Managing LTL properties in Event-B refinement

Steve Schneider¹, Helen Treharne¹, Heike Wehrheim², and David M. Williams³

¹ University of Surrey, England, UK
² University of Paderborn, Germany
³ VU University Amsterdam

Abstract. Refinement in Event-B supports the development of systems via proof based step-wise refinement of events. This refinement approach ensures safety properties are preserved, but additional reasoning is required in order to establish liveness and fairness properties. In this paper we present results which allow a closer integration of two formal methods, Event-B and linear temporal logic. In particular we show how a class of temporal logic properties can carry through a refinement chain of machines. Refinement steps can include introduction of new events, event renaming and event splitting. We also identify a general liveness property that holds for the events of the initial system of a refinement chain. The approach will aid developers in enabling them to verify linear temporal logic properties at early stages of a development, knowing they will be preserved at later stages. We illustrate the results via a simple case study.

1 Introduction

Event-B [1] is a step-wise development method with excellent tools: Rodin platform [2] providing proof support and ProB [11] providing model checking. As Hoang and Abrial [10] clearly state the focus of verification within Event-B has been on the safety properties of a system to ensure that “something (bad) never happens”. Typically, this has been done via the discharging of proof obligations. Nonetheless, the use of linear temporal logic (LTL) to specify temporal liveness properties has also been prevalent, for example in its application within the ProB tool [12]. The challenge is to identify more natural ways of integrating Event-B and LTL, so that LTL properties can be preserved by Event-B refinement, which is not currently the case in general.

Event-B describes systems in terms of machines with state, and events which are used to update the state. Events also have guards, which are conditions for the event to be enabled. One (abstract) machine may be refined by another (concrete) machine, using a refinement step. A linking invariant captures how the abstract and concrete states are related, and each abstract event must be refined by one or more concrete events whose state transformations match the abstract one in the sense of preserving the linking invariant. Refinement is transitive, so a sequence of refinement steps, known as a refinement chain, will result in a concrete machine which is a refinement of the original abstract one.
A particular feature provided by Event-B is the introduction of new events in a refinement step—events which do not refine any abstract event. This allows for refinements to add finer levels of granularity and concretisation as the design develops; there are many examples in [1]. These new events are invisible at the abstract level (they correspond to the abstract state not changing), and we generally need to verify that they cannot occur forever. Event-B makes use of labels to keep track of the status of events as a refinement chain progresses. Event-B labels are anticipated, convergent and ordinary. The labelling of events in Event-B form part of the core of a system description but their inclusion is primarily to support the proof of safety properties and ensuring that events cannot occur forever: convergent events must decrease a variant and anticipated events cannot increase it. In this paper all newly introduced events must be convergent or anticipated, and anticipated events must become convergent at some stage. As an initial example, consider a Lift machine with two events top and ground, representing movement to the top and to the ground floor. This can be refined by a machine Lift’ introducing two new anticipated events openDoors and closeDoors. The events top and ground are blocked when the doors are open, but enabled when the doors are closed.

Linear temporal logic provides a specification language for capturing properties of executions of systems, and is appropriate for reasoning about liveness and fairness. For example, we might verify for Lift that whenever top occurs, then eventually ground will occur. However, this is not guaranteed for its refinement Lift’: it may be that the doors open and close repeatedly forever following the top event, thus never reaching the next ground event. Alternatively it may be that the system deadlocks with the doors open, again preventing ground from occurring. Hence we see that LTL properties are not automatically preserved by Event-B refinement. In the first case we would require some assurance that openDoors and closeDoors cannot repeat forever without the lift moving; in the second case we require some liveness property on closeDoors to prevent termination with the doors open.

In this paper we present results for when temporal logic properties can be carried through Event-B refinement chains. The results generalise to events that are split—refined by several events—during a refinement chain. We also identify conditions on temporal logic properties that make them suitable for use in a refinement chain, since some properties are not preserved by Event-B refinement (for example, the property “closeDoor never occurs” holds for Lift but not for its refinement Lift’). The results are underpinned by our process algebra understanding of the Event-B semantics, in particular the traces, divergences and infinite traces semantics used for CSP and applied to Event-B in [15].

The paper is organised as follows: Section 2 provides the necessary Event-B refinement background and the refinement strategy we use in the paper. Section 3 introduces a running example. Section 4 defines the LTL we use. Sections 5 and 6 present and illustrate the main theoretical results. Proofs are included in the Appendix. We put our work into the context of related work in Section 7 and our future work in Section 8.
2 Event-B

2.1 Event-B Machines

An Event-B development is defined using \textit{machines}. A machine $M$ contains a vector of variables and a set of events. The \textit{alphabet} of $M$, $\alpha_M$, is the set of events defined in $M$. Each event $\text{evt}_i$ has the general form $\text{evt}_i \triangleq \text{any } x \text{ where } G_i(x, v) \text{ then } v : \text{BA}_i(v, x, v') \text{ end}$, where $x$ represents the parameters of the event, the guard $G_i(x, v)$ is the condition for the event to be enabled. The body is given by $v : \text{BA}_i(v, x, v')$ whose execution assigns to $v$ any value $v'$ which makes the \textit{before-after} predicate $\text{BA}_i(v, x, v')$ true. This simplifies to $\text{evt}_i \triangleq \text{when } G_i(v) \text{ then } v : \text{BA}_i(v, v') \text{ end}$ when there are no parameters, since the guard and the \textit{before-after} predicate does not refer to the parameters $x$.

Variables of a machine are initialised in an initialisation event $\text{init}$ and are constrained by an invariant $I(v)$. The Event-B approach to semantics is to associate proof obligations with machines. The key proof obligation, $\text{INV}$, is that all events must preserve the invariant. There is also a proof obligation on a machine with respect to deadlock freedom which means that a guard of at least one event in $M$ is always enabled. When this obligation holds $M$ is \textit{deadlock free}.

2.2 Event-B Refinement

An Event-B development is a sequence of B machines linked by a refinement relationship. In this paper we use $M$ and $M'$ when referring to a refinement between an \textit{abstract} machine $M$ and a \textit{concrete} machine $M'$ whereas a chain of refinements is referred to using numbered subscripts, i.e., $M_0, M_1, \ldots, M_n$, to represent the specific refinement levels.

A refinement machine can introduce new events and split existing events. We omit the treatment of merging events in this paper. New events are treated as refinements of $\text{skip}$, i.e., $\text{evt}_i'$ does not refine an event in $M$. Note that when splitting events, $M'$ has several events $\text{evt}_i'$ refining a single event $\text{evt}_i$.

A machine $M$ is considered to be refined by $M'$ if the given \textit{linking invariant} $J$ on the variables between the two machines is established by their initialisation, and preserved by all events. This requirement is captured by the $\text{INV}_\text{REF}$ proof obligation. Formally, we denote the refinement relation between two machines, written $M \bowtie M'$, when all the following proof obligations hold: feasibility $\text{FIS}_\text{REF}$, guard strengthening $\text{GRD}_\text{REF}$ and simulation $\text{INV}_\text{REF}$. Feasibility of an event is the property that, if the event is enabled (i.e., the guard is true), then there is some after-state. Guard strengthening requires that when a concrete event is enabled, then so is the abstract one. Finally, simulation ensure the occurrence of events in the concrete machine can be matched in the abstract one (including the initialisation event). Further details of these proof obligations can be found in [1].

In Section 4 we introduced the three kinds of labelling of events in Event-B: \textit{anticipated} ($a$), \textit{convergent} ($c$) and \textit{ordinary} ($o$) and noted that convergent events are those which must not execute forever whereas \textit{anticipated} events provide a
means of deferring consideration of divergence-freedom until later refinement steps. The proof obligation that deals with divergences is $\text{WFD}_\text{REF}$. It requires that the proposed variant $V$ of a refinement machine satisfies the appropriate properties: that it is a natural number, that it decreases on occurrence of any convergent event, and that it does not increase on occurrence of any anticipated event. Therefore, we augment the previous refinement relation with $\text{WFD}_\text{REF}$ such that $M \preceq_w M'$. Ordinary events can occur forever and therefore $\text{WFD}_\text{REF}$ is not applicable for such events.

### 2.3 Event-B Development strategies

Event-B has a strong but flexible refinement strategy which is described in [9]. In [15] we also discussed different Event-B refinement strategies and characterised them with respect to the approaches documented by Abrial in [1] and supported by the Rodin tool. In this paper we focus on the simplest strategy, and the one most commonly used. The strategy has the following set of restrictions on a refinement chain $M_0 \preceq_w M_1 \preceq_w \ldots \preceq_w M_n$:

1. all events in $M_0$ are labelled ordinary. This set of events is referred to as $O_0$.
2. each event of $M_i$ is refined by at least one event of $M_{i+1}$;
3. each new event in $M_i$ is either anticipated or convergent, where $i > 0$;
4. each event in $M_{i+1}$ which refines an anticipated event of $M_i$ is itself either convergent or anticipated;
5. refinements of convergent or ordinary events of $M_i$ are ordinary in $M_{i+1}$.
6. no anticipated events remain in the final machine.

Figure 1 illustrates our development strategy for a vending machine, detailed in Section 3, where $C_i$ is the set of convergent events within $M_i$, and $O_i$ is the set of ordinary events within $M_i$.

For example, $O_0 = \{ \text{selectBiscuit, selectChoc, dispenseBiscuit, dispenseChoc} \}$ and $C_0 = \emptyset$ in $VM_1$. In $VM_2$ we note that $C_1 = \{ \text{refund} \}$. In $VM_3$ we note that $C_2 = \{ \text{refill} \}$ and in $VM_4$ we have $C_3 = \{ \text{pay} \}$. Thus we denote $C_{\text{all}} = C_1 \cup C_2 \cup C_3$. 

![Fig. 1: Events and their annotations in the Vending Machine development](image-url)
2.4 Event-B Semantics

In this paper we define a trace of $M$ to be either an infinite sequence of events $(a, c$ or $o)$, i.e., $(e_0, e_1, \ldots)$ or a finite sequence of events, i.e., $(e_0, \ldots, e_{k-1})$ where the machine $M$ deadlocks after the occurrence of the final event. Traces correspond to maximal executions of machines. Plagge and Leuschel in [14] provided a definition of an infinite or finite path $\pi$ of $M$ in terms of a sequence of events and their intermediate states. In order to distinguish notation we use $u$ to represent a trace without the intermediate states. We need not consider the particular states within a trace in our reasoning which is based on infinite traces. When a machine $M$ is deadlock free all of its traces are infinite. We use the functions of concatenation ($\lhd$), projection ($\upharpoonright$) and length ($\#$) on finite and infinite traces.

A more complex behavioural semantics for B machines was given by Schneider et al. in [15] based on the weakest precondition semantics of [13,6] for action systems and CSP. In [15] there are two key results that enable us to reason about infinite sequences of convergent and ordinary events in this paper. Firstly, the following predicate captures that if an infinite trace $u$ performs infinitely many events from $C$ then it has infinitely many events from $O$, where $C$ and $O$ are sets of events.

**Definition 1.** $CA(C, O)(u) \equiv (\#(u \upharpoonright C) = \infty \Rightarrow \#(u \upharpoonright O) = \infty)$

$C$ and $O$ will be used to capture convergent and ordinary events through a development. For an Event-B machine $M$ the above means that it does not diverge on its $C$ events. This is precisely what we get when we prove $WFD\_REF$ but the above definition describes the result on traces.

The second result from [15], restated as Theorem 1, allows us to conclude that there are no infinite sequences of convergent events in the final machine of a refinement chain $M_n$. The function $g_{1,n}$ defines a compositional mapping for all concrete events to abstract events in terms of a function mapping $f$ at each refinement level where $f_{i+1}: \alpha M_{i+1} \Rightarrow \alpha M_i$ and $f_{i+1}(evt_{i+1}) = evt_i \Leftrightarrow evt_{i+1}$ refines $evt_i$. (Note that $g_{1,0}$ is the identity function.)

**Definition 2.** $g_{i,j} = f_j; f_{j-1}; \ldots; f_i$

**Theorem 1.** If $M_0 \preceq_W M_1 \preceq_W \ldots \preceq_W M_n$ then

$$M_n \sat CA(g_{1,n}^{-1}(C_0) \cup \ldots \cup g_{1,n}^{-1}(C_i) \cup \ldots \cup C_n, g_{1,n}^{-1}(O_0))$$

The result for our example is simply $VM_4 \sat CA(C_{alt}, O_0)$ since there is no renaming: each function mapping $f_i$ is the identity.

3 Example

In Section 2.3 we introduced a development strategy for a vending machine. Figures 2, 3, 4 and 5 illustrate a development chain from vending machine $VM_1$, $VM_2$, $VM_3$ to $VM_4$; there are no anticipated events in $VM_4$. Note the numbers
machine $VM_1$
variables $chosen$

invariant $chosen \subseteq \{\text{choc}, \text{biscuit}\}$

events

\begin{align*}
\text{init} \equiv & \ \text{chosen} := \{\} \\
\text{selectBiscuit} \equiv & \ \text{status} : \text{ordinary} \\
\text{when} \ & \text{biscuit} \notin \text{chosen} \ \text{then} \ \text{chosen} := \text{chosen} \cup \{\text{biscuit}\} \ \text{end} \\
\text{selectChoc} \equiv & \ \text{status} : \text{ordinary} \\
\text{when} \ & \text{choc} \notin \text{chosen} \ \text{then} \ \text{chosen} := \text{chosen} \cup \{\text{choc}\} \ \text{end} \\
\text{dispenseBiscuit} \equiv & \ \text{status} : \text{ordinary} \\
\text{when} \ & \text{biscuit} \in \text{chosen} \ \text{then} \ \text{chosen} := \text{chosen} - \{\text{biscuit}\} \ \text{end} \\
\text{dispenseChoc} \equiv & \ \text{status} : \text{ordinary} \\
\text{when} \ & \text{choc} \in \text{chosen} \ \text{then} \ \text{chosen} := \text{chosen} - \{\text{choc}\} \ \text{end} \\
\end{align*}

end

Fig. 2: $VM_1$

of the vending machines start from one. We introduce $VM_0$ in Section 6. Thus $M_0$ in Theorem 1 corresponds to $VM_1$ etc.

$VM_1$ is a simple machine that supports the selection and dispensing of chocolates and biscuits via four events: $\text{selectBiscuit}$, $\text{selectChoc}$, $\text{dispenseBiscuit}$ and $\text{dispenseChoc}$. We abbreviate their names in the narrative to $sb$, $sc$, $db$ and $dc$ respectively. The first refinement step introduces $VM_2$ and the notion of paying and refunding. The $\text{pay}$ event in $VM_2$ is always enabled and allows positive credit to be input. The machine allows a biscuit to be chosen if it has not already been chosen and additionally provided a payment has been made; a chocolate selection is similar. Hence the guards of all four of the original events $sb$, $sc$, $db$ and $dc$ are strengthened. The guard of the $\text{refund}$ event means that credit cannot be refunded for selected items and cannot occur forever since it is convergent. Importantly, the $\text{refundEnabled}$ flag is introduced so that it is only true after a dispense and prevents infinite loops of the $\text{pay}$ followed by $\text{refund}$.

$VM_3$ introduces the notion of stocked items and a new $\text{refill}$ event. We could have chosen many different guards for the $\text{refill}$ event. For example, we could have labelled it $\text{anticipated}$ with a guard of $\text{true}$. Instead we have made an underspecification where the stock can be restocked when there may be no biscuits or no chocolates, and established convergence. Again the guard of the four original events have been strengthened so that they are only enabled when the appropriate stocked item is in stock. But now $db$ and $dc$ also capture the non-deterministic notion of running out or not of their respective items. The guard of $\text{refund}$ remains unchanged. The guard of $\text{pay}$ has been strengthened so that it is only enabled when there is stock but this is not strong enough to prevent it happening infinitely often, hence it remains anticipated in $VM_4$.

The final machine, $VM_4$, is a straightforward data refinement which introduces the capacity of the machine. Apart from highlighting the refinement rela-
Fig. 3: VM$_2$

In this paper we use the grammar for the LTL operators presented by Plagge and Leuschel [14]:

\[
\phi ::= true \mid [x] \mid \neg \phi \mid \phi_1 \lor \phi_2 \mid \phi_1 \ U \phi_2
\]
A machine $M$ satisfies $\phi$, denoted $M \models \phi$, if all traces of $M$ satisfy $\phi$. The definition for $u$ to satisfy $\phi$ is defined by induction over $\phi$ as follows:

$$
\begin{align*}
&u \models \text{true} \\
&u \models [x] \iff u = \langle x \rangle \triangleleft u^1 \\
&u \models \neg \phi \iff \text{it is not the case that } u \models \phi \\
&u \models \phi_1 \lor \phi_2 \iff u \models \phi_1 \text{ or } u \models \phi_2 \\
&u \models \phi_1 U \phi_2 \iff \exists k \geq 0. \forall i < k. u^i \models \phi_1 \text{ and } u^k \models \phi_2 \\
\end{align*}
$$

where $u^n$ is $u$ with the first $n$ elements removed, i.e., $u = (x_0, \ldots, x_{n-1}) \setminus u^n$.

From these operators Plagge and Leuschel derived several additional operators, including: conjunction ($\phi_1 \land \phi_2$), finally (or eventually) ($F \phi$), and globally (or always) ($G \phi$), in the usual way; we also use these operators, and for explic-
machine \textit{VM}_4

constants capacity

properties capacity > 0

variables credit, chosen, refundEnabled, chocStock, biscuitStock

invariant credit \leq capacity \land chosen \subseteq \{choc, biscuit\} \land 
\text{refundEnabled} \in \text{BOOL} \land chocStock \leq capacity \land biscuitStock \leq capacity \land 
(choc \notin \text{stocked} \Rightarrow \text{chocStock} = 0) \land (choc \in \text{stocked} \Rightarrow \text{chocStock} \geq 0) \land 
(biscuit \notin \text{stocked} \Rightarrow \text{biscuitStock} = 0) \land (biscuit \in \text{stocked} \Rightarrow \text{biscuitStock} \geq 0)

variant max\{(\text{chocStock} + \text{biscuitStock}) - credit, 0\}

events

init \equiv \ldots \parallel \text{chocStock} := \text{capacity} \parallel \text{biscuitStock} := \text{capacity}

pay \equiv \text{status} : \text{convergent}

any x where x \in \mathbb{N}_1 \land (\text{chocStock} + \text{biscuitStock}) > \text{credit}

then credit := credit + x end \parallel \text{refundEnabled} := \text{FALSE} end

selectChoc \equiv \text{status} : \text{ordinary}

when \ldots \land \text{chocStock} > 0

then chosen := chosen \cup \{\text{choc}\} end

selectBiscuit \equiv \text{status} : \text{ordinary}

when \ldots \land \text{biscuitStock} > 0

then chosen := chosen \cup \{\text{biscuit}\} end

dispenseBiscuit \equiv \text{status} : \text{ordinary}

when \text{credit} > 0 \land \text{biscuit} \in \text{chosen} \land \text{biscuitStock} > 0

then \ldots end \parallel \text{chocStock} := \text{chocStock} - 1 end

dispenseChoc \equiv \text{status} : \text{ordinary}

when \text{credit} > 0 \land \text{choc} \in \text{chosen} \land \text{chocStock} > 0

then \ldots end \parallel \text{chocStock} := \text{chocStock} - 1 end

refund \equiv \text{status} : \text{ordinary}

refill \equiv \text{status} : \text{ordinary}

when \text{chocStock} = 0 \land \text{biscuitStock} = 0

then \text{chocStock} := \text{capacity} \parallel \text{biscuitStock} := \text{capacity} end

end

Fig. 5: VM\textsubscript{4}

Fitness we also provide direct definitions for them:

\[
\begin{align*}
\models u \models \phi_1 \land \phi_2 & \iff \models u \models \phi_1 \text{ and } \models u \models \phi_2 \\
\models F\phi & \iff \exists i \geq 0. u^i \models \phi \\
\models G\phi & \iff \forall i \geq 0. u^i \models \phi
\end{align*}
\]

We omit atomic propositions on states since our traces are only dealing with events and not paths of states and transitions. We also omit the next operator, see Section 7. In this paper our running example uses globally, finally, or and implies.

For example, the informal specification for the \textit{Lift} given in Section 1 that whenever \textit{top} happens then eventually \textit{ground} will happen, could be written as

\[G([\text{top}] \Rightarrow F[\text{ground}])\]
From our running VM example, the predicate $GF[selectBiscuit]$ expresses that $selectBiscuit$ occurs infinitely often: at any point there is always some occurrence of $selectBiscuit$ at some point in the future. We use this construction in the VM properties introduced in Section 5. For example, we have $\phi_2$ given as

$$\phi_2 = (\neg GF[selectBiscuit]) \Rightarrow G([selectChoc] \Rightarrow F[dispenseChoc])$$

This states that provided $selectBiscuit$ only occurs finitely often (i.e. eventually stops), then whenever $selectChoc$ occurs then $dispenseChoc$ will eventually occur.

It will also be useful to identify the events mentioned explicitly in an LTL formula $\phi$. This set is called the alphabet of $\phi$. This is written $\alpha(\phi)$, similar to the use of $\alpha_M$ for the alphabet of machine $M$. For LTL formulae it is defined inductively as follows:

**Definition 3.**

- $\alpha(true) = \{\}$
- $\alpha([x]) = \{x\}$
- $\alpha(\neg \phi) = \alpha(\phi)$
- $\alpha(\phi_1 \lor \phi_2) = \alpha(\phi_1) \cup \alpha(\phi_2)$
- $\alpha(\phi_1 \land \phi_2) = \alpha(\phi_1) \cup \alpha(\phi_2)$
- $\alpha(\phi_1 U \phi_2) = \alpha(\phi_1) \cup \alpha(\phi_2)$
- $\alpha(F\phi) = \alpha(\phi)$
- $\alpha(G\phi) = \alpha(\phi)$

For example, we have $\alpha(\phi_2) = \{selectBiscuit, selectChoc, dispenseChoc\}$ for $\phi_2$ above.

## 5 Preserving LTL properties

In this section we provide results to demonstrate when properties are preserved by refinement chains. Firstly, we consider chains which do not contain any renaming/splitting of events in a machine. Hence, each function mapping $f_i$ for $M_i \ldots M_n$ is the identity. The first result is a general result identifying a particular temporal property that will always hold for all refinement chains which abide by the rules of the strategy presented in Section 2.3. The second result given in Lemma 2 concerns the preservation of temporal properties that would be proposed by a specifier. We have already observed from the vending machine example that new events can be introduced during a refinement, e.g., pay, refill, etc. We aim for such properties to hold even though new anticipated and convergent events are being introduced.

Lemma 1 states that $M_n$ at the end of the refinement chain will always eventually perform one of the events of the initial machine $M_0$. In other words, $M_n$ will perform infinitely many of the initial events. This means that the events introduced along the refinement chain cannot occur forever at the expense of the original events. In our example, $\alpha M_0 = O_0$. 
Lemma 1. If \( M_0 \preceq_W M_1 \preceq_W \ldots \preceq_W M_n \) and \( M_n \) is deadlock free and \( M_n \) does not contain any anticipated events then \( M_n \models GF( \bigvee_{e \in \alpha M_0} [e] ) \).

Next we provide a definition which is used in Lemma 2 below and it enables us to gain insights into the kinds of temporal properties that are appropriate to be proposed and have the potential of being preserved through a refinement chain. Definition 4 describes a maximal execution satisfying a property \( \phi \). The execution may include some events which do not have an impact on whether the property holds or not therefore we can restrict the maximal execution to include only those events that impact on the property.

Definition 4. Let \( \beta \) be a set of events. Then \( \phi \) is \( \beta \)-dependent if \( \alpha(\phi) \subseteq \beta \) and \( u \models \phi \Leftrightarrow (u \restriction \beta) \models \phi \).

An example of a \( \beta \)-dependent property is \( GF(\text{pay}) \) where \( \beta = \{\text{pay}\} \). If \( u \models GF(\text{pay}) \) then \( u \restriction \{\text{pay}\} \models GF(\text{pay}) \), and vice versa. Conversely, \( \neg G(\text{pay}) \) is not \( \{\text{pay}\}\)-dependent. For example, if \( u = \{\text{pay, refill, pay, pay, \ldots}\} \) then \( u \models \neg G(\text{pay}) \) but \( u \restriction \{\text{pay}\} \not\models \neg G(\text{pay}) \).

As another example, define \( \beta = \{\text{sb, sc, db, dc}\} \). Then \( G(\text{sb} \lor \text{sc} \lor \text{db} \lor \text{dc}) \) is not \( \beta \)-dependent. This is exemplified by any trace \( u \) which contains events other than those in \( \beta \). In this case \( u \restriction \{\text{sb, sc, db, sc}\} \models G(\text{sb} \lor \text{sc} \lor \text{db} \lor \text{dc}) \) but \( u \not\models G(\text{sb} \lor \text{sc} \lor \text{db} \lor \text{dc}) \). \( VM_1 \) exhibits such traces. Observe that this property holds for \( VM_1 \) but not for \( VM_2 \): it is not preserved by refinement. Since it is not \( \beta \)-dependent Lemma 2 below is not applicable for this property.

Our main result for this section identifies conditions under which an LTL property \( \phi \) will be preserved in a refinement chain. The conditions are as follows:

- by the end of the refinement chain there should be no outstanding anticipated events (and so all newly introduced events have been shown to be convergent), as given by restriction 6 of the Development Strategy of Section 2.3;
- the final machine in the refinement chain must be deadlock-free; and
- all of the events that have an effect on whether or not \( \phi \) is true are already present in \( M_i \) (\( \phi \) is \( \beta \)-dependent for some \( \beta \subseteq \alpha M_i \)).

These conditions are enough to ensure that \( \phi \) is preserved through refinement chains. This means that \( M_i \) can be checked for \( \phi \), and we can be sure that the resulting system \( M_n \) will also satisfy it.

The lemma is formally expressed as follows:

Lemma 2. If \( M_i \models \phi \) and \( M_i \preceq_W \ldots \preceq_W M_n \) and \( 0 \leq i < n \) and \( M_n \) is deadlock free and \( M_n \) does not contain any anticipated events and \( \phi \) is \( \beta \)-dependent and \( \beta \subseteq \alpha M_i \) then \( M_n \models \phi \).

5.1 Preserving Vending Machine properties

We consider the application of the above Lemmas to our running example on the refinement chain

\[ VM_1 \preceq_W VM_2 \preceq_W VM_3 \preceq_W VM_4 \]
In this case we obtain immediately from Lemma 1 that

$$VM_4 \models GF([selectBiscuit] \lor [selectChoc] \lor [dispenseBiscuit] \lor [dispenseChoc])$$

Any execution of $VM_4$ will involve infinitely many occurrences of some of these events. The newly introduced events $pay$, $refund$, $refill$ cannot be performed forever without the occurrence of the original events.

We consider some further properties to illustrate the applicability of Lemma 2. Taking $VM_1$ to be the first machine in the refinement chain, we can consider the following temporal properties $\phi$ for $VM_1$:

$$\phi_1 = G([selectChoc] \lor [selectBiscuit] \Rightarrow F([dispenseChoc] \lor [dispenseBiscuit]))$$

$$\phi_2 = (\neg GF[selectBiscuit]) \Rightarrow G([selectChoc] \Rightarrow F[dispenseChoc])$$

$$\phi_3 = (\neg GF[selectChoc]) \Rightarrow G([selectBiscuit] \Rightarrow F[dispenseBiscuit])$$

$$\phi_4 = G([selectChoc] \Rightarrow F[dispenseChoc])$$

$$\phi_5 = G([selectBiscuit] \Rightarrow F[dispenseBiscuit])$$

We note that each of the properties are $\beta$-dependent. Next we consider whether $VM_1 \models \phi_i$ for each $i \in 1..5$. Note that in fact $VM_1 \not\models \phi_4$ and $VM_1 \not\models \phi_5$ since there is a trace for which the properties fail, e.g., in the case of $\phi_4$ the sequence $\langle sc, sb, db, sb, db, \ldots \rangle$ we could have an infinite loop of $sb, db$ events and never reach a $dc$ event. Thus Lemma 2 is not applicable to these properties.

The properties $\phi_2$ and $\phi_3$ are the strongest; $\phi_2$ states that if you do not always have an $sb$ then you will be able to choose a chocolate and for it to be dispensed, and the dual applies in $\phi_3$. Once we have also established the refinement chain $VM_1 \preceq_W VM_2 \preceq_W VM_3 \preceq_W VM_4$, and that $VM_4$ is deadlock free we can deduce using Lemma 2 that $VM_4 \models \phi_i$ for all $i \in 1..3$. Observe however that Lemma 2 does not establish that $\phi_i$ holds in all refinement machines, only those with no anticipated events. For example, $VM_2$ and $VM_3$ do not satisfy $\phi_1$, $\phi_2$ nor $\phi_3$ since $pay$ is anticipated and can be executed infinitely often.

Since $VM_2$ introduced the event $pay$ we can also introduce new temporal properties that are required to hold from $VM_2$ onwards. In other words, we apply Lemma 2 on the chain $VM_2 \preceq_W VM_3 \preceq_W VM_4$. The properties to consider are:

$$\phi_6 = G([pay] \Rightarrow F([dispenseBiscuit] \lor [dispenseChoc]))$$

$$\phi_7 = GF[pay]$$

The infinite behaviour of $pay$ means that $\phi_6$ is not satisfied in $VM_2$. However, $VM_2 \models \phi_7$ thus we can again apply Lemma 2 and obtain that $VM_4 \models \phi_7$ since $\phi_7$ is $\beta$-dependent. This exemplifies that new temporal properties can be added to the refinement verification chain.

We note that in fact $VM_4 \models \phi_6$. Thus $\phi_6$ and $\phi_7$ together imply that $GF([dispenseBiscuit] \lor [dispenseChoc])$ holds for $VM_4$. 


6 Extending preserving LTL properties to handle splitting events

In this section we generalise the results of Section 5 in order to deal with splitting events in Event-B, which occurs when abstract events are refined by several events in the concrete machine, corresponding to a set of alternatives. Consider as a motivating example $VM_0$ in Figure 6. This is refined by $VM_1$, with linking invariant $item = \text{card}(\text{chosen})$, $\text{selectItem}$ refined by both $\text{selectBiscuit}$ and $\text{selectChoc}$, and $\text{dispenseItem}$ refined by both $\text{dispenseBiscuit}$ and $\text{dispenseChoc}$. Splitting events also involves their renaming to allow for several concrete events to map to the same abstract one. A refinement step will therefore be associated with a renaming function $h$ from concrete events to the abstract events that they refine. In the general case $h$ will be many-to-one, since many concrete events may map to a single abstract event; and it will also be partial, since new events in the concrete machine will not map to any abstract event.

In general, each step in a refinement chain $M_0 \preceq W M_1 \preceq W \ldots \preceq W M_n$ will have an event renaming function $h_i$ corresponding to the renaming and splitting step from $M_i$ to $M_{i-1}$. We define $g_{i,n}$ to be the composition of these renaming function from $h_n$ down to $h_i$. Observe that $g_{i,n}$ will be undefined on any event that does not map to $M_{i-1}$, in other words any event that corresponds to an event introduced at some point in the refinement chain. For example, for the chain $VM_0 \preceq W VM_1 \preceq W \ldots \preceq W VM_4$, we obtain that $g_{1,4}(\text{selectBiscuit}) = g_{1,4}(\text{selectChoc}) = \text{selectItem}$, and $g_{1,4}(\text{dispenseBiscuit}) = g_{1,4}(\text{dispenseChoc}) = \text{dispenseItem}$, and $g_{1,4}$ is not defined on the remaining events of $VM_4$.

Lemma 3 generalises to state that the final machine in the refinement chain must always eventually perform some event relating to an event in the initial machine.

**Lemma 3.** If $M_0 \preceq W M_1 \preceq W \ldots \preceq W M_n$ and $M_n$ is deadlock free and $M_n$ does not contain any anticipated events then $M_n \models GF(\bigvee_{e \in g_{i,n}^{-1}(\alpha M_n)} e)$

Observe that if there is no renaming or splitting, then $g_{1,n}$ is the identity function on the events in $\alpha M_0$, yielding Lemma 1.

We are interested in how the LTL properties of an abstract machine becomes transformed through a refinement step such as $VM_0$ to $VM_1$. For example, the property $GF[\text{selectItem}]$ for $VM_0$ states that from any stage that is reached, $\text{selectItem}$ will eventually occur. This will translate to the property $GF([\text{selectBiscuit}] \lor [\text{selectChoc}])$ for $VM_1$. We now consider how LTL properties translate through a renaming function $h$.

For a given event renaming function $h$, we define $\text{trans}_h$ as the translation that maps LTL formulae by mapping abstract events to the disjunction of their corresponding concrete events, as follows:
Definition 5.

\[ \text{trans}_h(\text{true}) = \text{true} \]
\[ \text{trans}_h([x]) = \bigvee_{y : h(y) = x} [y] \]
\[ \text{trans}_h(\neg \phi) = \neg \text{trans}_h(\phi) \]
\[ \text{trans}_h(\phi_1 \lor \phi_2) = \text{trans}_h(\phi_1) \lor \text{trans}_h(\phi_2) \]
\[ \text{trans}_h(\phi_1 \land \phi_2) = \text{trans}_h(\phi_1) \land \text{trans}_h(\phi_2) \]
\[ \text{trans}_h(\phi_1 U \phi_2) = \text{trans}_h(\phi_1) U \text{trans}_h(\phi_2) \]
\[ \text{trans}_h(G \phi) = G \text{trans}_h(\phi) \]
\[ \text{trans}_h(F \phi) = F \text{trans}_h(\phi) \]

For example

\[ \text{trans}_h(G([\text{selectItem}] \Rightarrow F[\text{dispenseItem}])) \]
\[ = G(([\text{selectBiscuit}] \lor [\text{selectChoc}]) \Rightarrow F([\text{dispenseBiscuit}] \lor [\text{dispenseChoc}]))) \]

Lemma 2 generalises to Lemma 4 below, to state that LTL properties are carried along the refinement chain by translating them. In particular, if a property \( \phi \) is established for \( M_{i-1} \), then \( \text{trans}_{g_i,n}(\phi) \) will hold for \( M_n \):

**Lemma 4.** If \( M_{i-1} \models \phi \), and \( M_{i-1} \preceq \ldots \preceq \preceq M_n \), and \( 0 \leq i - 1 < n \), \( M_n \) is deadlock free and \( M_n \) does not contain any anticipated events and \( \phi \) is \( \beta \)-dependent and \( \beta \subseteq \alpha M_{i-1} \) then \( M_n \models \text{trans}_{g_i,n}(\phi) \)

For example, from the result for \( VM_0 \) that whenever \( \text{selectItem} \) occurs then \( \text{dispenseItem} \) will eventually occur,

\[ VM_0 \models G([\text{selectItem}] \Rightarrow F[\text{dispenseItem}])) \]
we obtain from Lemma 4 that

\[ VM_4 \models G( ([selectBiscuit] \lor [selectChoc]) \Rightarrow F([dispenseBiscuit] \lor [dispenseChoc])) \]

This states that whenever \( selectBiscuit \) or \( selectChoc \) occur, then \( dispenseBiscuit \) or \( dispenseChoc \) will eventually occur.

7 Discussion and related work

One of the few papers to discuss LTL preservation in Event-B refinement is Groslambert [8]. The LTL properties were defined in terms of predicates on system state rather than our paper’s formulation in terms of the occurrence of events. His paper focused only on the introduction of new convergent events. It did not include a treatment of anticipated events but this is unsurprising since the paper was published before their inclusion in Event-B. Our results are more general in two ways. Firstly, the results support the treatment of anticipated events. Secondly, we allow more flexibility in the development methodology. A condition of Groslambert’s results was that all the machines in the refinement chain needed to be deadlock free. The two main lemmas in our paper: Lemmas 2 and 4 do not require each machine in a refinement chain to be deadlock free, only the final machine. It is irrelevant if intermediate \( M_i \)’s deadlock as long as the deadlock is eventually refined away.

Groslambert deals with new events via stuttering and leaves them as visible events in a trace. This is why the LTL operators used by the author do not include the next operator \( (X) \). As new events may happen this may violate the \( X \) property to be checked. Plagge and Leuschel in [14] permit the use of the \( X \) operator since they treat the inclusion of new events as internal events which are not visible. Since we deal with new events as visible events we also lose the ability to reason about a temporal property using the typical \( X \) operator. Our reasoning is simpler than both Groslambert and Plagge and Leuschel since we only focus on events but this means we cannot have atomic propositions in our LTL, whereas they can.

The notion of verification of temporal properties of both classical and Event-B systems using proof obligations has been considered in many research papers. Abrial and Musat in an early paper, [3], introduced proof obligations to deal with dynamic constraints in classical B. In a more recent paper [10] Hoang and Abrial have also proposed new proof obligations for dealing with liveness properties in Event-B. They focus on three classes of properties: existence, progress and persistence, with a view to implementing them in Rodin. Bicarregui et al. in [5] introduced a temporal concept into events using the guard in the \texttt{when} clause and the additional labels of \texttt{within} and \texttt{next} so that the enabling conditions are captured clearly and separately. However, these concepts are not aligned with the standard Event-B labelling.

The interest of LTL preservation through refinement is wider than simply Event-B. Derrick and Smith [7] discuss the preservation of LTL properties in the
context of Z refinement but the authors extend their results to other logics such as CTL and the µ calculus. They focus on discussing the restrictions that are needed on temporal-logic properties and retrieve relations to enable the model checking of such properties. Their refinements are restricted to data refinement and do not permit the introduction of new events in the refinement steps. Our paper does permit new events to be introduced during refinement steps; the contribution is in identifying conditions for LTL properties to hold even in the context of such new events.

8 Conclusions and future work

The paper has provided foundational results that justify when temporal properties hold at the end of an Event-B refinement chain for developments which contain anticipated, convergent and ordinary events, which goes beyond that presented in [8]. The paper has also provided restrictions on the temporal properties in terms of being β-dependent which help to determine when a temporal property of interest should be introduced into the development chain.

We could extend the results to deal with merging events. The inclusion of the X LTL operator and availability will require use to look at execution paths which include state transitions (π paths). The inclusion of availability will enable us to address more advanced and useful notions of fairness in the context of temporal properties. Our notion of weak fairness will be akin to that described in Barradas and Bert in [4]. It will draw on work by Williams et al. [16]. We could also consider the impact on temporal property preservation in refinement chains which do not achieve convergence of all its new events by the end.

In ongoing work we are looking at event liveness via the proof obligation for strong deadlock freedom $S_{NDF}$. We have defined new labelling of events to so that liveness proofs are on particular events. This is analogous to proving $WFD\_REF$ for events that are labelled anticipated or convergent. We have recently defined the semantics of Event-B in terms of stable failures and detailed its relationship with $S_{NDF}$. We are currently combining these results with our work in [15] in order to provide a cohesive process algebra underpinning for Event-B.

Acknowledgments. Thanks to Thai Son Hoang and Thierry Lecomte for discussions about Event-B development strategies and the challenges of discharging liveness proofs. Thanks to Steve Wesemeyer for discussions on the example. Thanks to the reviewers for their constructive comments that helped to improve the paper.

References

1. J.-R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010.
2. J.-R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. STTT, 12(6):447–466, 2010.
A Proofs of Lemmas

In this appendix we provide the proofs for the main lemmas: Lemma 1, 2 and Lemma 3. We also provide additional supporting definitions and lemmas.

**Proof of Lemma 1** Since $M_n$ does not deadlock consider an infinite trace $u$ of $M_n$. Let $u = u | (C_1 \cup \ldots \cup C_n \cup O_0)$ since $M_n$ does not have any anticipated events. We aim to prove that $u | O_0$ is infinite. If $u | C_1 \cup \ldots \cup C_n$ is finite then $u | O_0$ is infinite. If $u | C_1 \cup \ldots \cup C_n$ is infinite then $u | O_0$ is infinite by Theorem 1 where $g_{1,n}^{-1}$ is simply the identity function.

**Proof of Lemma 2** Let $u$ be a trace of $M_n$ and since $M_n$ is deadlock free $u$ will be infinite. Since $M_i \preceq_M \ldots \preceq_M M_n$ then we deduce that $u \upharpoonright \alpha(M_i)$ is a trace of $M_i$. Therefore $u \upharpoonright \alpha(M_i) \models \phi$. Therefore, $u \upharpoonright \alpha(M_i) \models \beta \models \phi$, i.e., $u \upharpoonright \beta \models \phi$, since $\beta \subseteq \alpha(M_i)$. By Definition 1 it follows that $u \models \phi$ as required.
The following definition and lemmas regarding infinite traces and how they relate to temporal properties in the context of translation mappings: Definition 6, Lemma 5, Lemma 6 and Lemma 7 are required in order to prove Lemma 4 presented in Section 6 and proved below.

Given a partial surjective function $h : B \rightarrow A$, we define the completion of $h$ as follows:

**Definition 6.** Given a partial surjection $h : B \rightarrow A$, its completion $h_{\text{tot}} : B \rightarrow (A \cup (B - \text{dom}(h)))$ is the function $h$ augmented with the identity on all events not in the domain of $h$. $h_{\text{tot}}$ is defined as $h_{\text{tot}} = h \cup \text{id}(B - \text{dom}(h))$.

The $f_{i,1} \leq i \leq n$, for each machine $M_i \ldots M_n$ are examples of $h_{\text{tot}}$. $f_{2,\text{tot}}$ for $VM_2$ is \{ $(\text{pay}, \text{pay})$, $(\text{refund}, \text{refund})$, $(\text{sb}, \text{selectItem})$, $(\text{sc}, \text{selectItem})$, \ldots \}. The function $h$ lifts to $B^\omega \rightarrow A^\omega$ by applying $h$ pointwise to each event. $h_{\text{tot}}$ similarly lifts to $B^\omega \rightarrow (A \cup (B - \text{dom}(h)))^\omega$.

**Lemma 5.** If $\alpha(\phi) \subseteq \text{ran}(h)$ (i.e. all events mentioned in $\phi$ are in the range of $h$) then $\text{trans}_{h}(\phi)$ is the same as $\text{trans}_{h_{\text{tot}}}(\phi)$.

**Proof** This is because $h^{-1}$ and $h_{\text{tot}}^{-1}$ are the same for all events in $\text{ran}(h)$. \qed

The following lemma provides a result that allows us to demonstrate the relationship between an infinite trace $u$ preserving a transformed temporal property and a corresponding transformation on the trace preserving a temporal property.

**Lemma 6.** If $\alpha(\phi) \subseteq \text{ran}(h)$, then $u \in (\text{dom}(h))^* \Rightarrow (u \models \text{trans}_{h}(\phi) \iff h(u) \models \phi)$

**Proof:** By structural induction over the structure of $\phi$.

**Base Case:** Consider when $\phi$ is $[x]$. If $u \models \text{trans}_{h}[x]$

$\Leftrightarrow u \models \bigvee_{y|h(y)=x} y$ [by Definition 5]

$\Leftrightarrow \bigvee_{y|h(y)=x} u \models [y]$ [by model distribution]

$\Leftrightarrow \bigvee_{y|h(y)=x} u_0 = y$ [where $u_0$ is first element of $u$]

$\Leftrightarrow \bigvee_{y|h(y)=x} (h(u)_0 = h(y) = x$ [condition on $y$]

$\Leftrightarrow (h(u))_0 = x \Leftrightarrow h(u) \models [x]$

**Inductive Cases:** Consider where $\phi$ is $\neg \phi$. Then $u \models \text{trans}_{h}(\phi) \Leftrightarrow u \not\models \text{trans}_{h}(\phi) \Leftrightarrow h(u) \not\models \phi \Leftrightarrow h(u) \models \neg \phi$.

Proof of the other cases follow similarly. \qed

The next lemma allows us to demonstrate the relationship between a property $\phi$ which is $\beta$-dependent following transformation.
Lemma 7. If $\alpha(\phi) \subseteq \beta \subseteq \text{ran}(h)$, then if $\phi$ is $\beta$-dependent then $\text{trans}_h(\phi)$ is $h^{-1}(\beta)$-dependent.

Proof: Consider an infinite trace $v$. Then

$v \models \text{trans}_h(\phi) \\
\Leftrightarrow v \models \text{trans}_\text{tot}(\phi) \quad \text{[by Lemma 5 since $\alpha(\phi) \subseteq \beta \subseteq \text{ran}(h)$]} \\
\Leftrightarrow h_{\text{tot}}(v) \models \phi \quad \text{[by Lemma 6]} \\
\Leftrightarrow h_{\text{tot}}(v \upharpoonright h^{-1}(\beta)) \models \phi \quad \text{[by function mapping]} \\
\Leftrightarrow v \upharpoonright h^{-1}(\beta) \models \text{trans}_\text{tot}(\phi) \quad \text{[by Lemma 6]} \\
\Leftrightarrow v \upharpoonright h^{-1}(\beta) \models \text{trans}_h(\phi) \quad \text{[since $\alpha(\phi) \subseteq A \text{and } \beta \subseteq A$, so $h$ and $h_{\text{tot}}$ have the same effect]}$

$\blacksquare$

Proof of Lemma 4: Let $u$ to be an infinite execution of $M_n$ since $M_n$ is deadlock free. Let $h = g_{i,n}$. It follows from Theorem 6.5 of [15] that $h_{\text{tot}}(u) \upharpoonright \alpha(M_i)$ is an infinite execution of $M_i$. Then we obtain

$h_{\text{tot}}(u) \upharpoonright \alpha(M_i) \models \phi \quad \text{[since $M_i \models \phi$]} \\
\Rightarrow h_{\text{tot}}(u) \upharpoonright \alpha(M_i) \upharpoonright \beta \models \phi \quad \text{[since $\phi$ is $\beta$-dependent]} \\
\Rightarrow h_{\text{tot}}(u) \upharpoonright \beta \models \phi \quad \text{[by Definition 4]} \\
\Rightarrow h_{\text{tot}}(u \upharpoonright h^{-1}(\beta)) \models \phi \quad \text{[interaction of projection and mapping]} \\
\Rightarrow u \upharpoonright h^{-1}(\beta) \models \text{trans}_{\text{tot}}(\phi) \quad \text{[by Lemma 7]} \\
\Rightarrow u \upharpoonright h^{-1}(\beta) \models \text{trans}_h(\phi) \quad \text{[by Lemma 5]} \\
\Rightarrow u \models \text{trans}_h(\phi) \quad \text{[by Lemma 7]}$

as required. $\blacksquare$