_1$ and $@inv1\_3$ serve as auxiliary invariants to support $@inv1\_2$ and $@inv1\_4$. Some infinite event trace of machine $M0$ is shown in (4). In later sections we will gradually refine the machine with intermediate events with real-time properties. Also, refinement strategies and additional proof obligations will be presented.

$$TP(rcv\_current\_ack, \{rcv\_current\_ack, rcv\_abt\}, 0, pkt\_ddl) \tag{3}$$

$$\begin{cases} (snd\_start, rcv\_abt, tick, tick, ...) \\ (snd\_start, rcv\_current\_ack, rcv\_abt, tick, tick, ...) \\ (snd\_start, rcv\_current\_ack, tick, rcv\_abt, tick, tick, ...) \\ (snd\_start, rcv\_current\_ack, tick, rcv\_current\_ack, tick, tick, ...) \end{cases} \tag{4}$$

```
@inv1_1 s_st=working ∧c_st=working ⇒clk−pkt_tc≤  pkt_ddl
@inv1_2 pkt_tp≤  pkt_tc ⇒pkt_tc−pkt_tp≤  pkt_ddl
@inv1_3 r_st=working ∧c_st=failure ⇒clk−pkt_tc≤  pkt_ddl
@inv1_4 pkt_tc≤ abt_t ⇒abt_t− pkt_tc≤  pkt_ddl
```

```
event snd_start
where
  @grd1 s_st=success
  @grd2 s=0
then
  @act1 s_st:= working
  @act2 r_st:= working
  @act3 c_st:∈ {working, failure}
  @act4 pkt_tc:= clk
  @act5 pkt_tp:= clk
end

event rcv_abt
where
  @grd1 r_st=working
  @grd2 c_st=failure
then
  @act1 r_st:= failure
  @act2 abt_t:= clk
end
```

```
event rcv_current_ack
where
  @grd1 s_st=working
  @grd2 c_st=working
  @grd3 s+1<N
then
  @act1 s:= s+1
  @act2 c_st:∈ {working, failure}
  @act3 pkt_tc:= clk
  @act4 pkt_tp:= pkt_tc
end

event tick
where
  @grd1_1 s_st=working ∧c_st=working =>clk+1−
        pkt_tc≤  pkt_ddl
  @grd1_2 r_st=working ∧c_st=failure =>clk+1−
        pkt_tc≤  pkt_ddl
then
  @act1_1 clk:= clk+1
end
```

Figure 2: Machine $M0$ that Specifies the Abstract BRP Protocol

4. Trace Refinement Semantics

Abstraction and refinement are usually essential to manage the complexity of modeling and reasoning about a system. Refinement of a system usually involves changing the variables of the system [11]. Data refinement is used to add more details to the data structure in the model, either by replacing existing variables or adding new variables to the model. In Event-B machines, gluing invariants are used to link the variables in the refined model to the variables in the abstract model.

In this paper, we use $M =< S, S_0, E, K >$ to denote the abstract machine, which is data-refined to the concrete machine $M' =< S', S_0', F, K' >$. Syntactically the abstract and concrete state spaces are represented by the possible values of the variables v and w respectively. S_0' is defined by a predicate $L(w)$. The transition relation K' of an event $e \in F$ is defined by its guard $H_e(w)$ and action predicate $R_e(w)$. In general, gluing invariants define a relational mapping between concrete and abstract states. In this paper, instead of assuming a mapping function that maps states $s \in S'$ to states $s' \in S$ in our previous work [4], we assume $J \in S \leftrightarrow S'$ as a gluing relation that relates the states of M and M'. Syntactically, J is represented by a predicate $J(v, w)$. Event mapping function $g \in F \to E \cup \{skip\}$ is a total function from refined event labels to abstract event labels and *skip*. The *skip* events are mapped from concrete new events in M'. The infinite trace model allows us to give a proper treatment of timed traces. In the following section we use event trace inclusion to define the refinement semantics between different machines.

We first present the most straight-forward refinement semantics, which assumes that the concrete model does not introduce new events and there is no relabelling of the events. Then we generalize the result to allow for relabelling and new events. We treat traces for machines as compound traces of states and events. We first prove state trace refinement with forward simulation and Zorn's Lemma, which then can be used to prove the event trace refinement.

4.1. Forward Simulation and Infinite State Trace Refinement

Forward simulation is a sufficient condition for refinement of systems with finite traces [1]. In this paper, we show that forward simulation using a relational abstraction relation is sufficient for infinite trace refinement as well. Formally, forward simulation is defined in Definition 4.1. Figure 3 illustrates that the concrete machine M' simulates abstract M provided for each transition in M' that may lead from an set of states S'_i to a set of states S'_{i+1}, there exists a corresponding transition on the abstract machine M from an abstract state set S_i to a set of states S_{i+1}. The states S and S' are gluing related by J.

$$\begin{tikzcd}
S_i \arrow[r, "K(g(e))"] \arrow[d, leftrightarrow, "J"'] & S_{i+1} \arrow[d, leftrightarrow, "J"] \\
S_i' \arrow[r, "K'(e)"] & S_{i+1}'
\end{tikzcd}$$

$\forall e \cdot e \in F \wedge g(e) /= skip$

$$\begin{tikzcd}
& S_i \arrow[dl, leftrightarrow, "J"'] \arrow[dr, leftrightarrow, "J"] & \\
S_i' \arrow[rr, "K'(e)"] & & S_{i+1}'
\end{tikzcd}$$

$\forall e \cdot e \in F \wedge g(e) = skip$

Figure 3: Forward Simulation

Definition 4.1 (Forward Simulation)**.** *Let $M =< S, S_0, E, K >$ and machine $M' =< S', S'_0, F, K' >$. Let the event mapping function be $g \in F \to E \cup \{skip\}$. Let $J \in S \leftrightarrow S'$ be the gluing relation that relates the states of M and M', M is forward simulated by M' under K and K' provided:*

$$\begin{cases} S'_0 \subseteq J[S_0] \\ \forall e \cdot e \in F \wedge g(e) \neq skip \Rightarrow J; K'(e) \subseteq K(g(e)); J \\ \forall e \cdot e \in F \wedge g(e) = skip \Rightarrow J; K'(e) \subseteq J \end{cases}$$

In general, gluing invariants define a relational mapping between concrete and abstract states. We first define relational gluing traces in Definition 4.2 to link the states on the behavioral traces. $J(u_s, v_s)$ is used to relate the corresponding states in abstract and concrete traces. The notation $J[ts]$ is used to define the relational gluing trace image of state trace set ts, formally presented as (5).

$$J[ts] = \{v_s \mid u_s \in ts \wedge J(u_s, v_s)\} \tag{5}$$

Definition 4.2 (Relational Gluing Traces)**.** *The infinite state trace u_s is defined to be gluing related to infinite state trace v_s as $J(u_s, v_s)$, formally:*

$$J(u_s, v_s) \triangleq \forall i \cdot i \in \mathbb{N} \Rightarrow u_s(i) \mapsto v_s(i) \in J$$

Given that there are no relabelling and stuttering events in the refinement step, then for any abstract event trace $u_e \in e\_traces(M)$, the corresponding concrete state trace set is the subset of the relational gluing image of the corresponding abstract state trace set. In other words, for each concrete state trace $v_s \in s\_traces(M', u_e)$, there exists an abstract state trace u_s that is gluing related with v_s and it is an abstract trace. For the case where there is no relabelling and stuttering in the refinement step, Theorem 4.4 states that infinite state trace refinement could be proved with forward simulation. Based on the infinite state trace refinement, we provide the refinement semantics of event traces without relabelling and new events in Definition 4.3. The behavior of refined machine must be consistent with the behavior of the machine being refined during refinement [12]. Thus we use the event traces to define the refinement between machines. For machine M' to refine machine M, the concrete event trace set $e\_traces(M')$ must be the subset of the abstract event trace set $e\_traces(M)$ provided no relabelling or new events in the refinement step.

Definition 4.3 (Event Trace Refinement Without Relabelling and New Events)**.** *$M \sqsubseteq M'$ is defined as $e\_traces(M') \subseteq e\_traces(M)$.*

Theorem 4.4. *Assume the event mapping between machine M' and M does not relabel events nor does it introduce new events. Then the state trace set of M' is a subset of the relational gluing image of the state trace set of M when M is forward simulated by M', formally:*

$$\forall v_e \cdot v_e \in e\_traces(M') \Rightarrow s\_traces(M', v_e) \subseteq J[s\_traces(M, v_e)] \tag{6}$$

Proof Outline. *For $v_e \in e\_traces(M')$ we have:*

$$s\_traces(M', v_e) \subseteq J[s\_traces(M, v_e)]$$
$$\equiv \forall v_s \in s\_traces(M', v_e) \Rightarrow \exists u_s \cdot J(u_s, v_s) \wedge u_s \in s\_traces(M, v_e)$$

To prove the above, we outline the proof steps as follows:

- *We first construct a set U that contains the infinite abstract state traces or one of its prefixes related to each v_s by gluing invariant J in (7).*
- *Based on $S_0' \subseteq J[S_0]$ in Definition 4.1, $S_0 \neq \varnothing$. $s \in S_0$ is one of the elements in U. Thus the set U is not empty.*
- *Lemma 4.5 is constructed to show that infinite state traces in U are in $traces(M, u_e)$.*
- *We then show that any totally ordered set $Q \subseteq U$ has an upper bound in Lemma 4.7. That is, the main hypothesis of Zorn's lemma is satisfied.*
- *We use Zorn's Lemma to prove the existence of an infinite abstract state trace as a maximal element $u_{\mathcal{M}} \in U$ in a set of finite or infinite state traces that matches the given infinite concrete trace in Lemma 4.8. Thus $u_{\mathcal{M}}$ is an infinite trace and $u_{\mathcal{M}} \in U$, which shows that $u_{\mathcal{M}}$ is an infinite abstract state trace. Then we use $u_{\mathcal{M}}$ as a witness for the existence of u_s to prove the theorem.*

$$\begin{aligned} &U = \\ &\{u_s \mid \forall i \cdot i \geq 0 \Rightarrow u_s(0) \in S_0 \wedge u_s(i) \mapsto v_s(i) \in J \wedge u_s(i) \mapsto u_s(i+1) \in K(u_e(i))\} \\ &\cup \{w_s \mid \forall i \cdot i \geq 0 \Rightarrow w_s \in (0..k-1) \to S \wedge w_s(0) \in S_0 \wedge w_s(i) \mapsto v_s(i) \in J \\ &\wedge w_s(i) \mapsto w_s(i+1) \in K(u_e(i))\} \end{aligned} \tag{7}$$

We now proceed with the proof steps just outlined. In the first step, a set U that contains the infinite abstract traces and their prefixes related to v_s by gluing invariant J, formally presented as (7). We write $\preceq$ for the (standard) prefix order on U; that is, $u \preceq v$ when either u and v are both infinite and coincide, or u is a finite prefix of v. Based on Definition 2.3, we show that the infinite traces in U are the state traces of the abstract machine M.

Lemma 4.5. *The set U is partially ordered and the infinite traces of U are in $traces(M, u_e)$.*

Proof. The infinite traces u_s satisfy the trace properties defined in Definition 2.3, thus any $u \in \{u_s \mid \forall i \cdot i \geq 0 \Rightarrow u_s(0) \in S_0 \wedge u_s(i) \mapsto v_s(i) \in J \wedge u_s(i) \mapsto u_s(i+1) \in K(u_e(i))\}$ is in $traces(M, u_e)$. $\square$

Lemma 4.7 states that any totally ordered chain $Q \subseteq U$ has an upper bound. There are two cases for the totally ordered set Q. One case is when the chain stabilizes from some point, that is, all partial traces in the chain have size at most i for some $i \in \mathbb{N}$. In this case, the upper bound of Q are the traces of size i. The other case is when the chain does not stabilize, that is, for each $i \in \mathbb{N}$, there exists a partial trace in the set Q that has size no less than i. To help prove the second case, Lemma 4.6 proves that if any two state traces in a totally ordered set Q are defined on index $i \in \mathbb{N}$, then the states at index i in the two traces are the same. The proofs for Lemma 4.6 and 4.7 are given in Appendix A.

Lemma 4.6. *Given a totally ordered set Q of traces $u \in \mathbb{N} \to S$, if $i \in dom(u)$ and $i \in dom(u')$ where $u \in Q$ and $u' \in Q$, then $u(i) = u'(i)$.*

Lemma 4.7. *Given U in (7), any totally ordered set $Q \subseteq U$ has an upper bound.*

Given that the set U is a nonempty partially ordered set and every totally ordered set $Q \subseteq U$ has an upper bound, then there is a maximal element in U based on Zorn's Lemma [13]. Forward simulation is used to prove that the maximal element is an infinite trace in Lemma 4.8, which can be used to choose $u_s \in traces(M, u_e)$.

Lemma 4.8. *Assume M is forward simulated by M'. The set U in Formula (7) has a maximal element and that maximal element is an infinite trace.*

Proof. Based on Zorn's Lemma, then U has a maximal element $u_{\mathcal{M}} \in U$. Assume $u_{\mathcal{M}}$ is a finite trace with length k, then $u_{\mathcal{M}}(k-1) \notin dom(K(u_e(k-1)))$. Given the infinite state trace v_s and the forward simulation condition, we can prove that $u_{\mathcal{M}}(k-1) \in dom(K(u_e(k-1)))$ with the following proof, which derives a contradiction. Thus $u_{\mathcal{M}}$ is an infinite trace.

$$\begin{array}{ll} & \{u_{\mathcal{M}} \in U \text{ and } v_s \text{ is infinite}\} \\ \Rightarrow & u_{\mathcal{M}}(k-1) \mapsto v_s(k-1) \in J \wedge v_s(k-1) \mapsto v_s(k) \in K'(v_e(k-1)) \\ & \{\text{Definition 4.1 and } g(v_e(k-1)) = u_e(k-1)\} \\ \Rightarrow & \exists s \cdot s \mapsto v_s(k) \in J \wedge u_{\mathcal{M}}(k-1) \mapsto s \in K(u_e(k-1)) \\ \equiv & u_{\mathcal{M}}(k-1) \in dom(K(u_e(k-1))) \end{array}$$

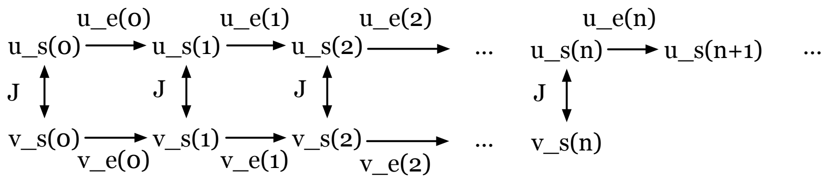
□

Proof of Lemma 4.8 means that the proof of Theorem 4.4 is now complete. Then we use Theorem 4.9 to prove that forward simulation is enough to prove a valid refinement between abstract and concrete machines. The proof for Theorem 4.9 is provided in Appendix A.

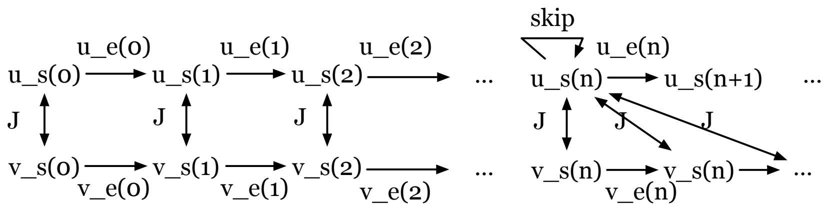
Theorem 4.9. *$M \sqsubseteq M'$ given M' is forward simulated by M and there are no relabelling and new events in the refinement step.*

4.2. Hiding Operator

In the previous section, we assume that there are no new events introduced in the refined machine. In practice, adding new events in the refined model would make the behavior steps more fine grained. Besides, relabelling also exists in generic Event-B refinements where an abstract event may be refined by multiple refined events. Figure 4 shows two cases where the concrete trace does not simulate the abstract trace. In the first case, the concrete trace comes to a deadlock. In our definition of timed systems, time should always progress. So the traces of the refined model should also be infinite traces. Based on our setting, additional conditions are required to exclude the behaviors that would



(a) Deadlock in Concrete Trace



(b) Infinite New Events in Concrete Trace

Figure 4: State Trace Refinement with Deadlock and Infinite New Events

cause deadlocks in the refined machine. In the second case, the introduced new events occur infinitely often from some point n in the concrete trace, which can only be mapped to the prefix of the abstract trace. Thus we want to show that the event traces that hide the stuttering *skip* events do correspond to abstract traces. He [14] defines a hiding operator on labeled transition systems and develops simulations that allow the concrete model to have hidden transitions. Butler shows that in CSP, hiding an infinite behavior causes the process to diverge [15]. Inspired by their work, we define a hiding operator on infinite event traces by using a hiding function h_D on an infinite set $D \subseteq \mathbb{N}$ in Definition 4.10. The hiding function h_D defines a bijection function from natural numbers to the infinite subset D. We define the function recursively. We map $h_D(0)$ to the minimal number in D since D is well-ordered. And $h_D(n+1)$ is mapped to the minimal number in the set $\{x \mid x \in D \wedge x > h_D(n)\}$. If D is finite, then h_D is not well-defined.

Definition 4.10 (Hiding Function). *Given an infinite set $D \subseteq \mathbb{N}$, the hiding* *function $h_D \in \mathbb{N} \rightarrowtail\!\!\!\!\twoheadrightarrow D$ is formally defined as:*

$$\begin{cases} h_D(n) = min(D), & n = 0 \\ h_D(n+1) = min(\{x \mid x \in D \land x > h_D(n)\}), & \forall n \cdot n \in \mathbb{N} \end{cases}$$

Firstly we want to prove that the hiding function is order isomorphic. Based on the order isomorphic function Definition 4.11 defined in [16], it is clear that the hiding function is order isomorphic when D is infinite. Here well-ordered sets are totally ordered sets whose non-empty subsets have a least element in the ordering [16]. Since D and $\mathbb{N}$ are well-ordered sets, then h_D is unique based on the unique order isomorphism theorem between well-ordered sets provided in Theorem 4.12.

Definition 4.11 (Order Isomorphic Function [16])**.** *Let P, Q be well-ordered sets. A function $f \in P \rightarrowtail\!\!\!\!\twoheadrightarrow Q$ is defined as order isomorphism as $P \cong Q$* ***iff***

$$P \cong Q \triangleq \forall x, y \cdot x \in P \land y \in P \land x < y \Rightarrow f(x) < f(y)$$

Theorem 4.12 (Unique Order Isomorphism between Well-ordered Sets [16])**.** *Let P, Q be well-ordered sets. If $P \cong Q$, there is exactly one order-isomorphism $f \in P \rightarrowtail\!\!\!\!\twoheadrightarrow Q$.*

Then we define the hiding operator in infinite event traces in Definition 4.13 by using the hiding function to define the event trace $u_e \setminus A$ with all events in A removed in u_e. The hiding operation is the backward composition of a range subtraction and the hiding function. The range subtraction removes the indexes together with the events in A. The hiding function h_D remaps the natural numbers to the remaining events. We use Lemma 4.14 to show that $v_e \setminus A$ is a well defined infinite trace provided that v_e does not end with an infinite suffix of events in A.

Definition 4.13 (Hiding Operator for Infinite Event Traces)**.** *Given an event trace $u_e \in \mathbb{N} \rightarrow E$ and a set of events A. Let $D = dom(u_e \rhd\!\!\!\!- A)$, then the infinite event trace $u_e \setminus A$ is formally defined as:*

$$u_e \setminus A \triangleq (u_e \rhd\!\!\!\!- A) \circ h_D \tag{8}$$

Lemma 4.14. *Given the infinite event trace $v_e \in \mathbb{N} \to E$, then $v_e \setminus A \in \mathbb{N} \to E$ if v_e does not have an infinite suffix of events in A.*

Proof. Assume v_e has an infinite suffix of events in A, formally:$\exists j \in \mathbb{N} \Rightarrow \forall i \cdot i > j \wedge v_e(i) \in A$. Then it is clear that $\forall k \cdot k \in dom(v_e \rhd A) \Rightarrow k \leq j$. Given that $h_D(n) = j$, then $\{x \mid x \in dom(v_e \rhd A) \wedge x > h_D(n)\} = \varnothing$. Then $h_D(n+1)$ is not well-defined, which derives a contradiction. □

4.3. Generic Refinement Semantics

In Section 4, we assume that the concrete model does not introduce new events and there is no relabelling of the events. In this section, we extend the refinement semantics 4.3 with the more generic Definition 4.15 by using the event mapping function $g \in F \to E \cup \{skip\}$. In the extended definition, we show that $M \sqsubseteq M'$ only if for any concrete event trace $v_e \in e\_traces(M')$, there exists an abstract trace $u_e \in e\_traces(M)$, and $g(v_e) \setminus skip = u_e$.

Definition 4.15 (Generic Event Trace Refinement)**.**

$$M \sqsubseteq M' \equiv \forall v_e \in e\_traces(M') \Rightarrow (g(v_e) \setminus skip) \in e\_traces(M)$$

In timed systems, the behavioral traces are infinite and time should always progress. There are two properties to be preserved during the refinement step for timed models. Firstly, the refined behavioral trace should be infinite, provided the abstract trace is infinite. Secondly, introducing new events in the refinement step should not lead to divergence. Besides forward simulation, additional conditions are required to preserve these two properties. To simplify the proof, we construct intermediate traces $traces^*(M)$ with stuttering events *skip* in Definition 4.16. The set of intermediate event or state traces of M is defined in Definition 4.17. The property that all traces in M' are convergent with respect to new events requires that an intermediate event trace $\hat{v}_e \in e\_traces^*(M)$ does not have an infinite suffix of *skip* events under the condition that the abstract machine M is not deadlocked. Based on Lemma 4.14, if $\hat{v}_e$ has an infinite suffix of *skip* events, then $\hat{v}_e \setminus skip$ is a finite trace, which cannot be a trace of an abstract machine that is not deadlocked.

Definition 4.16 (Intermediate Traces with Stuttering Events)**.** *The set of infinite traces traces*$^*(M)$ *of a machine* M *that include stuttering events is defined as:*

$$\begin{aligned} \mathit{traces}^*(M) \triangleq \{(\hat{v}_s, \hat{v}_e) \mid\ & \hat{v}_s \in \mathbb{N} \to S \wedge \hat{v}_e \in \mathbb{N} \to (E \cup \{\mathit{skip}\}) \\ & \wedge \hat{v}_s(0) \in \mathit{init} \\ & \wedge \forall i \cdot i \geq 0 \wedge \hat{v}_e(i) \neq \mathit{skip} \Rightarrow \hat{v}_s(i) \mapsto \hat{v}_s(i+1) \in K(\hat{v}_e(i)) \\ & \wedge \forall i \cdot i \geq 0 \wedge \hat{v}_e(i) = \mathit{skip} \Rightarrow \hat{v}_s(i) = \hat{v}_s(i+1)\} \end{aligned}$$

Definition 4.17 (The Set of Intermediate Event/State traces of M)**.** *A set of intermediate event traces of* M *is defined as:*

$$e\_\mathit{traces}^*(M) = \mathit{prj}2[\mathit{traces}^*(M)]$$

A set of intermediate state traces of M *for the event trace* $u_e \in e\_\mathit{traces}^*(M)$ *is defined as:*

$$s\_\mathit{traces}^*(M, u_e) = \{u_s \mid (u_s, u_e) \in \mathit{traces}^*(M)\}$$

The intermediate event trace maps concrete event labels F to abstract labels E and *skip*. We use Lemma 4.18 to show that the concrete event trace set preserves the behavior of intermediate event trace set if M' is forward simulated by M. The proof for Lemma 4.18 is provided in Appendix A.

Lemma 4.18. *Given machine* M *and* M' *with relabelling function* g *that allows new events, then* $g[e\_\mathit{traces}(M')] \subseteq e\_\mathit{traces}^*(M)$ *provided* M' *is forward simulated by* M*.*

Theorem 4.19 is used to show additional rules required to prove $M \sqsubseteq M'$ and the traces in M' are convergent with respect to new events $\mathcal{N}$. Forward simulation is used to prove trace inclusion of concrete event traces and the intermediate event traces with stuttering events. $\hat{v}_e \setminus \mathit{skip}$ is one of the abstract events only if it is an infinite trace. Based on Lemma 4.14, $\hat{v}_e \setminus \mathit{skip}$ is an infinite trace if $\hat{v}_e$ does not have an infinite suffix of *skip* events. Conditional convergence and weak fairness conditions are required to prove the infiniteness of the behavioral trace.

Theorem 4.19. *Given M with transition relation K and M' with transition relation K'. Let F be the set of event labels in M and $\mathcal{N}$ be the introduced new events in M'. $M \sqsubseteq M'$ provided the following conditions hold:*

- *M forward simulated by M'.*
- *M' is deadlock free relative to M: $J[dom(K)] \subseteq dom(K')$.*
- *M' is weakly $(F \setminus \mathcal{N})$-fair.*
- *Events $\mathcal{N}$ in machine M' are conditional convergent under the condition that events $F \setminus \mathcal{N}$ are disabled;*

Proof Outline. *We outline the proof with the following steps:*

- *We first prove that $g[e\_traces(M')] \subseteq e\_traces^*(M)$ provided M' is forward simulated by M with Lemma 4.18. Then we prove $\forall \hat{v}_e \cdot \hat{v}_e \in e\_traces^*(M) \Rightarrow \hat{v}_e \setminus skip \in e\_traces(M)$.*
- *Since M' is deadlock free relative to M, we show that traces in $e\_traces(M')$ are infinite traces when M is not deadlocked. Assume that there exists one trace $v \in traces(M')$ that is deadlocked at index l, formally presented as $v_s(l) \notin dom(K')$.*

$$\begin{array}{ll} & v_s(l) \notin dom(K') \\ \Rightarrow & \{J[dom(K)] \subseteq dom(K')\} \\ & v_s(l) \notin J[dom(K)] \\ \equiv & \{u_s(l) \mapsto v_s(l) \in J\} \\ & u_s(l) \notin dom(K) \end{array}$$

As M is not deadlocked so $\forall i \cdot i \in \mathbb{N} \Rightarrow u_s(i) \in dom(K)$, which contradicts the result that $u_s(l) \notin dom(K)$. Thus all traces in $traces(M')$ are infinite traces.

- *We then prove that the intermediate event trace $\hat{v}_e \in e\_traces^*(M)$ does not end with an infinite suffix of skip events by contradiction. Assume* *that $\hat{v}_e$ end with an infinite suffix of skip events. As $g(v_e) = \hat{v}_e$, then the concrete trace v_e ends with an infinite suffix of new events $\mathcal{N}$. In the case that $(F \setminus \mathcal{N})$ is disabled infinite many times in the suffix, based on the conditional convergence property, the new events will eventually be disabled and there cannot be an infinite suffix of new events. In the case that $(F \setminus \mathcal{N})$ is continuously enabled in the suffix, based on the weak fairness assumption some event $e \in (F \setminus \mathcal{N})$ will eventually occur. Thus $\hat{v}_e$ does not end with an infinite suffix of skip events. Based on Lemma 4.14, the intermediate event trace $\hat{v}_e \setminus skip$ is one of the abstract traces.*

New variables and events are introduced to Event-B models in data refinements. Based on the invariants $I(v)$ of machine M and gluing invariants $J(v, w)$ that relate the abstract and concrete variables between M and N, Abrial demonstrates how the translation of the forward simulation rule yields to various proof obligations, such as invariant preservation (INV), feasibility (FIS), guard strengthening (GRD) and so on, that are used by the Rodin [17] platform [1]. In this paper, we explore the refinement rules in terms of timed systems with infinite traces and infinitely many *Tick* events. Besides the forward simulation rule, the relative deadlock freedom rule, the conditional convergence rule, and weak fairness assumptions are required to prove that the refinement of a timed system still preserves the property that time can always proceed. The relative deadlock freeness rule guarantees that the refinement of a machine with infinite behavioral traces is always a machine with infinite traces. Conditional convergence and weak fairness assumptions are required to prevent new events from keeping occurring while the global clock can never proceed.

5. Refinement of Timing Properties in Event-B models

5.1. Trace Semantics of Real-time Properties

Based on Definition 2.12, machines with timing properties could be described with infinite traces. In this section, we use Definition 5.1 to define the refinement semantics of machines with real-time properties. The timed behavior of a concrete timed model should also be included in the abstract model. In our refinement strategies for timed systems with Event-B models, the abstract machine is extended with real-time trigger-response properties with no intermediate events between trigger and response events. Intermediate events are introduced in refinement steps.

Definition 5.1 (Refinement Semantics of Machines with Real-time Properties)**.** *$M \wedge TP(T, R, w, d) \sqsubseteq M' \wedge TP(T', R', w', d')$ is defined as (9).*

$$\begin{aligned} &\forall v_e \cdot v_e \in e\_traces(M') \wedge v_e \models TP(T', R', w', d') \\ \Rightarrow &\exists u_e \in e\_traces(M) \wedge u_e = (g(v_e) \setminus skip) \wedge u_e \models TP(T, R, w, d) \end{aligned} \tag{9}$$

To treat the infinite behavioral of timed traces, we use Lemma 5.2 to show convergence conditions and fairness assumptions under which $M \sqsubseteq M'$. Based on $M \sqsubseteq M'$, we also prove that the traces of the refined machine with intermediate events satisfy the trigger-response property in Lemma 5.2. Lemma 5.3 is used to show that given $M \sqsubseteq M'$, the timed property (that time should always progress) is preserved in the refinement step. The proofs for Lemma 5.2, 5.3 and 5.4 are provided in Appendix B.

Lemma 5.2. *Let M be an Event-B machine with trigger-response pair (T, R) and R is enabled immediately after execution of T. Let $g[T'] = T$ and $g[R'] = R$. We define $H' \subseteq F$ as intermediate events between T' and R' in M' and $g[H'] = \{skip\}$. Assume M is weakly fair with respect to R and M' is weakly fair with respect to $H' \cup R'$. If M' simulates M under J, H' is convergent under the condition of $\neg en(R')$ and $J[en(R)] \subseteq en(H' \cup R')$, then $M \sqsubseteq M'$ and M'* ***satisfies*** *$TR(T', R')$.*

Lemma 5.3. *Given $M \sqsubseteq M'$ and $\forall u_e \cdot u_e \in e\_traces(M) \Rightarrow timed(u_e)$, then $\forall v_e \cdot v_e \in e\_traces(M') \Rightarrow timed(v_e)$.*

Lemma 5.4 proves that all traces of M' that satisfy the timing property $TP(T', R', w', d')$ correspond to traces of M that satisfy the original timing property $TP(g[T'], g[R'], w, d)$ when $M \sqsubseteq M'$.

Lemma 5.4. *Given $M \sqsubseteq M'$, then $M \wedge TP(g[T'], g[R'], w, d) \sqsubseteq M' \wedge TP(T', R', w', d')$ if $w \leq w' \wedge d' \leq d$.*

5.2. Refinement Strategy Applied to the Case Study

Event-B has a strong and flexible refinement strategy described in [18, 19]. Based on the monotonicity of timing properties in Event-B models, we design the two-step refinement strategy to refine timed systems. In the first step, we introduce intermediate events to the abstract model while preserving the abstract timing properties with a strategy that has the following restrictions on the machines in the refinement chain $M_0 \sqsubseteq M_1 \sqsubseteq ... \sqsubseteq M_n$:

1. Each event of M_i is refined by at least one event of M_{i+1};
2. Given M_i, intermediate events are introduced as new events in M_{i+1};
3. M_{i+1} is deadlock free relative to M_i;
4. M_{i+1} is forward simulated by M_i;
5. The new events introduced in M_{i+1} are either anticipated or conditional convergent under the condition that the refined response events are disabled, M_{i+1} is weakly fair on these new events;
6. All anticipated events should be refined to conditional convergent events, no anticipated events remain in the final machine;
7. All refinement steps $M_i, 0 \leq i \leq n$ are weakly fair on the response events and *Tick* events;

In condition 5), new events introduced in the refined step are either *anticipated* or *conditional convergent* so that these events are never executed forever. A variant $V(v)$, that has to be decreased by every *convergent* event and must not be increased by *anticipated* events, needs to be introduced from new proof obligations to prevent the forever execution of new events. The details that refine *anticipated* events to *conditional convergent* events could be deferred to later refinement steps. Nevertheless, these *anticipated* events should be eventually refined to *convergent* events in the refinement chain. Detailed examples are provided in the BRP case study. Based on Theorem 4.19 and Lemma 5.2, conditions 3)-7) can be used to guarantee that for all $i \in 0..n-1$, $M_i \sqsubseteq M_{i+1}$ and M_{i+1} satisfies the trigger-response property. Then in the second step, the abstract timing properties could be refined to sequential or alternative timing properties between trigger, intermediate and response events.

The two-step refinement strategy could be used to resolve different levels of nondeterminism of intermediate events. Take the BRP case study as an example, the abstract machine with timing property (3) could be refined with several intermediate events. Figure 5 shows the time diagram of BRP in different refinement levels. In the first refinement, we introduce *rcv_current_pkt* and *snd_retry* as intermediate events to present that the packet could either be received by the receiver or resent by the transmitter. The nondeterminism choice of intermediate events leads to alternative responses. Then we refine the abstract timing properties into several sub-timing properties. In the second refinement, we use $TR(rcv\_current\_pkt, rcv\_abt)$ as the trigger-response pair and introduce *snd_retry* as the intermediate event. Moreover, the timing property is refined to sequential sub-timing properties in the second step.

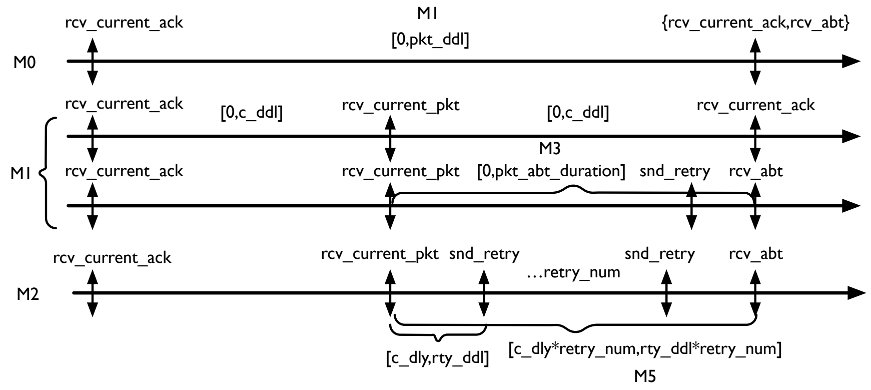


Figure 5: Time Diagram of BRP in Different Refinement Levels

The machine $M1$, given in Figure 6, introduces variables r and *rcv_file* to denote the transmitted file is denoted by *rcv_file* of length r in the receiver part. In the case that the channel is working and the receiver receives the packet (*rcv_current_pkt* event), the file pointer r in the receiver is increased by one. In the case that the channel is broken and the transmitter resends the packet (*snd_retry* event), the file pointer r is reset to s. By using the two-step refinement strategy, we first use the Rodin tool to verify that $M1$ refines $M0$ and $M1$ is deadlock-free. In $M1$, *rcv_current_pkt* and *snd_retry* are set to *anticipated* status to show that they might become convergent in later refinements. We also assume weak fairness on all the events in the machine to prove the infiniteness of the behavioral trace. In the second step, we refine the timing property (3) into three sub-timing properties in (11). Specifically, (3) is refined to (11a) and (11b) in the case that channel is working properly, and (11c) in the case that the channel is broken. We assume that the transmission delay and deadline for the channel are *c_dly* and *c_ddl*, respectively. Once the receiver received the packet, it should abort within *pkt_abt_duration* if the channel is broken. The relations between these timing properties are shown in (10).

$$\begin{cases} c\_ddl > c\_dly > 0 \\ pkt\_ddl > 2 * c\_ddl \\ pkt\_abt\_duration + c\_ddl < pkt\_ddl \end{cases} \tag{10}$$

We use @$grd2\_1$ and @$grd2\_2$ of *tick* event in $M1$ to replace @$grd1\_1$ of *tick* event in $M0$. @$inv2\_5$ and @$inv2\_7$ are used to show that (11a) and (11b) are satisfied in $M1$. @$inv2\_4$ and @$inv2\_6$ serve as auxiliary invariants to support @$inv1\_5$ and @$inv1\_7$. In the case that the channel is broken, we use @$grd2\_3$ and @$grd2\_4$ of *tick* event in $M1$ to replace @$grd1\_2$ of *tick* event in $M0$. @$inv2\_9$ is used to present timing property (11c).

$$TP(rcv\_current\_ack, rcv\_current\_pkt, 0, c\_ddl) \tag{11a}$$

$$TP(rcv\_current\_pkt, rcv\_current\_ack, 0, c\_ddl) \tag{11b}$$

$$TP(rcv\_current\_pkt, rcv\_abt, 0, pkt\_abt\_duration) \tag{11c}$$

Timing is important to synchronize the status of transmitter and receiver in BRP. In $M2$, we want to guarantee that the receiver will abort based on the

```
@inv2_4 s_st=working ∧c_st=working ∧r=s ⇒clk−pkt_tc≤  c_ddl
@inv2_5 pkt_tc≤  pkt_rcv_t ⇒pkt_rcv_t − pkt_tc≤  c_ddl
@inv2_6 s_st=working ∧c_st=working ∧r=s+1 ⇒clk−pkt_rcv_t≤  c_ddl
@inv2_7 pkt_rcv_t≤  pkt_tc ∧s≠0 ⇒pkt_tc − pkt_rcv_t≤  c_ddl
@inv2_8 r_st=working ∧c_st=failure ∧r=s+1 ⇒clk−pkt_rcv_t≤  pkt_abt_duration
@inv2_9 pkt_rcv_t≤ abt_t ⇒abt_t− pkt_rcv_t≤  pkt_abt_duration
```

```
anticipated event rcv_current_pkt
where
  @grd2_1 r_st=working
  @grd2_2 c_st=working
  @grd2_3 s_st=working
  @grd2_4 r+1<N
  @grd2_5 r=s
then
  @act2_1 r:= r+1
  @act2_2 rcv_file:= rcv_file ∪{r+1 ↦ file(
        s+1)}
  @act2_3 pkt_rcv_t:= clk
end

anticipated event snd_retry
where
  @grd2_2 c_st=failure
  @grd2_3 r+1<N
then
  @act2_1 r:= s
  @act2_2 rcv_file:=  1..s ◁ file
end
```

```
event rcv_current_ack extends
      rcv_current_ack
where
  @grd2_1 r=s+1
end

event rcv_abt extends rcv_abt
where
  @grd2_1 r=s+1
end

event tick refines tick
where
  @grd2_1 s_st=working ∧c_st=working ∧r
        =s ⇒clk+1−pkt_tc≤  c_ddl
  @grd2_2 s_st=working ∧c_st=working ∧r
        =s+1 ⇒clk+1−pkt_rcv_t≤  c_ddl
  @grd2_3 r_st=working ∧c_st=failure ∧r=
        s ⇒clk+1−pkt_tc≤  c_ddl
  @grd2_4 r_st=working ∧c_st=failure ∧r=
        s+1 ⇒clk+1−pkt_rcv_t≤
        pkt_abt_duration
then
  @act1_1 clk:= clk+1
end
```

Figure 6: Machine $M1$ that Introduces *rcv_current_pkt* and *snd_retry* as Intermediate Events

timing information on its end. In this refinement, we first refine the *anticipated* event *snd_retry* to *convergent* event by introducing variable *rty_cnt* to denote a retry counter. The transmitter will decide to abort if *rty_cnt* reaches the constant *rty_num*. We use $retry\_num - rty\_cnt$ as the variant and the convergent event *snd_retry* always decreases the variant.

Then in the second step, we refine timing property (11c) to timing property (12a) and (12b). It takes the receiver at least $2 * c\_dly$ time units but at most *retry_ddl* time units before knowing that the transmitter has not sent the new packet. And after all re-send attempts the receiver guarantees that the transmitter has aborted. Then it aborts after $2 * c\_dly * rty\_num$ time units and within $retry\_ddl * rty\_num$ time units. In the model, we replace @$grd2\_4$ of *tick* event in $M1$ with @$grd3\_1$ of *tick* event in $M2$ with the assumption that $rty\_ddl > 2 * c\_ddl$ and $pkt\_abt\_duration \geq retry\_num * rty\_ddl$. @$inv3\_2$ and @$inv3\_3$ are used to present timing property (12a) and (12b) respectively. @$inv3\_1$ serves as the auxiliary invariant for @$inv3\_2$.

$$TP(rcv\_current\_pkt, snd\_retry, 2 * c\_dly * rty\_cnt, retry\_ddl * rty\_cnt) \quad (12a)$$

$$TP(rcv\_current\_pkt, rcv\_abt, 2 * c\_dly * rty\_num, retry\_ddl * rty\_num) \quad (12b)$$

In the last step, we refine the *anticipated* event *rcv_current_pkt* to convergent event with the variant $N - r$. Then the convergent event *rcv_current_pkt* always decreases the variant.

6. Related Work

Timing issues are critical in real-time systems such as medical devices and high precision control systems. Timing properties should be reasoned about together with the system to guarantee the safety and responsiveness of the whole system. Xie et al. extended the TiMo, a process algebra for mobile distributed systems, with the deadline time constraints on actions [20]. Sheng et al. presented the algebraic semantics for multithreaded discrete event simulation

```
invariants
@rty_cnt_type rty_cnt∈ ℕ
@inv3_1 r_st=working ∧c_st=failure ∧r=s+1 ⇒clk−pkt_rcv_t≤  rty_ddl ∗ rty_cnt
@inv3_2 pkt_rcv_t≤  snd_rty_t ∧r=s+1 ⇒snd_rty_t− pkt_rcv_t≤  rty_ddl ∗ rty_cnt
@inv3_3 pkt_rcv_t≤ abt_t ⇒abt_t− pkt_rcv_t≤  rty_ddl ∗ retry_num
variant
retry_num−rty_cnt
```

```
convergent event snd_retry extends
      snd_retry
where
  @grd3_1 rty_cnt<retry_num
  @grd3_2 clk≥  pkt_rcv_t+ c_dly
  @grd3_3 snd_rty_t>pkt_rcv_t⇒clk≥
          snd_rty_t +2∗c_dly
then
  @act3_1 rty_cnt:= rty_cnt+1
  @act3_2 snd_rty_t:= clk
end

event rcv_abt extends rcv_abt
where
  @grd3_1 rty_cnt=retry_num
end
```

```
event rcv_current_ack extends rcv_current_ack
then
  @act4_1 rty_cnt:= 0
end
event tick refines tick
where
  @grd2_1 s_st=working ∧c_st=working ∧r=s ⇒
          clk+1−pkt_tc≤  c_ddl
  @grd2_2 s_st=working ∧c_st=working ∧r=s+1
           ⇒clk+1−pkt_rcv_t≤  c_ddl
  @grd2_3 r_st=working ∧c_st=failure ∧r=s ⇒
          clk+1−pkt_tc≤  c_ddl
  @grd3_1 r_st=working ∧c_st=failure ∧r=s+1
          ⇒clk+1−pkt_rcv_t≤  rty_ddl ∗ rty_cnt
then
  @act1_1 clk:= clk+1
end
```

Figure 7: Machine $M2$ that Uses *snd_retry* as Intermediate Events between *rcv_current_pkt* and *rcv_abt*

language, where time is used as a delay component to suspend the execution of the program [21]. Alur developed the formalism of timed automata, which integrate global clocks into the state transition systems, to model and analyze the timing behavior of real-time systems [22]. Henzinger et al. propose the idea of Timed Transition Systems (TTS), which put quantitative lower-bound and upper-bound timing constraints on transitions [23]. Based on the idea of TTS, Ostroff summarizes composition rules, and refinement rules that can be used to model discrete real-time systems with the support of model checking tools[24]. These approaches are used to verify timing properties on individual transitions of the system, while our approach verifies timing properties between different transitions in a system, which could be used as high-level requirement specification.

Event-B is a modeling language that supports modeling refinement but lacks explicit support for expressing and verifying timing constraints [25]. Abadi and Lamport specified and reasoned about real-time systems by representing time as an ordinary variable [26]. Influenced by their work, Butler and Falampin proposed an approach to model and refine timing properties in classical B [5], which adds a clock variable representing the current time and an operation that advances the clock [27]. 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 [28, 25]. Sarshogh and Butler developed a trigger-response pattern to extend Event-B models with discrete timing properties such as deadline, delay, and expiry [28]. 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 [25]. However, neither Sarshogh nor Sulskus resolves the issue of *Zeno* behavior in timed systems. On top of their contribution, we add fairness assumptions and relative deadlock freedom to eliminate *Zeno* behavior in Event-B models with real-time trigger-response properties.

Refinement mappings have been used to prove that one specification implements another [29]. To address the problem that a refinement mapping does not exist because the concrete specification hides the history or future details of the abstract specification, Abadi and Lamport proposed to use auxiliary variables such as history variables or prophecy variables with stuttering to create a valid refinement mapping [30]. Based on timed automata, Lynch defined a formal timed transition system and used it to develop several simulation proof techniques such as forward and backward simulations, hybrid forward-backward, and backward-forward simulations [31]. Back and Xu investigated refinement calculus for fair action systems [32]. Forward and backward simulations are extended for verifying termination and correctness of fair action systems. Our work extends the refinement semantics of Event-B models in terms of infinite behavioral traces with forward simulation and additional conditions.

7. Conclusion

To summarize, our work with refinement semantics and trace semantics of timing properties has yielded two main contributions. First, additional conditions are provided to verify refinement in Event-B models in terms of infinite behavioral traces. Zorn's Lemma is used together with forward simulation to prove infinite state/event trace refinement in refinement steps. Second, we use real-time trigger-response properties to formalize high-level timing properties between different transitions in a discrete system. Our work applies conditional convergence, deadlock freedom and weak fairness assumptions to exclude the *Zeno* behavior in timed systems. Additional conditions related to the time constraints are also provided to preserve the consistency of timing properties in different refinement levels. Also, we develop a two-step refinement strategy to refine real-time systems. Based on the refinement semantics and refinement strategy, we use the bounded retransmission protocol case study to illustrate the refinement conditions.

We impose weak fairness assumptions on intermediate events, response events, and the *Tick* event to ensure that they do get executed sufficiently often when enabled. However, our approach does not rule out several rounds of trigger-response events occurring within one clock tick. One solution to rule out these behaviors is to force the lower bound of real-time trigger-response property $w > 0$. In the cases that there is no lower bound of delay time for trigger-response properties, some more substantial fairness restrictions such as finitary fairness [33] or bounded fairness [34] could be used to guarantee that the *Tick* event gets the chance to proceed. Also, our previous work introduces a nondeterministic queue-based scheduling policy with some additional gluing invariants to refine real-time properties in concurrent systems [35].

This paper introduced additional proof obligations required to model discrete timing properties, which are not supported natively by the Rodin platform. A plugin could be built to extend the Rodin platform to support discrete-time modeling with trigger-response properties as well as conditional convergent event modeling in Event-B.

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.

References

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

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

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

[4] 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/`

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

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

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

[8] N. Abid, S. Dal-Zilio, D. L. Botlan, Real-time specification patterns and tools, CoRR abs/1301.7534 (2013). `arXiv:1301.7534`.
URL `http://arxiv.org/abs/1301.7534`

[9] C. Zhu, M. J. Butler, C. Cîrstea, Semantics of real-time trigger-response properties in Event-B, in: 2018 International Symposium on Theoretical Aspects of Software Engineering, TASE 2018, Guangzhou, China, August 29-31, 2018, 2018, pp. 150–155. `doi:10.1109/TASE.2018.00028`.
URL `https://doi.org/10.1109/TASE.2018.00028`

[10] P. R. D'Argenio, J.-P. Katoen, T. C. Ruys, J. Tretmans, The bounded retransmission protocol must be on time!, in: International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 1997, pp. 416–431.

[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] K. Robinson, System modelling & design using Event-B, The University of New South Wales. Recuperado en agosto de (2012).

[13] K. Conrad, Zorn's lemma and some applications, Expository papers (2016).

[14] J. He, Various simulations and refinements, in: Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems), Springer, 1989, pp. 340–360.

[15] M. Butler, A CSP approach to action systems, Ph.D. thesis, Oxford University (1992).
URL `https://eprints.soton.ac.uk/250974/`

[16] K. Devlin, The joy of sets: fundamentals of contemporary set theory, Springer Science & Business Media, 2012.

[17] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, L. Voisin, Rodin: an open toolset for modelling and reasoning in Event-B, STTT 12 (6) (2010) 447–466.

[18] S. Hallerstede, M. Leuschel, D. Plagge, Validation of formal models by refinement animation., Sci. Comput. Program. 78 (3) (2013) 272–292.

[19] S. Schneider, H. Treharne, H. Wehrheim, The behavioural semantics of Event-B refinement, Formal aspects of computing 26 (2) (2014) 251–280.

[20] W. Xie, S. Xiang, H. Zhu, A utp approach for rtimo, Formal Aspects of Computing 30 (6) (2018) 713–738.

[21] F. Sheng, H. Zhu, J. He, Z. Yang, J. P. Bowen, Theoretical and practical aspects of linking operational and algebraic semantics for mdesl, ACM Transactions on Software Engineering and Methodology (TOSEM) 28 (3) (2019) 1–46.

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

[23] T. A. Henzinger, Z. Manna, A. Pnueli, Timed transition systems, in: J. W. de Bakker, C. Huizing, W. P. de Roever, G. Rozenberg (Eds.), Real-Time: Theory in Practice, Springer Berlin Heidelberg, Berlin, Heidelberg, 1992, pp. 226–251.

[24] J. S. Ostroff, Composition and refinement of discrete real-time systems, ACM Trans. Softw. Eng. Methodol. 8 (1) (1999) 1–48. `doi:10.1145/295558.295560`.
URL `http://doi.acm.org/10.1145/295558.295560`

[25] 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/`

[26] 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`

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

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

[29] L. Lamport, Specifying concurrent program modules, ACM Transactions on Programming Languages and systems (2016).

[30] M. Abadi, L. Lamport, The existence of refinement mappings, Theoretical Computer Science 82 (2) (1991) 253–284.

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

[32] R. J. Back, Q. Xu, Refinement of fair action systems, Acta Informatica 35 (2) (1998) 131–165.

[33] R. Alur, T. A. Henzinger, Finitary fairness, ACM Transactions on Programming Languages and Systems (TOPLAS) 20 (6) (1998) 1171–1194.

[34] N. Dershowitz, D. N. Jayasimha, S. Park, Bounded Fairness, Springer Berlin Heidelberg, 2003, pp. 304–317. doi:10.1007/978-3-540-39910-0_14.

[35] C. Zhu, M. Butler, C. Cirstea, Formalizing hierarchical scheduling for refinement of real-time systems, Science of Computer Programming (2020) 102390.

Appendix A. Proofs for Section 4 (Trace Refinement Semantics)

Lemma 4.6. *Given a totally ordered set Q of traces $u \in \mathbb{N} \to S$, if $i \in dom(u)$ and $i \in dom(u')$ where $u \in Q$ and $u' \in Q$, then $u(i) = u'(i)$.*

Proof.

$$\begin{aligned}
& \forall i, u, u' \cdot i \in dom(u) \wedge i \in dom(u') \wedge u \in Q \wedge u' \in Q \\
\equiv \quad & \{\text{Fix } i \in dom(u) \wedge i \in dom(u')\} \\
& \forall u, u' \cdot u \in Q \wedge u' \in Q \\
\Rightarrow \quad & \{\text{Q is totally ordered set}\} \\
& u \preceq u' \vee u' \preceq u \\
\Rightarrow \quad & \begin{cases} u(i) = u'(i) & \text{where } u \preceq u' \wedge i \in dom(u) \\ u(i) = u'(i) & \text{where } u' \preceq u \wedge i \in dom(u') \end{cases} \\
\equiv & u(i) = u'(i)
\end{aligned}$$

□

Lemma 4.7. *Given U in (7), any totally ordered set $Q \subseteq U$ has an upper bound.*

Proof. Case 1: The chain stabilizes from some point. That is, all partial traces in the chain have size at most i, formally:

$$\exists i \in \mathbb{N} \Rightarrow (\forall u \in Q \Rightarrow \#(u) \leq i) \wedge (\exists u \in Q \Rightarrow \#(u) = i)$$

In this case, the upper bound of the chain is the partial trace whose size is with length i. As the set Q is totally ordered, so $\forall u, u' \in Q \Rightarrow u \preceq u' \vee u' \preceq u$. If the size of a trace is larger than i, then it contradicts the assumption that $\forall u \in Q \Rightarrow \#(u) \leq i$.

Case 2: The chain does not stabilize. For each $i \in \mathbb{N}$, there exists a partial trace in the chain that has size no less than i, formally:

$$\forall i \cdot i \in \mathbb{N}, \exists u \in Q \wedge \#(u) \geq i \tag{A.1}$$

In this case we define a u_ω as $\forall i, u \cdot i \in \mathbb{N} \wedge \#(u) \geq i \Rightarrow u_\omega(i) = u(i)$. u_ω is well defined based on Lemma 4.6 as for all traces that are defined in index i, the state in index i is the same. First we prove that $u_\omega \in U$.

$$\begin{aligned}
& \forall i \cdot i \in \mathbb{N}, \exists u \in Q \wedge \#(u) \geq i \\
\equiv \quad & \{\text{Definition for } U \text{ in } (7)\} \\
& \forall i \cdot i \in \mathbb{N}, \exists u \wedge u(i) \mapsto v_s(i) \in J \wedge u(i) \mapsto u(i+1) \in K(u_e(i)) \\
\equiv \quad & \{\text{let } u_\omega(i) = u(i) \text{ and } u_\omega(i+1) = u(i+1) \text{ where } \#(u) \geq i+1\} \\
& \forall i \cdot i \geq 0 \Rightarrow u_\omega(i) \mapsto v_s(i) \in J \wedge u_\omega(i) \mapsto u_\omega(i+1) \in K(u_e(i))
\end{aligned}$$

Based on the Definition of u_ω, $\forall u \in Q \Rightarrow u \preceq u_\omega$. Thus u_ω is the upper bound for the totally order set Q. □

Theorem 4.9. *$M \sqsubseteq M'$ given M' is forward simulated by M and there are no relabelling and new events in the refinement step.*

Proof.

$$\begin{aligned}
& \forall v \cdot v \in e\_traces(M') \\
\equiv \quad & \{\text{Fix } v \in e\_traces(M') \text{and Lemma 2.6}\} \\
& s\_traces(M', v) \neq \varnothing \\
\Rightarrow \quad & \{\text{Infinite State Trace Refinement: Theorem 4.4}\} \\
& s\_traces(M, v) \neq \varnothing \\
\Rightarrow \quad & \{\textit{Lemma } 2.6\} \\
& e\_traces(M) \neq \varnothing \\
\equiv \quad & \exists u \in e\_traces(M) \wedge u = v
\end{aligned}$$

□

Lemma 4.18. *Given machine M and M' with relabelling function g that allows new events, then $g[e\_traces(M')] \subseteq e\_traces^*(M)$ provided M' is forward simulated by M.*

Proof.

$$\forall v_e \in e\_traces(M')$$

$$\Rightarrow \quad \{\text{Fix } v_e \in e\_traces(M') \text{ and Lemma 2.6}\}$$

$$s\_traces(M', v_e) \neq \varnothing$$

$$\Rightarrow \quad \{\text{Infinite State Trace Refinement: Theorem 4.4}\}$$

$$s\_traces^*(M, g(v_e)) \neq \varnothing$$

$$\Rightarrow \quad \{\text{Lemma 2.6}\}$$

$$\exists u_e \in e\_traces^*(M) \wedge u_e = g(v_e)$$

□

Appendix B. Proofs for Section 5 (Refinement of Timing Properties in Event-B models)

Lemma 5.2. *Let M be an Event-B machine with trigger-response pair (T, R) and R is enabled immediately after execution of T. Let $g[T'] = T$ and $g[R'] = R$. We define $H' \subseteq F$ as intermediate events between T' and R' in M' and $g[H'] = \{skip\}$. Assume M is weakly fair with respect to R and M' is weakly fair with respect to $H' \cup R'$. If M' simulates M under J, H' is convergent under the condition of $\neg en(R')$ and $J[en(R)] \subseteq en(H' \cup R')$, then $M \sqsubseteq M'$ and M'* ***satisfies*** *$TR(T', R')$.*

Proof. H' are new events introduced to M'. Based on Theorem 4.19, $M \sqsubseteq M'$.

$$\{M \sqsubseteq M'\}$$

$$\equiv \quad \forall v_e \cdot v_e \in e\_traces(M') \Rightarrow \exists u_e \cdot u_e \in e\_traces(M) \wedge u_e = g(v_e \setminus \mathcal{H}') \wedge u_e \models TR(T, R)$$

$$\equiv \quad \{\text{ Fix } v_e \in e\_traces(M') \text{ and Definition 2.9}\}$$

$$\forall i \cdot i \geq 0 \wedge u_e(i) \in T \Rightarrow (\exists j \cdot j > i \wedge u_e(j) \in R \wedge \forall k \cdot i < k < j \Rightarrow u_e(k) \notin T)$$

$$\Rightarrow \quad \{u_e = g(v_e \setminus \mathcal{H}') \text{ , let } D = dom(v_e \rhd H')\}$$

$$\forall i \cdot i \geq 0 \wedge u_e(i) \in T \wedge v_e(h_D(i)) \in g^{-1}[T] \Rightarrow (\exists j \cdot j > i \wedge u_e(j) \in R \wedge u_e(h_D(j)) \in g^{-1}[R]$$
$$\wedge \forall k \cdot h_D(i) < k < h_D(j) \Rightarrow u_e(k) \notin g^{-1}[T])$$

$$\equiv v_e \models TR(T', R')$$

□

Lemma 5.3. *Given* $M \sqsubseteq M'$ *and* $\forall u_e \cdot u_e \in e\_traces(M) \Rightarrow timed(u_e)$*, then* $\forall v_e \cdot v_e \in e\_traces(M') \Rightarrow timed(v_e)$*.*

Proof.

$$
\begin{aligned}
& \{M \sqsubseteq M'\} \\
\equiv\quad & \forall v_e \cdot v_e \in e\_traces(M') \Rightarrow \exists u_e \cdot u_e \in e\_traces(M) \wedge u_e = g(v_e \setminus \mathcal{H}') \wedge timed(u_e) \\
\equiv\quad & \{ \text{ Fix } v_e \in e\_traces(M') \text{ and Definition 2.10}\} \\
& \forall i \cdot i \in \mathbb{N} \Rightarrow (\exists j \cdot j \geq i \wedge u_e(j) = Tick) \\
\Rightarrow & \{u_e = g(v_e \setminus \mathcal{H}') \text{ , let } D = dom(v_e \rhd H')\} \\
& \forall i \cdot i \in \mathbb{N} \Rightarrow (\exists j \cdot j \geq i \wedge u_e(j) = Tick \wedge v_e(h_D(j)) = Tick) \\
\equiv & timed(v_e)
\end{aligned}
$$

□

Lemma 5.4. *Given* $M \sqsubseteq M'$*, then* $M \wedge TP(g[T'], g[R'], w, d) \sqsubseteq M' \wedge TP(T', R', w', d')$ *if* $w \leq w' \wedge d' \leq d$*.*

Proof.

$$
\begin{array}{ll}
 & \forall v_e \cdot v_e \in e\_traces(M') \wedge v_e \models TP(T', R', w', d') \\
\equiv & \{\text{Fix } v_e \in e\_traces(M') \wedge v_e \models TP(T', R', w', d')\} \\
 & \forall i \cdot i \geq 0 \wedge v_e(i) \in T' \Rightarrow \\
 & \exists j \cdot j > i \wedge v_e(j) \in R' \wedge (w' \leq \#(v_e[i,j] \upharpoonright Tick) \leq d') \\
\Rightarrow & \{M \sqsubseteq M'\} \\
 & \exists u_e \cdot u_e \in e\_traces(M) \wedge u_e = g(v_e \setminus \mathcal{H}') \\
\Rightarrow & \{u_e = g(v_e \setminus \mathcal{H}') \text{ , let } D = dom(v_e \rhd H')\} \\
 & \forall i \cdot i \geq 0 \wedge v_e(i) \in T' \wedge u_e(h_D^{-1}(i)) \in g[T'] \Rightarrow \\
 & \exists j \cdot j > i \wedge v_e(j) \in R' \wedge u_e(h_D^{-1}(j)) \in g[R'] \wedge (w' \leq \#(u_e[h_D^{-1}(i), h_D^{-1}(j)] \upharpoonright Tick) \leq d') \\
\Rightarrow & \{w \leq w' \wedge d' \leq d\} \\
 & \forall i \cdot i \geq 0 \wedge v_e(i) \in T' \wedge u_e(h_D^{-1}(i)) \in g[T'] \Rightarrow \\
 & \exists j \cdot j > i \wedge v_e(j) \in R' \wedge u_e(h_D^{-1}(j)) \in g[R'] \wedge (w \leq \#(u_e[h_D^{-1}(i), h_D^{-1}(j)] \upharpoonright Tick) \leq d) \\
\equiv & \{\text{Lemma 5.3 and Definition 2.11}\} \\
 & u_e \models TP(g[T'], g[R'], w, d)
\end{array}
$$

□