The Behavioural Semantics of Event-B Refinement

Steve Schneider[1], Helen Treharne[1], Heike Wehrheim[2]
[1]{S.Schneider,H.Treharne}@surrey.ac.uk,
Department of Computing, University of Surrey

[2]wehrheim@uni-paderborn.de
Department of Computer Science, University of Paderborn

Abstract. Event-B provides a flexible framework for stepwise system development via refinement. The framework supports steps for (a) refining events (one-by-one), (b) splitting events (one-by-many), and (c) introducing new events. In each of the steps events can be indicated as *convergent* (to be made internal) or *anticipated* (treatment deferred to a later refinement step). All such steps are accompanied with precise proof obligations. However, no behavioural semantics has been provided to validate the proof obligations, and no formal justification has previously been given for the application of these rules in a refinement chain. Behavioural semantics expresses a clear relationship between the first and last machines in a refinement chain. The framework we present provides a coherent justification for Abrial's approach to refinement in Event-B, and its generalisation to interface extension: adding events to the interface. In this paper, we give a behavioural semantics for Event-B refinement, with a treatment for the first time of splitting events and of anticipated events, adding to the well-understood treatment of convergent events. To this end, we define a CSP semantics for Event-B and show how the different forms of Event-B refinement can be captured as CSP refinement. It turns out that the appropriate CSP refinement relationship is influenced by the particular Event-B *development strategy* taken. We present two such strategies, one allowing, the other disallowing interface extensions.

Keywords: Event-B, CSP, refinement, traces, divergences, development strategy.

1. Introduction

Event-B [Abr10, MAV05] provides a framework for formal system development through stepwise refinement. Refinement allows to gradually build a complex system model by introducing more detail with every step. Individual refinement steps in Event-B are verified with respect to their proof obligations, and the transitivity of refinement ensures that the final system description is a refinement of the initial one. The refinement

Correspondence and offprint requests to: S. Schneider, H. Treharne, H. Wehrheim process allows new events to be introduced through the refinement process, in order to provide the more concrete implementation details necessary as refinement proceeds.

The framework allows for a great deal of flexibility to cover a broad range of system developments. The recent book [Abr10] comprising case studies from rather diverse areas shows that this goal is actually met. The basic modelling entity of Event-B are *machines* and their *events*. The flexibility of Event-B development a result of the different ways of dealing with events during refinement. At each step existing events of an Event-B machine need to be refined. This can be achieved by (a) simply keeping the event as is, (b) refining it into another event, possibly because of a change of the state variables, or (c) splitting it into several events[1]. Furthermore, every refinement step allows for the introduction of new events. To help reasoning about divergence, events are in addition classified as *ordinary*, *anticipated* or *convergent*. New events introduce new detail at the interface of a machine specification, and are introduced as anticipated or convergent. Convergent events are considered internal and therefore must not be executed forever. Their treatment within action systems refinement has been well established [But92]. For anticipated events the decision whether this is to be an internal or external event (i.e., part of the interface) is deferred to later refinement steps, but must be made by the end of the refinement chain. The use of anticipated events is a more recent development, introduced within the context of Event-B [Abr10, ABH$^+$10]. Refinement steps come with precise proof obligations on all kinds of events; appropriate tool support helps in discharging these [BH07, ABHV08, ABH$^+$10]. Event-B is essentially a *state-based* specification technique like Z [WD96] or Alloy [Jac02], and proof obligations therefore reason about predicates on states.

However, no behavioural semantics has been provided to validate the proof obligations, and no formal justification has previously been given for the application of these rules in a refinement chain. In fact, it is hard to find one fixed formal semantics for Event-B at all. Instead, a recent article of Hallerstede [Hal11] advocates the absence of one fixed semantics as an advantage because it increases the flexibility of the specification formalism and allows to employ it in a variety of modelling domains — all with the same type of proof obligations. Nevertheless, even Hallerstede mentions that the design of Event-B has been inspired by action systems [BvW98], and Butler [But12] also implicitly uses an action system, i.e. weakest-precondition semantics for Event-B machines. Here we also follow this view on Event-B and treat events like guarded commands with a weakest-precondition semantics.

Our specific interest however is in refinement, and in particular in investigating the effect of the proof obligations on the various types of events with respect to the semantics. Our behavioural semantics expresses a clear relationship between the machines in a refinement chain, in particular the initial and the final one. The framework we present provides a coherent justification for Abrial's approach to refinement in Event-B, and its generalisation to interface extension: adding events to the interface. Our framework is based on the process algebra CSP which provides us — besides a sound and large body of underlying theory — with a clear notion of refinement and a number of operators for dealing with specific sets of events in refinement (e.g. hiding).

In order to understand the relationship between Event-B and CSP refinement, these two concepts need to be set in a single framework. Both formalisms support a variety of different forms of refinement: Event-B by means of several proof obligations related to refinement, out of which the system designer chooses an appropriate set; CSP by means of its different semantic domains of traces, failures and divergences. The aim of this paper is to give a behavioural semantics for Event-B refinement in terms of CSP's behaviour-oriented process refinement[Hoa85]. This will also provide the underlying results that support refinement in the combined formalism Event-B$\|$CSP [STW10]. It turns out that CSP supports an approach to refinement consistent with that of Event-B. It faithfully reflects all of Event-B's possibilities for refinement, including splitting events and new events. It also deals with the Event-B approach of anticipated events as a means to defer consideration of divergence-freedom. Our results involve support for individual refinement steps as well as additional results for the resulting refinement chain. Our work is thus in line with previous studies relating state-based with behaviour-oriented refinement (see e.g. [BD02, DB03, BD09]).

It turns out that our results closely depend on the *development strategy* chosen for the refinements. Development strategies impose rules on the use of different types of events and their refinements. To the best of our knowledge development strategies for Event-B have not been discussed so far, let alone formalised. Tools supporting development and proof of Event-B designs like Rodin [ABH$^+$10] only give very weak restrictions on the use of event types. Thus another contribution of this paper is the formal definition of

[1] A fourth option is merging of events which we do not consider here.

two different development strategies for Event-B, depending on whether or not to enable interface extension: where the machine interface (akin to an API) is extended to allow additional events. Basically, strategy I disallows interface extensions whereas strategy II allows them. This means that with strategy I the interface of a system is given by the initial machine specification, and all events introduced during refinement steps are eventually hidden. This is the approach taken in [Abr10]. Strategy II is the more general one, allowing for a greater flexibility in development, however, at the price of achieving a weaker relationship between the machines in a refinement. Strategy II is the approach supported by Rodin [EB11]. For both strategies we have identified results about the corresponding CSP refinement. Refinement in Event-B does not require liveness properties to be preserved. We therefore do not consider them in this paper.

The paper is structured as follows. The next section introduces the necessary background on Event-B and CSP, and also introduces the first part of our running example. Section 3 gives the CSP semantics for Event-B based on weakest preconditions. In Section 4 we precisely fix the notion of refinement used in this paper, both for CSP and for Event-B, and further develop our example. Section 5 then sets the two refinement definitions in relation, first of all just for one step in a refinement and independent of the strategy used for chains of refinement. It turns out that the appropriate refinement concept of CSP that underpins Event-B is infinite-traces-divergences refinement. Sections 6 and 7 then present a formal definition of the two strategies I and II and set these in relation with a CSP refinement concept for chains of refinements. The last section concludes.

This paper is an extension of [STW11b] introducing a new refinement strategy, with its associated definitions and new results, and including a larger example used throughout the whole paper for both strategies.

2. Background

We start with a short introduction to Event-B and CSP. For more detailed information see [Abr10] and [Sch99] respectively.

2.1. Event-B

Event-B [Abr10, MAV05] is a state-based specification formalism based on set theory. The basic modelling entity in Event-B are machines with their events. Here we describe the basic elements of an Event-B machine required for this paper; a full description of the formalism can be found in [Abr10].

The main modelling construct in Event-B is a *machine*, which encapsulates state and events which can query or update the state. A machine specification describes state information in terms of state variables and invariants on them. The machine describes how the state is initialised, and also describes guarded *events*, which describe how and when the state may be updated. These are the core elements of Event-B that we are concerned with in this paper.

Event-B machines are structured into a number of *clauses* which are concerned with different aspects of the specification. Here we concentrate on those clauses concerned with state and events. We will therefore describe a machine M with a list of state variables v, a state invariant $I(v)$, and a set of events $ev, \ldots$ to update the state (see left of Figure1). Initialisation is a special event *init* which sets the initial state of the machine, and its guard is *true*.

Event-B also in general allows sets which introduce new types, and constants. However, for our purposes the treatment of elements such as sets and constants are independent of the results of this paper, and so we will not include them here. However, they can be directly incorporated without affecting our results.

A machine M will have various proof obligations on it. These include consistency obligations: that events preserve the invariant. They can also include (optional) deadlock-freeness obligations: that at least one event guard is always true.

Central to an Event-B description is the definition of the events, each consisting of a *guard* $G(v)$ over the variables, and a *body*, usually written as an assignment S on the variables. The body defines a *before-after predicate* $BA(v, v')$ describing changes of variables upon event execution, in terms of the relationship between the variable values before (v) and after (v'). The body can also be written as $v :| BA(v, v')$, whose execution assigns to v any value v' which makes the predicate $BA(v, v')$ true (see right of Fig. 1, where BA is the predicate in event *evt*). In Event-B an event may also introduce local variables, which can be included in

```
machine  $M$                      $ev \mathrel{\widehat{=}}$
variables  $v$                        when
invariant  $I(v)$                         $G(v)$
events  $init, ev, \ldots$                then
end                                       $v :\mid BA(v, v')$
                                      end
```

Fig. 1. Template of an Event-B machine and an event.

the guard (which constrains what values they can take), and in the body where they can have an effect on the change of state. Such events are constructed as:

```
$evt \mathrel{\widehat{=}}$
    any
        $x$
    where
        $G(v, x)$
    then
        $v :\mid BA(v, x, v')$
    end
```

For our purposes, the treatment of local variables is orthogonal to our results, and so we do not consider them here. They can be directly incorporated without affecting our results. We will thus use the form of events presented in Fig. 1.

The Event-B approach to semantics, provided in [Abr10, MAV05], is to associate proof obligations with machines. The key proof obligation on events is that they preserve the invariant: when an event is called within its guard, then the state resulting from executing the body should meet the invariant. For example, in the case of the machine in Fig. 1 (which does not include sets or constants) we obtain the following proof obligation **INV** on events which have the form of *evt*:

$I(v) \wedge G(v) \wedge BA(v, v')$ $\vdash$ $I(v')$	**INV**

This is a particular case of the **INV** rule given in [Abr10, page 189]. Discharging this proof obligation establishes that the event preserves the invariant, and so is well-defined.

There is no definitive presentation of the core proof obligations that any machine must satisfy, but the proof obligations presented in [Abr10, MAV05] provide the key rules. In practice, the proof obligations generated by the Rodin platform [ABH$^+$10] give the *de facto* rules for a machine.

In this paper we will make use of *weakest precondition* semantics for guarded commands, applying them to B events. We introduce this semantics in Section 3. This semantics gives a similar treatment of events as the proof obligations approach of Event-B, but from the point of view of considering an event as defining a relation between before- and after-states. The proof obligations approach is geared towards verification and is directly suited to the tool support.

As an example of an Event-B machine, consider the machine given in Fig. 2 abstractly specifying the (protocol of use of a) basket in a store. The machine represents the initial step of a development. In later sections we will gradually refine this machine, and by this introduce the concept of refinement in Event-B; in particular the different kinds of events occurring during refinement.

The machine defines a basket to be in one of two possible states: either empty or complete. Complete corresponds to the case when the customer has decided to have completed his shopping. The notation for the events is a short version of the above given as the guard for the events is true and the before-after-predicate allows for a single after value only, thus the event is simply written as an assignment. The two events (which are both unguarded and thus can always be executed) do the checkout and empty the basket, respectively. In addition, the machine has an event *init* which is however omitted since it is empty.

machine $Basket0$
variables $state0$
invariant $state0 \in \{state\_empty, state\_complete\}$
events
  init $\widehat{=}$
    $state0 := state\_empty$
  checkout $\widehat{=}$
    $state0 := state\_complete$
  empty $\widehat{=}$
    $state0 := state\_empty$
end

Fig. 2. Event-B machine *Basket*0

2.2. CSP

CSP, Communicating Sequential Processes, introduced by Hoare [Hoa85] is a formal specification language aiming at the description of communicating processes. A process is characterised by the events it can engage in and their ordering. Events will in the following be denoted by $a_1, a_2, \ldots$. The set of all possible events is denoted Σ. Process expressions are built out of events using a number of composition operators. In this paper, we will make use of just three of them: *interleaving* ($P_1 \mid\mid\mid P_2$), executing two processes in parallel without any synchronisation; *hiding* ($P \setminus N$), making a set N of events internal; and *renaming* ($f(P)$ and $f^{-1}(P)$), changing the names of events according to a renaming function f. If f is a non-injective function, $f^{-1}(P)$ will offer a choice of events b such that $f(b) = a$ whenever P offers event a.

Every CSP process P has an alphabet αP. Its semantics is given using the Failures/Divergences/Infinite Traces semantic model for CSP. This is presented as $\mathcal{U}$ in [Ros98] or FDI in [Sch99]. The semantics of a process can be understood in terms of four sets, T, F, D, I, which are respectively the (finite) traces, failures, divergences, and infinite traces of P. These are understood as observations of possible executions of the process P, in terms of the events from αP that it can engage in.

Traces are finite sequences of events from P's alphabet: $tr \in \alpha P^*$. The set $traces(P)$ represents the possible finite sequences of events that P can perform. Since this paper is not addressing liveness, failures will not need to be considered in this paper and are therefore not explained here.

Divergences are finite sequences of events on which the process might diverge: perform an infinite sequence of internal events (such as an infinite loop) at some point during or at the end of the sequence. After a process has diverged, the semantics allows any events from Σ to be possible. The set $divergences(P)$ is the set of all possible divergences for P. Infinite traces $u \in \alpha P^\omega$ are infinite sequences of events. The set $infinites(P)$ is the set of infinite traces that P can exhibit. For technical reasons it also contains those infinite sequences that have some prefix which is a divergence.

Definition 2.1. A process P is *divergence-free* if $divergences(P) = \{\}$.

We use tr to refer to finite traces. These can also be written explicitly as $\langle a_1, a_2, \ldots, a_n \rangle$. The empty trace is $\langle\rangle$, concatenation of traces is written as $tr_1 \frown tr_2$. We use u to refer to infinite traces. Given a set of events A, the *projections* $tr \upharpoonright A$ and $u \upharpoonright A$ are the traces restricted to only those events in A. Note that $u \upharpoonright A$ might be finite, if A events appear only finitely many times in u. Conversely, $tr \setminus A$ and $u \setminus A$ are those traces with the events in A removed. The length operator $\#tr$ and $\#u$ gives the length of the trace it is applied to which can be an integer or infinity. The alphabet operator αtr and αu gives us the alphabet of a trace, i.e., the set of events occuring in it. Given two traces tr_1 and tr_2, $tr_1 \mid\mid\mid tr_2$ defines the set of all traces presenting interleavings of the first and the second trace, i.e., combinations which keeps the ordering of the original traces but arbitrarly mix elements from the first with the second trace. Finally, given a renaming function f on the set of events, $f(tr)$ is the trace in which all elements are renamed according to f. Interleaving and renaming are also defined in a similar way on infinite traces. As an example for all these definitions consider the following:

$\langle a, b\rangle \frown \langle c, b, c\rangle = \langle a, b, c, b, c\rangle$
$\langle a, b, a, c, c, c, b\rangle \upharpoonright \{a, b\} = \langle a, b, a, b\rangle$
$\langle a, b, a, c, c, c, b\rangle \setminus \{a, b\} = \langle c, c, c\rangle$
$\#\langle a, b, a, c, c, c, b\rangle = 7$
$\langle a, c, d, b\rangle \in (\langle a, b\rangle \mid\mid\mid \langle c, d\rangle)$
$\langle a, d, c, b\rangle \notin (\langle a, b\rangle \mid\mid\mid \langle c, d\rangle)$
$f(\langle a, b, b\rangle) = \langle c, d, d\rangle$ for $f : \{a \mapsto c, b \mapsto d\}$

Such operations on traces are, for instance, used to define the semantics of the process composition operators in CSP, like hiding, interleaving and renaming explained above. Since we make use of them later, we will give the formal definitions for the infinite traces and divergences semantics here.

Definition 2.2. Let P be a process, A a set of events, f a renaming function.

$$\begin{aligned}
divergences(P \setminus A) &= \{(tr \setminus A) \frown tr' \mid tr \in divergences(P)\} \\
&\quad \cup \{(u \setminus A) \frown tr' \mid u \in infinites(P) \wedge \#(u \setminus A) < \infty\} \\
infinites(P \setminus A) &= \{u \setminus A \mid u \in infinites(P) \wedge \#(u \setminus A) = \infty\} \\
&\quad \cup \{tr \frown u' \mid tr \in divergences(P \setminus A)\}
\end{aligned}$$

$$\begin{aligned}
divergences(P \mid\mid\mid Q) &= \{tr \frown tr' \mid tr \in (tr_P \mid\mid\mid tr_Q), tr_P \in divergences(P) \wedge tr_Q \in traces(Q)\} \\
&\quad \cup \{tr \frown tr' \mid tr \in (tr_P \mid\mid\mid tr_Q), tr_P \in traces(P) \wedge tr_Q \in divergences(Q)\} \\
infinites(P \mid\mid\mid Q) &= \{u \mid u \in (u_P \mid\mid\mid u_Q), u_P \in infinites(P) \wedge u_Q \in infinites(Q)\} \\
&\quad \cup \{u \mid u \in (u_P \mid\mid\mid tr_Q), u_P \in infinites(P) \wedge tr_Q \in traces(Q)\} \\
&\quad \cup \{u \mid u \in (tr_P \mid\mid\mid u_Q), tr_P \in traces(P) \wedge u_Q \in infinites(Q)\} \\
&\quad \cup \{tr \frown u' \mid tr \in divergences(P \mid\mid\mid Q)\}
\end{aligned}$$

$$\begin{aligned}
divergences(f(P)) &= \{f(tr) \frown tr' \mid tr \in divergences(P)\} \\
infinites(f(P)) &= \{f(u) \mid u \in infinites(P)\} \\
&\quad \cup \{tr \frown u' \mid tr \in divergences(f(P))\}
\end{aligned}$$

$$\begin{aligned}
divergences(f^{-1}(P)) &= \{tr \frown tr' \mid f(tr) \in divergences(P)\} \\
infinites(f^{-1}(P)) &= \{u \mid f(u) \in infinites(P)\} \\
&\quad \cup \{tr \frown u' \mid tr \in divergences(f(P))\}
\end{aligned}$$

In this definition, tr' ranges over finite sequences of events from Σ, and u' ranges over infinite sequences of events from Σ.

As a first observation on infinite traces, hiding and divergences, we get the following:

Lemma 2.3. If P is divergence-free, and for every infinite trace u of P we have $\#(u \setminus A) = \infty$, then $P \setminus A$ is divergence-free.

Proof: Follows immediately from the semantics of the hiding operator. $\square$

Later, we furthermore use *specifications* on traces or, more generally, on CSP processes. Specifications are given in terms of predicates. If *SPEC* is a predicate on a particular semantic element, then we write P **sat** *SPEC* to denote that all relevant elements in the semantics of P meet the predicate *SPEC*. For example, if $SPEC(u)$ is a predicate on infinite traces, then P **sat** $SPEC(u)$ is equivalent to $\forall\, u \in infinites(P) \,.\, SPEC(u)$.

3. CSP semantics for Event-B machines

The development of Event-B, in particular of its notion of refinement, has been largely inspired by action systems [Hal11]. Thus Morgan's CSP semantics for action systems [Mor90] allows traces, failures, and divergences to be defined for Event-B machines, in terms of the sequences of events they can and cannot engage in. Butler's extension to handle unbounded nondeterminism [But92] defines the infinite traces for action systems. These together give a way of considering Event-B machines as CSP processes, and treating them within the CSP semantic framework. Note that the notion of *traces* for machines is different to that presented in [Abr10], where traces are considered as sequences of *states* rather than our treatment of traces as sequences of *events*. In this paper we use the infinite traces model in order to give a proper treatment of divergence under hiding. This is required to establish our main result concerning divergence-freedom under hiding of new events. Consideration of finite traces alone is not sufficient for this result.

More specifically, infinite traces need to be explicitly recorded since in the presence of unbounded nondeterminism they cannot be derived from the finite traces alone. Event-B easily allows for specifying unbounded nondeterminism. For this, consider for instance a machine initially choosing a value of some variable n from NAT and then executing an event ev n times. This machine has no infinite trace of ev events, but finite traces of every length n. We need to distinguish this machine from a machine which simply executes ev forever since when hiding this event one machine will have divergent behaviour while the other does not.

The CSP semantics is based on the weakest-precondition semantics of events. Let S be a statement (of an event). Then, following the notation of [Abr05] we will use $[S]R$ to denote the weakest precondition for statement S to establish postcondition R. Weakest preconditions for events of the form **when** $G(v)$ **then** $S(v)$ **end** are given using Morgan's semantics for guarded commands [Mor88]:

$$[\,\textbf{when}\;\; G(v)\;\; \textbf{then}\;\; S(v)\;\; \textbf{end}\,]P = G(v) \Rightarrow [S(v)]P$$

An event **when** $G(v)$ **then** $S(v)$ **end** executes $S(v)$ when the guard $G(v)$ is true, but it is blocked and cannot execute when the guard is false. Intuitively, [**when** $G(v)$ **then** $S(v)$ **end**]P is true in a state when every execution of the event from that state is guaranteed to terminate and reach a state in which P holds. Hence when $G(v)$ is true we require that $[S(v)]P$ holds, that every execution of $S(v)$ is guaranteed to terminate and reach a state in which P holds. Conversely, when $G(v)$ is false then it is vacuously true that every execution of **when** $G(v)$ **then** $S(v)$ **end** terminates and reaches a state where P holds, since there are no such executions.

Similarly, events in the general form **when** $G(v)$ **then** $v :\mid BA(v, v')$ **end** have a weakest-precondition semantics as follows:

$$[\,\textbf{when}\;\; G(v)\;\; \textbf{then}\;\; v :\mid BA(v,v')\;\; \textbf{end}\,]P = G(v) \Rightarrow \forall\, v'.(BA(v,v') \Rightarrow P[v'/v])$$

Observe that for the case $P = \mathit{true}$ we have

$$[\,\textbf{when}\;\; G(v)\;\; \textbf{then}\;\; v :\mid BA(v,v')\;\; \textbf{end}\,]\mathit{true} = \mathit{true}$$

Weakest preconditions and proof obligations are closely connected, and proof obligations associated with machines can also be expressed and understood in weakest-precondition terms. For example the requirement that an event ev should preserve the invariant I is naturally expressed as

$$I \Rightarrow [ev]I$$

This states that if the invariant I holds for an initial state then performing ev from that state should reach a final state where I again holds. Applying the weakest precondition semantics for an event of the form **when** $G(v)$ **then** $v :\mid BA(v, v')$ **end** we obtain the requirement

$$I(v) \Rightarrow [\,\textbf{when}\;\; G(v)\;\; \textbf{then}\;\; v :\mid BA(v,v')\;\; \textbf{end}\,]I(v)$$

This reduces to the equivalent requirement (where v and v' are both under an implicit universal quantification):

$$(I(v) \wedge G(v) \wedge BA(v,v') \Rightarrow I(v'))$$

Based on the weakest precondition semantics of [Mor90, But92] for action systems, we can define the traces, divergences and infinite traces of an Event-B machine[2].

Traces The traces of a machine M are those sequences of events $tr = \langle a_1, \ldots, a_n\rangle$ which are possible for M (after initialisation $init$): those that do not establish false:

$$\mathit{traces}(M) = \{tr \mid \neg[\mathit{init};tr]\mathit{false}\}$$

[2] Failures can be defined as well but are omitted since they are not needed for our approach.

Here, the weakest precondition on a sequence of events is the weakest precondition of the sequential composition of those events: $[\langle a_1, \ldots, a_n \rangle]P$ is given as $[a_1; \ \ldots \ ; a_n]P = [a_1](\ldots([a_n]P)\ldots)$.

Divergences A sequence of events tr is a divergence if the sequence of events is not guaranteed to terminate, i.e., $\neg[init; \ tr]true$. Thus

$$divergences(M) = \{tr \mid \neg[init;tr]true\}$$

Note that any Event-B machine M with events of the form ev given in Fig. 1 is divergence-free. This is because $[ev]true = true$ for such events (and for $init$), and so $[init; \ tr]true = true$. Thus no potential divergence tr meets the condition $\neg[init; \ tr]true$.

Infinite Traces The technical definition of infinite traces is given in [But92], in terms of least fixed points of predicate transformers on infinite vectors of predicates. Informally, an infinite sequence of events $u = \langle u_0, u_1, \ldots \rangle$ is an infinite trace of M if there is an infinite sequence of predicates P_i such that $\neg[init](\neg P_0)$ (i.e., some execution of $init$ reaches a state where P_0 holds), and $P_i \Rightarrow \neg[u_i](\neg P_{i+1})$ for each i (i.e., if P_i holds then some execution of u_i can reach a state where P_{i+1} holds).

$$\begin{aligned} infinites(M) = \{u \mid \text{there is a sequence} \langle P_i \rangle_{i \in \mathbb{N}} \ . \ & \neg[init](\neg P_0) \wedge \\ & \text{for all } i \ . \ P_i \Rightarrow \neg[u_i](\neg P_{i+1}) \ \} \end{aligned}$$

These definitions give the CSP Traces/Divergences/Infinite Traces semantic model of Event-B machines in terms of the weakest-precondition semantics of events. For the above given Event-B machine $Basket0$ we for instance get $\langle empty, empty, complete, empty \rangle \in traces(Basket0)$, $\langle empty, empty, empty, \ldots \rangle \in infinites(Basket0)$ and $divergences(Basket0) = \{\}$. The infinite sequence of predicates P_i for deriving infinite traces is here always $P_i = true$ for all i.

4. Refinement

In this paper, we intend to give a CSP account of Event-B refinement. The previous section provides us with a technique for relating Event-B machines to the semantic domain of CSP processes. Next, we will briefly rephrase the refinement concepts in CSP and Event-B before explaining Event-B refinement in terms of CSP refinement.

4.1. Event-B refinement

In Event-B, the (intended) refinement relationship between machines is directly written into the machine definitions. As a consequence of writing a refining machine, a number of proof obligations come up. Here, we assume a machine and its refinement take the following form:

machine M_0
variables v
invariant $I(v)$
events $init_0, ev_0, ev_0', \ldots$
end

machine M_1
refines M_0
variables w
invariant $J(v, w)$
variant $V(w)$
events $init_1, ev_1, ev_1', \ldots$
end

The machine M_0 is actually refined by machine M_1, written $M_0 \preccurlyeq M_1$, if the given *linking invariant* J on the variables of the two machines is established by their initialisations, and preserved by all events. Any transition performed by a concrete event of M_1 can be matched by a step of the corresponding abstract event of M_0 (or *skip* for newly introduced events) to maintain J. This is similar to the approach of downwards simulation data refinement [DB01]. We next look at this in more detail, and give the proof obligations associated with these conditions. Formally, the refinement relation $M_0 \preccurlyeq M_1$ between abstract machine M_0 and concrete machine M_1 holds if the four proof obligations given below all hold: **FIS_REF**, **GRD_REF**, **INV_REF** and **WFD_REF**. The first three must hold for all events, and the fourth gives the additional requirement for convergent and anticipated events.

First of all, we need to look at events again. Figure 3 gives the shape of an event and a refinement

```
$ev_0 \;\widehat{=}$                         $ev_1 \;\widehat{=}$
    **status** $st_0$                        **refines** $ev_0$
    **when**                                 **status** $st_1$
        $G(v)$                               **when**
    **then**                                     $H(w)$
        $v :\mid BA0(v, v')$                 **then**
    **end**                                      $w :\mid BA1(w, w')$
                                             **end**
```

Fig. 3. An event and a refinement of it

```
$ev_1 \;\widehat{=}$
    **status** $st_1$
    **when**
        $H(w)$
    **then**
        $w :\mid BA1(w, w')$
    **end**
```

Fig. 4. A new event not refining any event

of it. Fig. 4 gives the shape of a new event, which (implicitly) refines *skip*. We see that an event in the refinement now also gets a *status*. The status can be ordinary (not noted in this case), or *anticipated* or *convergent*. There are different strategies on how these types of events are systematically used in an Event-B development via refinement. The basic idea however is that convergent events are those which are considered internal and must not be executed forever, and anticipated events are those which will take on their roles in later refinement steps only and in these might become convergent. Later, we present two development strategies which precisely fix the use of convergent and anticipated events: Section 6 will give rules for the use of convergent and anticipated events in strategy I, and Section 7 will do so for strategy II.

Both convergent and anticipated events introduce further proof obligations: to prevent execution "forever" the refining machine has to give a variant V (see above in M_1), and V has to be decreased by every convergent event and must not be increased by anticipated events. Before going into the proof obligations in detail, we extend the example of the last section by another machine *Basket*1 refining *Basket*0.

The machine *Basket*1, given in Figure 5, introduces one new possible state for the basket, *state_changing*. This describes the state of the basket during shopping when the customer inserts or removes goods into / from the basket. A corresponding event *change* brings the basket into this state. This event has status anticipated: we have not yet decided on its real use. Besides defining the possible values for *state*, the invariant now also specifies the link between the variables of *Basket*0 and *Basket*1, and we see that the new state *state_changing* matches both of the old states.

To see that *Basket*1 is indeed a refinement of *Basket*0 we next describe each of the proof obligations in turn. We have simplified them from their form in [MAV05] by removing explicit references to sets and constants. Alternative forms of these proof obligations are given in [Abr10, Section 5.2: Proof Obligation Rules].

FIS_REF: Feasibility 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. In other words, the body of the event will not block when the event is enabled.

The rule for feasibility of a concrete event is:

$I(v) \wedge J(v, w) \wedge H(w)$ $\vdash$ $\exists\, w'.BA1(w, w')$	**FIS_REF**

```
machine $Basket1$
refines $Basket0$
variables $state$
invariant
  $state \in \{state\_complete, state\_empty, state\_changing\} \wedge$
  $state = state\_complete \Rightarrow state0 = state\_complete \wedge$
  $state = state\_empty \Rightarrow state0 = state\_empty$
variant 0
events
  init $\widehat{=}$
    $state := state\_empty$
  checkout $\widehat{=}$
    $state := state\_complete$
  empty $\widehat{=}$
    $state := state\_empty$
  change $\widehat{=}$
     status : anticipated
    $state := state\_changing$
end
```

Fig. 5. Event-B machine *Basket*1

The proof obligation FIS_REF is easy to show for all the events of *Basket*1 as all events have simple assignments instead of before-after-predicates.

GRD_REF: Guard Strengthening This requires that when a concrete event is enabled, then so is the abstract one. The rule is:

$I(v) \wedge J(v,w) \wedge H(w) \vdash G(v)$	**GRD_REF**

Showing GRD_REF for *Basket*1 is straightforward as well as both abstract and concrete events have guards equal to true. Later, we will see more complicated refinements. Observe that this proof obligation is trivial if the abstract event is *skip*, since in that case $G(v)$ is *true*.

INV_REF: Simulation This ensures that the occurrence of events in the concrete machine can be matched in the abstract one (including the initialization event). If there is an abstract event then the rule is:

$I(v) \wedge J(v,w) \wedge H(w) \wedge BA1(w,w') \vdash \exists\, v'.(BA0(v,v') \wedge J(v',w'))$	**INV_REF**

New events are treated as refinements of *skip*. In this case the abstract state does not change (i.e., $BA0$ is the identity relation), and the rule is

$I(v) \wedge J(v,w) \wedge H(w) \wedge BA1(w,w') \vdash J(v,w'))$	**INV_REF**

For *Basket*1, event *change* is a new event. It corresponds to *skip* as it is always enabled and only changes the state into *state_changing*, which corresponds to both of the abstract states *state_empty* and *state_complete*.

The two parts of the variant rule WFD_REF below must hold respectively for all convergent and anticipated events.

WFD_REF: Variant This rule ensures that the proposed variant V 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:

$I(v) \wedge J(v,w) \wedge H(w) \wedge BA1(w,w')$ $\vdash$ $V(w) \in \mathbb{N} \wedge V(w') < V(w)$	**WFD_REF (convergent event)**

$I(v) \wedge J(v,w) \wedge H(w) \wedge BA1(w,w')$ $\vdash$ $V(w) \in \mathbb{N} \wedge V(w') \leqslant V(w)$	**WFD_REF (anticipated event)**

The variant used for $Basket0$ is a constant and thus kept by the anticipated event *change*. This completes the proof that $Basket1$ is a refinement of $Basket0$.

Event-B also allows a variety of further optional proof obligations on refinement steps, depending on what is appropriate for the application. These capture additional properties concerned with liveness (enabledness of sets of events) and deadlock-freedom (enabledness of at least one event), and are not the concern of this paper.

4.2. CSP refinement

Based on the semantic domains of traces, failures, divergences and infinite traces, different forms of refinement can be given for CSP. The basic idea underlying these concepts is, however, always the same: the refining process should not exhibit a behaviour which was not possible in the refined process. The different semantic domains then supply us with different forms of "behaviour". In this paper we will use the following refinement relation, based on traces and divergences:

$$\begin{aligned} P \sqsubseteq_{TDI} Q \mathrel{\widehat{=}} & \ \mathit{traces}(Q) \subseteq \mathit{traces}(P) \\ & \wedge \mathit{divergences}(Q) \subseteq \mathit{divergences}(P) \\ & \wedge \mathit{infinites}(Q) \subseteq \mathit{infinites}(P) \end{aligned}$$

Refinement in Event-B also allows for the possibility of introducing new events. CSP refinement as defined above, however, assumes the alphabet of the processes to be the same. As our objective is to formally state the relationship between Event-B machines in a refinement chain via a form of CSP refinement, we need a way of incorporating additional events into process refinement. As a first idea, we could *hide* the new events in the refining process, and only afterwards check for refinement. Hiding potentially introduces divergences, namely, when there is an infinite sequence of new events in the infinite traces. When this is the case, we have no $\sqsubseteq_{TDI}$ relationship. Thus hiding gives us the appropriate treatment of new events only in the case where no infinite sequences of new events occur (and this is what Event-B proof obligations guarantee for some sort of new events). As an alternative way of treating new events, we will use $P \mid\mid\mid RUN_N$ as a lazy abstraction operator [Ros98]. RUN_N defines a divergence-free process capable of executing any order of events from the set N:

$$\begin{aligned} & \mathit{divergences}(RUN_N) = \{\} \\ & \mathit{traces}(RUN_N) = N^* \\ & \mathit{infinites}(RUN_N) = N^\omega \end{aligned}$$

This process together with the interleaving operator will enable us to characterise Event-B refinement introducing new events in CSP terms. The RUN_N process gives no restriction on the behaviour of new events. Thus comparing Event-B machines P (or better, its CSP semantics) and P' which has introduced new events N via checking of $P \mid\mid\mid RUN_N \sqsubseteq_{TDI} P'$ would mean that we check whether there is a refinement relation concerning the "old" events while allowing for every behaviour of the new events.

The following lemma gives the relationship between refinement involving interleaving, and refinement involving hiding.

Lemma 4.1. If $P_0 \mid\mid\mid RUN_N \sqsubseteq_{TDI} P_1$ and $N \cap \alpha P_0 = \{\}$ and $P_1 \setminus N$ is divergence-free, then $P_0 \sqsubseteq_{TDI} P_1 \setminus N$.

Proof: Assume that (1) $P_0 \mid\mid\mid RUN_N \sqsubseteq_{TDI} P_1$, (2) $N \cap \alpha P_0 = \{\}$ and (3) $P_1 \setminus N$ is divergence-free. We need to show that the (finite and infinite) traces as well as divergences of $P_1 \setminus N$ are contained in those of P_0.

Traces Let $tr \in \mathit{traces}(P_1 \setminus N)$. By semantics of hiding there is some $tr' \in \mathit{traces}(P_1)$ s.t. $tr' \setminus N = tr$. By (1) $tr' \in \mathit{traces}(P_0 \mid\mid\mid RUN_N)$. By (2) and the semantics of $\mid\mid\mid$ we get $tr' \setminus N \in \mathit{traces}(P_0)$ and thus $tr \in \mathit{traces}(P_0)$.

Divergences By (3) $\mathit{divergences}(P_1 \setminus N) = \{\}$, thus nothing to be proven here.

Infinites Let $u \in \mathit{infinites}(P_1 \setminus N)$. By the semantics of hiding there is some $u' \in \mathit{infinites}(P_1)$ such that $u' \setminus N = u$ and $\#(u' \setminus N) = \infty$. By (1) $u' \in \mathit{infinites}(P_0 \mid\mid\mid RUN_N)$ and by (2) and semantics of interleave we get $u' \setminus N = u \in \mathit{infinites}(P_0)$.

□

As a side remark: The reverse direction holds as well. Note that we do not need the condition on alphabets here (as $N \cap \alpha P_0 = \{\}$ follows from $P_0 \sqsubseteq_{TDI} P_1 \setminus N$).

Lemma 4.2. If $P_0 \sqsubseteq_{TDI} P_1 \setminus N$ and $P_1 \setminus N$ divergence-free, then $P_0 \mid\mid\mid RUN_N \sqsubseteq_{TDI} P_1$.

Proof: Assume that (1) $P_0 \sqsubseteq_{TDI} P_1 \setminus N$ and (2) $P_1 \setminus N$ divergence-free.

Traces Let $tr \in \mathit{traces}(P_1)$. Then by semantics of hiding $tr \setminus N \in \mathit{traces}(P_1 \setminus N)$, and thus by (1) $tr \setminus N \in \mathit{traces}(P_0)$. Furthermore $tr \upharpoonright N \in \mathit{traces}(RUN_N)$, and $tr \in (tr \setminus N \mid\mid\mid tr \upharpoonright N)$. Thus $tr \in \mathit{traces}(P_0 \mid\mid\mid RUN_N)$.

Divergence As $P_1 \setminus N$ is divergence-free, so is P_1, hence nothing to be shown here.

Infinites Let $u \in \mathit{infinites}(P_1)$. As $P_1 \setminus N$ is divergence-free, $\#(u \setminus N) = \infty$. Thus by (1) $u \setminus N \in \mathit{infinites}(P_0)$. The projection onto N, $u \upharpoonright N$ is either finite or infinite, and $u \upharpoonright N \in \mathit{traces}(RUN_N)$ or $u \upharpoonright N \in \mathit{infinites}(RUN_N)$. In both cases $u \in (u \setminus N \mid\mid\mid u \upharpoonright N)$ and thus $u \in \mathit{infinites}(P_0 \mid\mid\mid RUN_N)$.

□

Both an interleaving with the RUN_N process and hiding of N can be used as a way of abstracting away from particular new events N occurring in a process P_1 but not in P_0. The two lemmas show that in the absence of a divergence on N in P_1 (i.e., P_1 has no infinite sequence with events eventually only from N) these two can be considered as dual.

5. Event-B refinement as CSP refinement

With these definitions in place, we can now look at our main issue, the characterisation of Event-B refinement via CSP refinement. Here, we in particular need to look at the different forms of events in Event-B during refinement. Events can have status convergent or anticipated, or might have no status. This partitions the set of events of M into three sets: anticipated A, convergent C, and the remaining ones, ordinary events O (neither anticipated nor convergent). The alphabet of M, the set of all possible events, is thus given by $\alpha M = A \cup C \cup O$. In the CSP refinement, these will take different roles.

A system development in Event-B via refinement usually proceeds in several steps, gradually introducing more detail into the specification, i.e., we will have a sequence of machines $M_0, M_1, \ldots, M_n$ related by refinement. Now consider an Event-B Machine M_i and its refinement M_{i+1}: $M_i \preccurlyeq M_{i+1}$. The machine M_i has anticipated events A_i, convergent events C_i, and ordinary events O_i, and M_{i+1} similarly has event sets A_{i+1}, C_{i+1}, and O_{i+1}. Each event ev_i in M_i is refined by at least one event of M_{i+1} (if no such event appears in M_{i+1} we assume the event to be refined by itself). Each event ev_{i+1} in M_{i+1} either refines a single event ev_i in M_i (indicated by the clause 'refines ev_i' in the description of ev_{i+1}) or does not refine any event of M_i. The set of new events N_{i+1} comprises those events which are not refinements of events in M_i. $M_i \preccurlyeq M_{i+1}$ thus induces a partial surjective function $f_{i+1} : \alpha M_{i+1} \twoheadrightarrow\!\!\!\!\!\!\!+\;\; \alpha M_i$ where $f_{i+1}(ev_{i+1}) = ev_i \Leftrightarrow ev_{i+1}$ **refines** ev_i. Observe that αM_{i+1} is partitioned by $f_{i+1}^{-1}(\alpha M_i)$ and N_{i+1}.

For the two Event-B machines $Basket0$ and $Basket1$ we have $O_0 = \{empty, checkout\}$, $A_0 = C_0 = \{\}$, and $O_1 = \{empty, checkout\}$ and $N_1 = A_1 = \{change\}$. The function f_1 thus is the mapping $\{empty \mapsto empty, checkout \mapsto checkout\}$.

5.1. New events

The above definitions fix the relation between events in M_i and M_{i+1} independent of a particular development strategy. With these at hand, we can look at the CSP semantics of Event-B and see what the corresponding notion of refinement in CSP is. First of all, for the new events arising in the refinement we can use the lazy abstraction operator via the RUN process to get our desired result, disregarding the issue of divergence for a moment.

The following lemma gives our first result on the relationship between Event-B refinement and CSP refinement. Observe that we use priming notation when considering the general relationship between a machine M and its refinement M', rather than the indexing notation used for refinement chains.

Lemma 5.1. If $M \preccurlyeq M'$ and the refinement introduces new events N' and uses the mapping f', then $f'^{-1}(M) \mid\mid\mid RUN_{N'} \sqsubseteq_{TDI} M'$.

Proof: We assume state variables of M and M' named as given above, i.e., state variables of M are v and of M' are w. Let $tr = \langle a_1, \ldots, a_n \rangle \in traces(M')$. We need to show that $tr \in traces(f'^{-1}(M) \mid\mid\mid RUN_{N'})$. First of all note that the interleaving operator merges the traces of two processes together, i.e., the traces of $f'^{-1}(M) \mid\mid\mid RUN_{N'}$ are simply those of $f'^{-1}(M)$ with new events arbitrarily inserted. The proof proceeds by induction on the length of the trace.

Induction base Assume $n = 0$, i.e., $tr = \langle\rangle$. This case is trivial, since $\langle\rangle \in traces(f'^{-1}(M) \mid\mid\mid RUN_{N'})$.

Induction step Assume that for a trace $tr_0 = \langle a_1, \ldots, a_{j-1} \rangle \in traces(M')$ we have already shown that $tr_0 \in traces(f'^{-1}(M) \mid\mid\mid RUN_{N'})$ and this has led us to a pair of states v_{j-1}, w_{j-1} such that $J(v_{j-1}, w_{j-1})$. (Observe that in the case where $tr_0 = \langle\rangle$ the initialisation event $init'$ has been executed bringing the machine M' into a state w_0. By INV_REF on $init$, we find a state v_0 such that $J(v_0, w_0)$.)
Now two cases need to be considered:

1. $a_j \notin N'$: Assume a_j in M' to be of the form

   $$\textbf{when}\ \ H(w)\ \ \textbf{then}\ \ w :\mid BA'(w, w')\ \ \textbf{end}$$

   and $f'(a_j)$ in M of the form

   $$\textbf{when}\ \ G(v)\ \ \textbf{then}\ \ v :\mid BA(v, v')\ \ \textbf{end}$$

   Since a_j is executed in w_{j-1} we have $H(w_{j-1})$. By GRD_REF we thus get $G(v_{j-1})$. Furthermore, for w_j with $BA'(w_{j-1}, w_j)$ we find, by INV_REF, a state v_j such that $J(v_j, w_j)$ and $BA(v_{j-1}, v_j)$. Hence $tr_0 \frown \langle a_j \rangle \in traces(f'^{-1}(M) \mid\mid\mid RUN_{N'})$.
2. $a_j \in N'$: Similar to the previous case. Here, a_j refines skip and thus $v_j = v_{j-1}$ and the event a_j comes from $RUN_{N'}$.

In the same way we can carry out a proof for infinite traces. For divergences it is trivial since $divergences(M') = \{\}$. □

For our example, we thus get $f_1^{-1}(Basket0) \mid\mid\mid RUN_{\{change\}} \sqsubseteq_{TDI} Basket1$.

This lemma can be generalised to a chain of refinement steps. For this, we assume that we are given a sequence of Event-B machines M_i with their associated processes P_i, and every refinement step introduces some set of new events N_i.

Theorem 5.2. If a sequence of processes P_i, mappings f_i, and sets N_i are such that

$$f_{i+1}^{-1}(P_i) \mid\mid\mid RUN_{N_{i+1}} \sqsubseteq_{TDI} P_{i+1} \tag{1}$$

for each i, then

$$f_n^{-1}(\ldots(f_1^{-1}(P_0))\ldots) \mid\mid\mid RUN_{f_n^{-1}(\ldots f_2^{-1}(N_1)\ldots)\cup\ldots\cup f_n^{-1}(N_{n-1})\cup N_n} \sqsubseteq_{TDI} P_n$$

```
machine $Basket2$
refines $Basket1$
variables $state, tot$
invariant $tot \in NAT$
variant $0$
events
  init $\widehat{=}$
     $state := state\_empty \; || \; tot := 0$
  checkout $\widehat{=}$
     when $tot > 0$
     then $state := state\_complete$ end
  empty $\widehat{=}$
     when $tot = 0$
     then $state := state\_empty$ end
  add $\widehat{=}$
     refines $change$
     status : anticipated
    $tot := tot + 1 \; || \; state := state\_changing$
  remove $\widehat{=}$
     refines $change$
     status : anticipated
     when $tot > 0$
     then $tot := tot - 1 \; || \; state := state\_changing$ end
end
```

Fig. 6. Event-B machine *Basket*2

Proof: Two successive refinement steps combine to provide a relationship between P and P'' of the same form as Line 1 above, as follows:

$$\begin{array}{rcll} f''^{-1}(P') \;|||\; RUN_{N''} & \sqsubseteq_{TDI} & P'' & \text{(given)} \\ f''^{-1}(f'^{-1}(P) \;|||\; RUN_{N'}) \;|||\; RUN_{N''} & \sqsubseteq_{TDI} & P'' & \text{(line (1), monotonicity of } f''^{-1}\text{, transitivity of } \sqsubseteq) \\ f''^{-1}(f'^{-1}(P)) \;|||\; RUN_{f''^{-1}(N')} \;|||\; RUN_{N''} & \sqsubseteq_{TDI} & P'' & \text{(Law: } f^{-1}(P \;|||\; Q) = f^{-1}(P) \;|||\; f^{-1}(Q)) \\ f''^{-1}(f'^{-1}(P)) \;|||\; RUN_{f''^{-1}(N') \cup N''} & \sqsubseteq_{TDI} & P'' & \text{(Law: } RUN_A \;|||\; RUN_B = RUN_{A \cup B}) \end{array}$$

Hence the whole chain of refinement steps can be collected together, yielding the result. □

To see how this works for our example, we extend our chain of refinements by another machine, *Basket*2, shown in Figure 6. In *Basket*2 we *split* one event, namely *change*, into two, adding and removing goods from the basket. Both these events are tagged as anticipated. For this refinement we thus have $N_2 = \{\}$, $A_2 = \{add, remove\}$, $C_2 = \{\}$ and $f_2 : \{add \mapsto change, remove \mapsto change\}$.

The above lemma on refinement chains and new events then gives us the following result, relating the initial machine *Basket*0 and *Basket*2.

$$f_2^{-1}(f_1^{-1}(Basket0)) \;|||\; RUN_{\{add,remove\}} \sqsubseteq_{TDI} Basket2 \; .$$

This result states that traces, infinite traces, and divergences of *Basket*2 must also be possible for $f_2^{-1}(f_1^{-1}(Basket0)) \;|||\; RUN_{\{add,remove\}}$. For example, consider the trace

$$\langle add, add, checkout, remove, remove, empty \rangle \in traces(Basket2)$$

We find that

$$\langle checkout, empty \rangle \in traces(f_2^{-1}(f_1^{-1}(Basket0)))$$

and

$$\langle add, add, remove, remove \rangle \in traces(RUN_{\{add,remove\}})$$

so indeed we find that

$$\langle add, add, checkout, remove, remove, empty\rangle \in traces(f_2^{-1}(f_1^{-1}(Basket0)) \mathbin{|||} RUN_{\{add,remove\}})$$

The infinite trace $\langle add\rangle^\omega infinites(Basket2)$, and it is also in $infinites(RUN_{\{add,remove\}})$, hence in $infinites(f_2^{-1}(f_1^{-1}(Basket0)) \mathbin{|||} RUN_{\{add,remove\}})$.

There are no divergences at any refinement level: no divergences can be introduced through refinement.

5.2. Convergent and anticipated events

The previous result lets us relate the first and last Event-B machine in a chain of refinements. Due to the lazy abstraction operator (and the resulting possibility of defining refinement without hiding new events), we considered divergence-free processes there: all processes P_i representing Event-B machines, are divergence free by definition. However, Event-B refinement is concerned with a particular form of divergence and its avoidance. A sort of divergence would arise when new events (or more specifically, convergent events) could be executed forever, and this is what the proof obligations for variants rule out.

We would like to capture the impact of convergence and anticipated sets of events in the CSP semantics as well. To do so, we first of all define the specification predicate

$$CA(C, O)(u) \mathrel{\widehat{=}} (\#(u \upharpoonright C) = \infty \Rightarrow \#(u \upharpoonright O) = \infty)$$

Intuitively, this states that all infinite traces having infinitely many convergent (C) events also have infinitely many (O) ordinary events (and thus cannot execute convergent events alone forever). In this case we say that the Event-B machine *does not diverge on C events*.

Definition 5.3. Let M be an Event-B machine with its alphabet αM containing event sets C and O with $C \cap O = \{\}$. M *does not diverge on C events* if M **sat** $CA(C, O)$.

Observe that if a machine has no convergent events ($C = \{\}$) then it trivially satisfies the specification $CA(C, O)$. Note that the first machine M_0 in an Event-B refinement chain has no convergent events — convergent events only come into play during refinement — so it will satisfy $CA(C, O)$.

Lemma 5.4. If $M \preccurlyeq M'$, and M' has convergent, anticipated, and ordinary sets of events C', A', and O' respectively, then M' **sat** $CA(C', O')$.

Proof: We prove this by contradiction. Assume $\neg M'$ **sat** $CA(C', O')$. Then there is some $u \in infinites(M')$ such that $\#(u \upharpoonright C') = \infty$ and $\#(u \upharpoonright O') < \infty$. Then there must be some tr_0, u' such that $u = tr_0 \frown u'$ with $u' \in (C' \cup A')^\omega$ (i.e., tr_0 is a prefix of u containing all the O' events). Moreover, $\#u' \upharpoonright C' = \infty$.

Now since $M \preccurlyeq M'$ we have by GRD_REF and INV_REF that there is some pair of states (v, w) (abstract and concrete state) reached after executing tr_0 for which $J(v, w)$ and $I(v)$ is true. Furthermore, $V(w)$ is a natural number. Also by $M \preccurlyeq M'$ we have an infinite sequence of pairs of states (v_i, w_i) (for the remaining infinite trace u') such that $J(v_i, w_i)$. Since each event in u' is in A' or C' we have from WFD_REF that $V(w_{i+1}) \leqslant V(w_i)$ for each i. Further, for infinitely many i's (i.e., those events in C') we have $V(w_{i+1}) < V(w_i)$. Thus we have a sequence of values $V(w_i)$ decreasing infinitely often without ever increasing. This contradicts the fact that the $V(w_i) \in \mathbb{N}$. □

A number of further interesting properties can be deduced for the specification predicate CA.

Lemma 5.5. Let P be a CSP process and $C, C', O \subseteq \alpha P$ nonempty finite sets of events.

1. If P **sat** $CA(C, O)$ then $f^{-1}(P)$ **sat** $CA(f^{-1}(C), f^{-1}(O))$.
2. If P **sat** $CA(C, O)$ and $N \cap C = \{\}$ then $P \mathbin{|||} RUN_N$ **sat** $CA(C, O)$.
3. If P **sat** $CA(C, O)$ and P **sat** $CA(C', C \cup O)$ then P **sat** $CA(C \cup C', O)$.
4. If P **sat** $CA(C, O)$ and $C \cap O = \{\}$ then $P \setminus C$ is divergence-free.
5. If P **sat** $CA(C, O)$ then P **sat** $CA(C, O \cup X)$ for every set X of events.

Proof:

1. Assume that $u \in \mathit{infinites}(f^{-1}(P))$ and $\#(u \upharpoonright f^{-1}(C)) = \infty$. From the first we get $f(u) \in \mathit{infinites}(P)$. From the latter it follows that $\#(f(u) \upharpoonright C) = \infty$. With P **sat** $CA(C, O)$ we have $\#(f(u) \upharpoonright O) = \infty$ and hence $\#(u \upharpoonright f^{-1}(O)) = \infty$.
2. Let $u \in \mathit{infinites}(P \mathbin{|||} RUN_N)$ and $\#(u \upharpoonright C) = \infty$. With $N \cap C = \{\}$ we get $\#((u \setminus N) \upharpoonright C) = \infty$. By definition of $|||$ we have $u \setminus N \in \mathit{infinites}(P)$ ($u \setminus N$ is infinite since $\#((u \setminus N) \upharpoonright C) = \infty$). By P **sat** $CA(C, O)$ we get $\#((u \setminus N) \upharpoonright O) = \infty$, hence $\#(u \upharpoonright O) = \infty$.
3. Let $u \in \mathit{infinites}(P)$ such that $\#(u \upharpoonright (C \cup C')) = \infty$. Both C and C' are finite sets hence either $\#(u \upharpoonright C) = \infty$ or $\#(u \upharpoonright C') = \infty$ (or both). In the first case we get $\#(u \upharpoonright O) = \infty$ by P **sat** $CA(C, O)$. In the second case it follows that $\#(u \upharpoonright (C \cup O)) = \infty$ and hence again $\#(u \upharpoonright C) = \infty$ or directly $\#(u \upharpoonright O) = \infty$.
4. First of all note that if P **sat** $CA(C, O)$ then P is divergence free. Now assume that there is a trace $tr \in \mathit{divergences}(P \setminus C)$. Then there exists a trace $u \in \mathit{infinites}(P)$ such that $tr = u \setminus C$, and so $\#(u \setminus C) < \infty$. Hence $\#(u \upharpoonright C) = \infty$. However, as $C \cap O = \{\}$, $\#(u \upharpoonright O) \neq \infty$ which contradicts P **sat** $CA(C, O)$.
5. Follows from the fact that if $\#(u \upharpoonright O) = \infty$ then $\#(u \upharpoonright O \cup X) = \infty$.

□

The most interesting of these properties is probably the fourth one: it relates the specification predicate to the definition of divergence freedom in CSP. In CSP, a process does not diverge on a set of events C if $P \setminus C$ is divergence-free.

We also obtain a result for the specification predicate CA with respect to the relationship between a machine and a refinement of it.

Lemma 5.6. Let $M \preccurlyeq M'$ with an associated refinement function f'. Let C, C', O and O' be convergent and ordinary sets such that M **sat** $CA(C, O)$ and M' **sat** $CA(C', O')$, and $O' = f'^{-1}(C) \cup f'^{-1}(O) \cup X$. The set X is the ordinary events in M' that do not refine ordinary or convergent events in M. Then:

M' **sat** $CA(f'^{-1}(C) \cup C'\ ,\ f'^{-1}(O) \cup X)$

Proof: Assume $u \in \mathit{infinites}(M')$ and $\#(u \upharpoonright (f'^{-1}(C) \cup C')) = \infty$. We aim to establish that $\#(u \upharpoonright f'^{-1}(O) \cup X) = \infty$. We have $\#(u \upharpoonright f'^{-1}(C)) = \infty$ or $\#(u \upharpoonright C') = \infty$.

In the former case, Lemma 5.1 yields that $f'(u \upharpoonright f^{-1}(\alpha M)) \in \mathit{infinites}(M)$. Then

$$\begin{array}{rl}
\#(u \upharpoonright f'^{-1}(C)) = \infty & \text{(given)} \\
\#(f'(u \upharpoonright f'^{-1}(C)) \upharpoonright C) = \infty & \text{(since renaming preserves length)} \\
\#(f'(u \upharpoonright f'^{-1}(\alpha M)) \upharpoonright C) = \infty & \text{(since } C \subseteq \alpha M) \\
\#(f'(u \upharpoonright f'^{-1}(\alpha M)) \upharpoonright O) = \infty & \text{(by } M \textbf{ sat } CA(C, O)) \\
\#(u \upharpoonright f'^{-1}(\alpha M)) \upharpoonright f'^{-1}(O) = \infty & \text{(since renaming preserves length)} \\
\#(u \upharpoonright f'^{-1}(O)) = \infty & \text{(since } O \subseteq \alpha M) \\
\#(u \upharpoonright f'^{-1}(O) \cup X) = \infty & \text{(since } f'^{-1}(O) \subseteq f'^{-1}(O) \cup X)
\end{array}$$

In the latter case Lemma 5.4 yields that $\#(u \upharpoonright O') = \infty$. Then

$$\begin{array}{rl}
\#(u \upharpoonright O') = \infty & \\
\#(u \upharpoonright f'^{-1}(C) \cup f'^{-1}(O) \cup X) = \infty & \text{(since } O' = f'^{-1}(C \cup O) \cup X) \\
\#(u \upharpoonright f'^{-1}(O) \cup X) = \infty \vee \#(u \upharpoonright f'^{-1}(C)) = \infty &
\end{array}$$

The first disjunct is the desired result, the second is the one already treated above.

□

We use the CA specification predicate to reason about (the absence of) divergence on particular events in an Event-B machine. However, from now on our results will be specific to a particular development strategy, and the way it imposes rules on convergent and anticipated events. Therefore, we next present our first development strategy.

6. Strategy I

In an Event-B machine the status of an event fixes its type and thus the proof obligations derived for the event. However, it is not only the status and its proof obligations which influences the corresponding notion of refinement in CSP. Another decisive issue is the *development strategy.* A development strategy essentially fixes how anticipated, convergent and ordinary events are used in a development via refinement, in particular, how the status of an event may change in a refinement step. Though development strategies have not been discussed explicitly in the Event-B literature so far, let alone formalised, it is clear that Event-B designers actually follow strategies (though not necessary all the same). So, in the following we will (for the first time) define (two) development strategies for Event-B.

The first strategy, which we have already presented in [STW11b], assumes that the initial specification M_0 already fixes the interface of the system to the environment, and later refinement steps only make the internals of the system more precise. We call this approach refinement with *no interface extension.* The second approach assumes that the interface can be extended in every refinement step, called refinement with *interface extension.* The role of anticipated and convergent events in these approaches is the following. One idea behind the Event-B refinement proof obligations is that internal events should not be executed forever, i.e., there should not be an infinite sequence of internal events only so as to enable interaction with the environment via the interface. Therefore we have particular proof obligations (using variants) on anticipated and convergent events. In the non-extensible interface approach, all newly introduced events are internal and thus eventually need to be shown to converge. In the extensible interface approach there is the option of a newly introduced event to become part of the interface and thus not to be shown to be convergent.

In this section we will follow the strategy I (no interface extension), and in particular precisely define what rules this approach imposes on refinement. The next section will look at strategy II. With non-extensible interfaces we have the following restrictions on the event sets in a development $M_0 \preccurlyeq M_1 \preccurlyeq \ldots \preccurlyeq M_n$.

1. in the initial machine all events are ordinary (and thus fix the interface of the system),
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;
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}, i.e., they are not given a status;
6. no anticipated events remain in the final machine.

The conditions imposed by the rules are formalised as follows:

1. $A_0 = C_0 = \{\}$;
2. $ran(f_{i+1}) = A_i \cup C_i \cup O_i$;
3. $N_i = (A_i \cup C_i) \setminus dom(f_i)$;
4. $f_{i+1}^{-1}(A_i) \subseteq A_{i+1} \cup C_{i+1}$;
5. $f_{i+1}^{-1}(C_i \cup O_i) = f_{i+1}^{-1}(C_i) \cup f_{i+1}^{-1}(O_i) = O_{i+1}$;
6. $A_n = \{\}$.

These relationships between the classes of events are illustrated in Fig. 7.

Note that condition 5 satisfies the condition of Lemma 5.6 (with $X = \{\}$), and so that lemma applies to refinement steps in strategy I.

To be able to apply this to our example, we extend it with a new machine *Basket*3.*I* (depicted in Figure 8) containing for the first time convergent events. In the second development strategy we will take a different route here and instead make a refinement to machine *Basket*3.*II*. Here, the basket is extended with variable *tot* counting the number of goods in the basket, and a boolean variable *scanning* as a flag for the start of the scanning procedure at the counter. Event *checkout* starts the scanning procedure and event *scan* scans every good in the basket. Event *empty* can happen when the basket is empty. It stops scanning and thereby enables event *add* again. Observe that while the basket is not scanning, *scan* cannot occur before *checkout* occurs. This ensures that the trace $\langle add, remove\rangle^\omega$ cannot occur. Since this trace violates $CA(C_{3.I}, O_{3.I})$, ruling it out is necessary to ensure that $CA(C_{3.I}, O_{3.I})$ holds on *Basket*3.*I*.

Since $Basket2 \preccurlyeq Basket3.I$ we have that

$$f_{3.I}^{-1}(Basket2) \mid\mid\mid RUN_{\{\}} \sqsubseteq_{TDI} Basket3.I$$

No new event is introduced in *Basket*3.*I*, and this is equivalent to the simpler formulation

$$f_{3.I}^{-1}(Basket2) \sqsubseteq_{TDI} Basket3.I$$

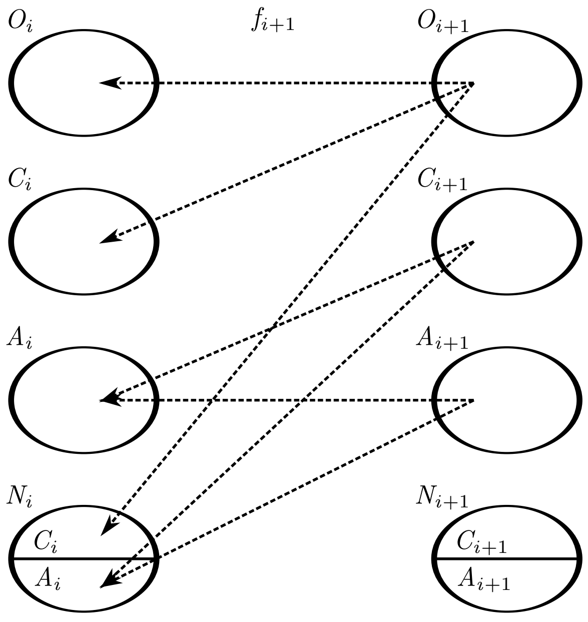


Fig. 7. Relationship between events in a refinement step in the non-extensible interface approach: f_{i+1} maps events in M_{i+1} to events in M_i that they refine.

For example, the trace $\langle \mathit{add}, \mathit{add}, \mathit{checkout}, \mathit{scan}, \mathit{scan}, \mathit{empty} \rangle \in \mathit{traces}(\mathit{Basket}3.I)$ corresponds, via the renaming function $f_{3.I}$, to the trace $\langle \mathit{add}, \mathit{add}, \mathit{checkout}, \mathit{remove}, \mathit{remove}, \mathit{empty} \rangle \in \mathit{traces}(\mathit{Basket}2)$. The results states that any trace of $\mathit{Basket}3.I$ must correspond to some trace of $\mathit{Basket}2$.

The converse need not hold. The trace $\langle \mathit{add}, \mathit{remove} \rangle \in \mathit{traces}(\mathit{Basket}2)$ does not have any associated trace in $\mathit{Basket}_{3.I}$.

The event *scan* is tagged convergent and thus the proof obligation WFD_REF needs to be shown. The variant specified in the machine is *if scanning* $=$ *true then tot else* 0. It is decreased by *scan* as it sets *tot* to $\mathit{tot} - 1$. Hence we obtain that $\mathit{Basket}3.I \setminus \{\mathit{scan}\}$ is divergence-free.

Thus we obtain
We require the following lemma in order to establish Theorem 6.2 subsequently:

Lemma 6.1. If $M_0 \preccurlyeq M_1 \preccurlyeq \ldots \preccurlyeq M_n$ then

$$O_n = (f_n^{-1}(\ldots f_1^{-1}(C_0)\ldots) \cup \ldots f_n^{-1}(C_{n-1})) \cup f_n^{-1}(\ldots f_1^{-1}(O_0)\ldots)$$

Proof: By induction on n.

Induction base Assume $n = 0$. In this case the statement reduces to $O_0 = O_0$ which is trivially true.

Induction step Assume the result for i, we aim to establish it for $i+1$. By Condition 5 of strategy I we have that $O_{i+1} = f_{i+1}^{-1}(C_i) \cup f_{i+1}^{-1}(O_i)$. Hence

$$\begin{array}{rcll} O_{i+1} & = & f_{i+1}^{-1}(C_i) \cup f_{i+1}^{-1}(O_i) & \text{(condition 5)} \\ & = & f_{i+1}^{-1}(C_i) \cup f_{i+1}^{-1}(f_i^{-1}(\ldots f_1^{-1}(C_0)\ldots) \cup \ldots f_i^{-1}(C_{i-1})) & \\ & & \qquad \cup f_i^{-1}(\ldots f_1^{-1}(O_0)\ldots) & \text{(inductive hypothesis)} \\ & = & (f_{i+1}^{-1}(\ldots f_1^{-1}(C_0)\ldots) \cup \ldots f_{i+1}^{-1}(C_i)) \cup f_{i+1}^{-1}(\ldots f_1^{-1}(O_0)\ldots)) & \text{(reordering)} \end{array}$$

which establishes the case.

The result follows by induction. □

We now obtain the following result on refinement chains:

Theorem 6.2. If $M_0 \preccurlyeq M_1 \preccurlyeq \ldots \preccurlyeq M_n$ then

$$M_n \ \mathbf{sat}\ \mathit{CA}((f_n^{-1}(\ldots f_1^{-1}(C_0)\ldots) \cup \ldots f_n^{-1}(C_{n-1}) \cup C_n)\,,\, f_n^{-1}(\ldots f_1^{-1}(O_0)\ldots))$$

```
**machine** $Basket3.I$
**refines** $Basket2$
**variables** $tot, state, scanning$
**invariant** $scanning \in \{true, false\}$
**variant** $if\ scanning = true\ then\ tot\ else\ 0$
**events**
  init $\widehat{=}$
     $state := state\_empty \mid\mid scanning := false \mid\mid tot := 0$
  checkout $\widehat{=}$
     **when** $tot > 0$
     **then** $state := state\_complete \mid\mid scanning := true$ **end**
  empty $\widehat{=}$
     **when** $tot = 0$
     **then** $state := state\_empty \mid\mid scanning := false$ **end**
  add $\widehat{=}$
     **status :** anticipated
     **when** $scanning \neq true$
     **then** $tot := tot + 1 \mid\mid state := state\_changing$ **end**
  scan $\widehat{=}$
     **refines** $remove$
     **status :** convergent
     **when** $tot > 0 \wedge scanning = true$
     **then** $tot := tot - 1 \mid\mid state := state\_changing$ **end**
**end**
```

Fig. 8. Event-B machine *Basket*3.*I*

Proof: By induction on n.

Induction base Assume $n = 0$. In this case $C_0 = \{\}$ since M_0 contains no convergent events, and so M_0 **sat** $CA(C_0, O_0)$ is vacuously true. Observe that $C_0 = (f_n^{-1}(\dots f_1^{-1}(C_0)\dots) \cup \dots f_n^{-1}(C_{n-1}) \cup C_n)$, and $O_0 = f_n^{-1}(\dots f_1^{-1}(O_0)\dots))$ with $n = 0$, establishing the case.

Induction step Assume the result for i, we aim to establish it for $i+1$. The inductive hypothesis gives:

$$M_i \textbf{ sat } CA((f_i^{-1}(\dots f_1^{-1}(C_0)\dots) \cup \dots f_i^{-1}(C_{i-1}) \cup C_i)\,,\, f_i^{-1}(\dots f_1^{-1}(O_0)\dots))$$

and from Lemma 5.4 we have:

$$M_{i+1} \textbf{ sat } CA(C_{i+1}, O_{i+1})$$

Now for $M_i = M$ and $M_{i+1} = M'$ we define

$$C = (f_i^{-1}(\dots f_1^{-1}(C_0)\dots) \cup \dots f_i^{-1}(C_{i-1}) \cup C_i)$$
$$O = f_i^{-1}(\dots f_1^{-1}(O_0)\dots)$$
$$C' = C_{i+1}$$
$$O' = O_{i+1}$$

We observe from Lemma 6.1 that $O' = f_{i+1}^{-1}(C) \cup f_{i+1}^{-1}(O)$. It follows from Lemma 5.6 (with $X = \{\}$) that

$$M_{i+1} \textbf{ sat } CA(f_{i+1}^{-1}(f_i^{-1}(\dots f_1^{-1}(C_0)\dots) \cup \dots f_i^{-1}(C_{i-1}) \cup C_i) \cup C_{i+1}\,,\, f_{i+1}^{-1}(f_i^{-1}(\dots f_1^{-1}(O_0)\dots))),$$

i.e.,

$$M_{i+1} \textbf{ sat } CA(f_{i+1}^{-1}(\dots f_1^{-1}(C_0)\dots) \cup \dots f_{i+1}^{-1}(C_i) \cup C_{i+1}\,,\, f_{i+1}^{-1}(\dots f_1^{-1}(O_0)\dots))$$

establishing the case.

The result follows by induction. □

Finally, we would like to put together these results into one result relating the initial machine M_0 to the final

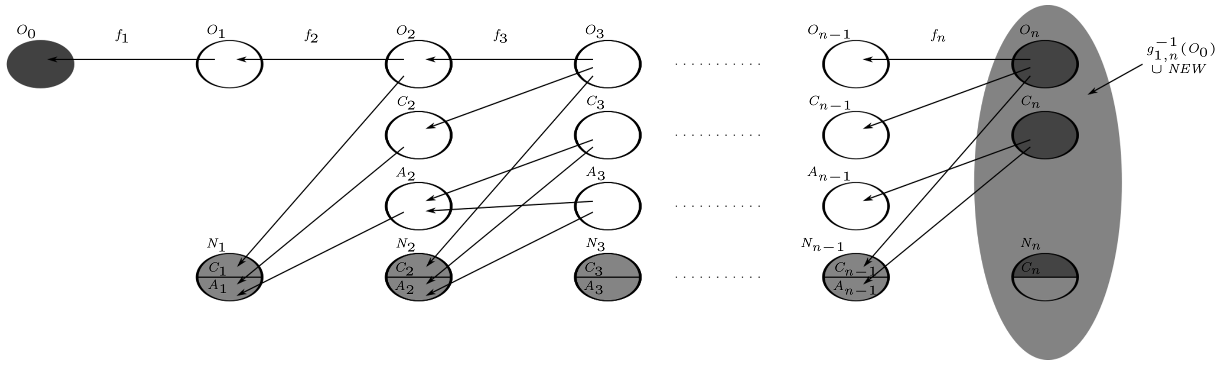


Fig. 9. Constructing *NEW*

machine M_n in the refinement chain. This result should use hiding for the treatment of new events, and — by stating the relationship between M_0 and $M_n \setminus \{new\ events\}$ via infinite-traces-divergences refinement — show that Event-B refinement actually does not introduce divergences on new events. This will be shown in Theorem 6.5 below. For such chains of refinement steps we have, by definition of strategy I, $A_0 = C_0 = \{\}$ (initially we have neither anticipated nor convergent events), and $A_n = \{\}$ (at the end all anticipated events have become convergent).

For this, we first of all need to find out what the "new events" are in the final machine. Define $g_{i,j}$ as the functional composition of the event mappings from f_j to f_i: (note that $g_{1,0}$ is the identity function)

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

Then by repeated application of

$$C_j \cup A_j \cup O_j = f_j^{-1}(C_{j-1} \cup A_{j-1} \cup O_{j-1}) \cup N_j$$

we obtain

$$C_j \cup A_j \cup O_j = g_{1,j}^{-1}(C_0 \cup A_0 \cup O_0) \cup g_{2,j}^{-1}(N_1) \cup \ldots \cup g_{j,j}^{-1}(N_{j-1}) \cup N_j \tag{2}$$

Also, by repeated application of

$$O_j = f_j^{-1}(O_{j-1}) \cup f_j^{-1}(C_{j-1})$$

we obtain

$$O_j \cup C_j = g_{1,j}^{-1}(O_0) \cup g_{1,j}^{-1}(C_0) \cup g_{2,j}^{-1}(C_1) \cup \ldots \cup g_{j,j}^{-1}(C_{j-1}) \cup C_j \tag{3}$$

In a full refinement chain $M_0 \preccurlyeq \ldots \preccurlyeq M_n$ we have, by the conditions of strategy I, that $A_0 = \{\}$,$C_0 = \{\}$, and $A_n = \{\}$.

Definition 6.3. The sets *NEW* and *CON* are defined as follows:

$$NEW = g_{2,n}^{-1}(N_1) \cup \ldots \cup g_{n,n}^{-1}(N_{n-1}) \cup N_n$$
$$CON = g_{2,n}^{-1}(C_1) \cup \ldots \cup g_{n,n}^{-1}(C_{n-1}) \cup C_n$$

These constructions are illustrated in Figures 9 and 10.

Lemma 6.4. *NEW* = *CON*

Proof From Equation 2 above with $j = n$, and using $A_0 = C_0 = A_n = \{\}$ we obtain

$$C_n \cup O_n = g_{1,n}^{-1}(O_0) \cup NEW$$

From Equation 3 above with $j = n$ we obtain

$$C_n \cup O_n = g_{1,n}^{-1}(O_0) \cup CON$$

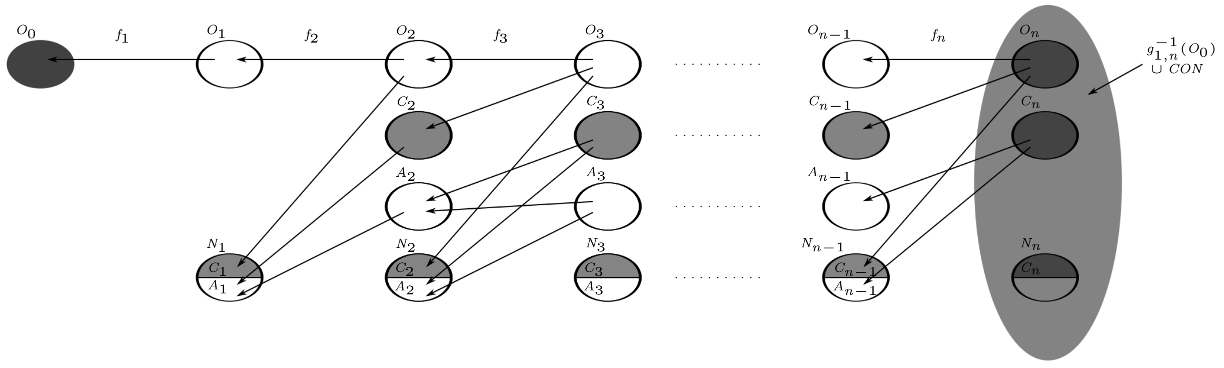


Fig. 10. Constructing *CON*

Observe that $g^{-1}_{1,n}(O_0) \cap NEW = \{\}$ and $g^{-1}_{1,n}(O_0) \cap CON = \{\}$, hence

$$(C_n \cup O_n) \setminus g^{-1}_{1,n}(O_0) = NEW = CON$$

which yields the result. □.

From Theorems 5.2 and 6.2 above respectively we obtain that

$$f_n^{-1}(\ldots(f_1^{-1}(M_0))\ldots) \;|||\; RUN_{NEW} \sqsubseteq_{TDI} M_n$$

$$\text{and} \quad M_n \ \mathbf{sat}\ CA(CON\ ,\ f_n^{-1}(\ldots f_1^{-1}(O_0)\ldots)\ )$$

Lemma 5.5(4) yields that $M_n \setminus CON$ is divergence-free, i.e., $M_n \setminus NEW$ is divergence-free by Lemma 6.4. Hence by Lemma 4.1 we obtain that

$$g^{-1}_{1,n}(M_0) \sqsubseteq_{TDI} M_n \setminus NEW \tag{4}$$

or, equivalently, that the following theorem holds true.

Theorem 6.5. Let $M_0 \preccurlyeq M_1 \preccurlyeq \ldots \preccurlyeq M_n$ be a chain of refinement steps obeying strategy I and refining events according to functions f_i, and let NEW be as defined in Definition 6.3. Then

$$M_0 \sqsubseteq_{TDI} g_{1,n}(M_n \setminus NEW)$$

Proof: This follows from the result in Line 4 above, using the CSP law $f(f^{-1}(P)) = P$. □

This result guarantees that Event-B refinement in strategy I neither introduces "new traces on old events", nor introduces divergences on new events. This gives us the precise account of Event-B refinement in terms of CSP.

Again, we make yet another extension of our example to demonstrate this result on a full refinement chain. Our final machine for strategy I shown in Figure 11 is $Basket4.I$. This machine is introducing a capacity for a basket and now allows adding only when the basket is not yet full. It furthermore makes event *add*, which has been anticipated so far, convergent. The corresponding variant to be shown to be decreased is $CAP - tot$. Note that $Basket4.I$ now has no anticipated events anymore and can thus serve as the final machine in our refinement chain. The function $f_{4.I}$ is the identity (no renaming or splitting of events).

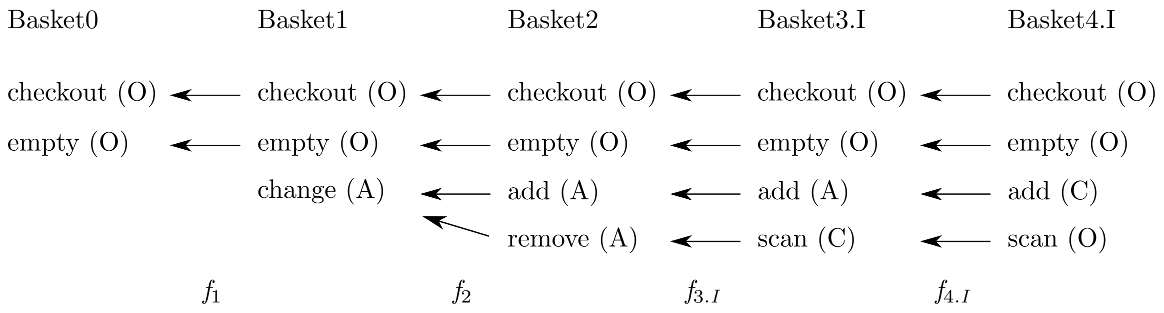
This completes the chain of refinements $Basket0 \preccurlyeq Basket1 \preccurlyeq Basket2 \preccurlyeq Basket3.I \preccurlyeq Basket4.I$. Figure 12 shows the events appearing in the machines together with their status. By Theorem 6.5 we get our final result relating the CSP semantics of the initial with the final machine:

$$Basket0 \sqsubseteq_{TDI} f_1(f_2(f_{3.I}(f_{4.I}(Basket4.I \setminus \{add, scan\}))))$$

Following the non-extensible interface approach all newly introduced events have been made convergent and thus can safely be hidden without introducing divergence in the CSP semantics.

```
machine Basket4.I
refines Basket3.I
variant $CAP - tot$
events
  init $\widehat{=}$
     $state := state\_empty \parallel scanning := false \parallel tot := 0$
  checkout $\widehat{=}$
     when $tot > 0$
     then $state := state\_complete \parallel scanning := true$ end
  empty $\widehat{=}$
     when $tot = 0$
     then $state := state\_empty \parallel scanning := false$ end
  add $\widehat{=}$
     status : convergent
     when $scanning \neq true \wedge tot < CAP$
     then $tot := tot + 1 \parallel state := state\_changing$ end
  scan $\widehat{=}$
     when $tot > 0 \wedge scanning = true$
     then $tot := tot - 1 \parallel state := state\_changing$ end
end
```

Fig. 11. Event-B machine *Basket4.I*



$$\begin{tikzcd}
\text{Basket0} & & \text{Basket1} & & \text{Basket2} & & \text{Basket3.I} & & \text{Basket4.I} \\
\text{checkout (O)} & & \text{checkout (O)} \arrow[ll] & & \text{checkout (O)} \arrow[ll] & & \text{checkout (O)} \arrow[ll] & & \text{checkout (O)} \arrow[ll] \\
\text{empty (O)} & & \text{empty (O)} \arrow[ll] & & \text{empty (O)} \arrow[ll] & & \text{empty (O)} \arrow[ll] & & \text{empty (O)} \arrow[ll] \\
 & & \text{change (A)} & & \text{add (A)} \arrow[ll] & & \text{add (A)} \arrow[ll] & & \text{add (C)} \arrow[ll] \\
 & & & & \text{remove (A)} \arrow[llu] & & \text{scan (C)} \arrow[ll] & & \text{scan (O)} \arrow[ll] \\
 & f_1 & & f_2 & & f_{3.I} & & f_{4.I} &
\end{tikzcd}$$

Fig. 12. Machines and events in the development according to strategy I

7. Strategy II: Interface Extension

In this section we will look at the second development strategy which allows the interface of an Event-B machine to be extended. The main difference to the first strategy is that (a) new events can immediately have status ordinary, and (b) not all anticipated events need to be made convergent, but can also be made ordinary. For a development $M_0 \preccurlyeq M_1 \preccurlyeq \ldots \preccurlyeq M_n$ we thus get the following rules:

1. in the initial machine all events are ordinary;
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 or ordinary;
4. each event in M_{i+1} which refines an anticipated event of M_i is itself either convergent or anticipated or ordinary;
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.

The first as well as the last condition is just added to make reasoning about refinement relationships clearer. Both conditions impose no real restrictions on the development: given a sequence of machines which do not adhere to condition 1 and 6, we can simply extend it with two new machines without getting additional proof obligations. The first condition can be met by introducing an extra initial machine with just the ordinary

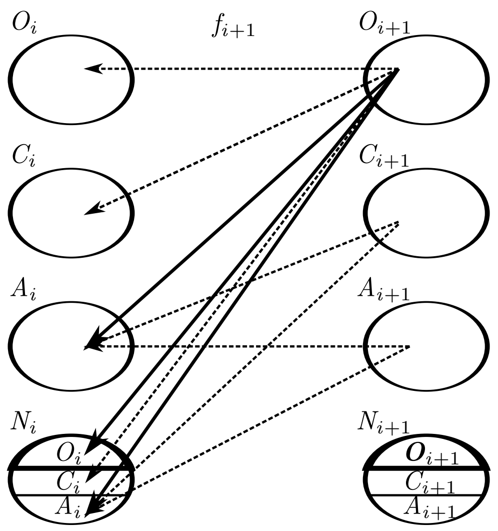


Fig. 13. Relationship between events in a refinement step in the extensible interface approach: f_{i+1} maps events in M_{i+1} to events in M_i that they refine. The differences with the non-extensible interface approach are highlighted in bold.

events of the (previously initial) machine, and then in a first refinement steps introduce all the (initially wanted) anticipated and convergent events; the last condition can be similarly achieved by introducing an extra final machine which makes all anticipated events ordinary. Note also that some conditions do not impose any restrictions on refinements; they are merely stated here for getting a closer correspondence to the rules of strategy I. Again, we formalise the conditions.

1. $A_0 = C_0 = \{\}$;
2. $ran(f_{i+1}) = A_i \cup C_i \cup O_i$;
3. $N_i = (A_i \cup C_i \cup O_i) \setminus dom(f_i)$;
4. $f_{i+1}^{-1}(A_i) \subseteq C_{i+1} \cup A_{i+1} \cup O_{i+1}$;
5. $f_{i+1}^{-1}(C_i \cup O_i) \subseteq O_{i+1}$;
6. $A_n = \{\}$.

These relationships between the events are illustrated in Fig. 13. We observe that strategy II is the more flexible one. At each step in the refinement it allows the addition of new events, which (a) are added to the interface (new ordinary events), or (b) are added to the set of internal events (new convergent events) or for which the decision is deferred to later steps (anticipated events).

To see how this kind of development fits into a CSP refinement, we again calculate two sets of events introduced in a development. However, they play a different role this time. The first set, called IF, is the set of events which will be added during development as *external* events. Together with the ordinary events of the initial machine they constitute the interface of the system. Events are considered external when they are never shown to be convergent. We get additional external events when we have new events being ordinary, or new events being anticipated and never made convergent in the subsequent refinement steps. IF can be incrementally defined:

Definition 7.1.

$$\begin{aligned}
IF_0 &= \{\} \\
IF_1 &= (f_1^{-1}(A_0) \cap O_1) \cup (N_1 \cap O_1) \\
IF_2 &= f_2^{-1}(IF_1) \cup (f_2^{-1}(A_1) \cap O_2) \cup (N_2 \cap O_2) \\
&\ \ldots \\
IF_j &= f_j^{-1}(IF_{j-1}) \cup (f_j^{-1}(A_{j-1}) \cap O_j) \cup (N_j \cap O_j)
\end{aligned}$$

Finally, $IF = IF_n$.

The second set of events is INT, the set of events added during development and considered *internal* in the final machine. Here, we find all those events which have been proven to be convergent in some of the development steps. Again, this set can be incrementally defined:

Definition 7.2.

$$\begin{aligned} INT_0 &= \{\} \\ INT_1 &= C_1 \\ INT_2 &= f_2^{-1}(INT_1) \cup C_2 \\ &\dots \\ INT_j &= f_j^{-1}(INT_{j-1}) \cup C_j \end{aligned}$$

Here, we get $INT = INT_n = g_{2,n}^{-1}(C_1) \cup g_{3,n}^{-1}(C_2) \cup \ldots \cup C_n$. The two sets of events IF and INT are disjoint.

Lemma 7.3. Let IF, INT be the extended interface set and the set of internal events, respectively, as constructed in Definitions 7.1 and 7.2. Then

$$IF \cap INT = \{\} \ .$$

Proof: We show this by induction on the number of steps in the development. More precisely, we show for every machine M_i, $0 \leq i \leq n$, that $INT_i \cap IF_i = \{\}$ (and $INT_i \cap A_i = \{\}$). As a first observation note that $IF_i \subseteq O_i$.

Induction base Initially, both IF_0 and INT_0 are empty as is A_0, thus all three sets are mutually disjoint.

Induction step Assume $INT_i \cap IF_i = \{\}$, $INT_i \cap A_i = \{\}$ and $IF_i \cap A_i = \{\}$.

- Proof of $INT_{i+1} \cap A_{i+1} = \{\}$. By definition $INT_{i+1} = f_{i+1}^{-1}(INT_i) \cup C_{i+1}$. The intersection of A_{i+1} and C_{i+1} is empty since every event can only have one status in a refinement step. Furthermore, $f_{i+1}^{-1}(INT_i) \cap A_{i+1} = \{\}$ since $INT_i \subseteq C_i \cup O_i$ and by rule 5 of the strategy none of these events can become anticipated.
- Proof of $IF_{i+1} \cap INT_{i+1} = \{\}$. We have

  $$\begin{aligned} IF_{i+1} &= f_{i+1}^{-1}(IF_i) \cup (f_{i+1}^{-1}(A_i) \cap O_{i+1}) \cup (N_{i+1} \cap O_{i+1}) \\ INT_{i+1} &= f_{i+1}^{-1}(INT_i) \cup C_{i+1} \end{aligned}$$

  For their intersection we look at several cases: (1) $f_{i+1}^{-1}(IF_i) \cap f_{i+1}^{-1}(INT_i) = \{\}$ follows from f_{i+1} being a function and the induction hypothesis $IF_i \cap INT_i = \{\}$; (2) $f_{i+1}^{-1}(IF_i) \cap C_{i+1} = \{\}$ since $IF_i \subseteq O_i$, $f_{i+1}^{-1}(O_i) \subseteq O_{i+1}$ (rule 5 of the strategy) and $O_{i+1} \cap C_{i+1} = \{\}$; (3) $f_{i+1}^{-1}(A_i) \cap O_{i+1} \cap C_{i+1} = \{\}$ since $O_{i+1} \cap C_{i+1} = \{\}$; (4) $f_{i+1}^{-1}(A_i) \cap O_{i+1} \cap f_{i+1}^{-1}(INT_i) = \{\}$ follows from $A_i \cap INT_i = \{\}$ and f_{i+1} being a function, and (5) $N_{i+1} \cap O_{i+1} \cap C_{i+1} = \{\}$ since the sets of ordinary and convergent events are disjoint, and finally (6) $N_{i+1} \cap O_{i+1} \cap f_{i+1}^{-1}(INT_i) = \{\}$ since new and existing events are always different. □

Note furthermore the event set NEW constructed in the last section is the set INT: all newly introduced events are internal in strategy I. The set IF would be empty in refinement chains of strategy I.

Next, we need to find out how this type of development fits into CSP refinement. We will see that the main difference to the first development strategy is that not all newly introduced events are made convergent but some are instead used to extend the interface. All convergent events can safely be hidden without introducing new divergences; all events in the interface extension can however only be tackled by the RUN process. Our relationship between the initial and final machine will thus take the form (renamings omitted):

$$M_0 \mid\mid\mid RUN_{IF} \sqsubseteq_{TDI} M_n \setminus INT \ .$$

Again, we first of all consider just two machines M_0 and M_1. Our first step is a generalisation of Lemma 4.1, which allows to split the events over the RUN process into two sets, one of which is kept in RUN and the other moved to the right hand side process and hidden.

Lemma 7.4. IF $P_0 \mid\mid\mid RUN_{N_1 \cup N_2} \sqsubseteq_{TDI} P_1$ and $N_1 \cap N_2 = \{\}$ and $(N_1 \cup N_2) \cap \alpha P_0 = \{\}$ and $P_1 \setminus N_2$ divergence-free, then $P_0 \mid\mid\mid RUN_{N_1} \sqsubseteq_{TDI} P_1 \setminus N_2$.

Proof: Assume that (1) $P_0 \mid\mid\mid RUN_{N_1 \cup N_2} \sqsubseteq_{TDI} P_1$ and (2) $N_1 \cap N_2 = \{\}$ and (3) $(N_1 \cup N_2) \cap \alpha P_0 = \{\}$ and (4) $P_1 \setminus N_2$ divergence-free.

Traces Let $tr \in traces(P_1 \setminus N_2)$. By semantics of hiding there is some $tr' \in traces(P_1)$ such that $tr' \setminus N_2 = tr$. By (1) $tr' \in traces(P_0 \mid\mid\mid RUN_{N_1 \cup N_2})$. By (2) and (3) $tr' \setminus N_2 = tr \in traces(P_0 \mid\mid\mid RUN_{N_1})$.

Divergences By (3) $divergences(P_1 \setminus N_2) = \{\}$, thus nothing to be proven here.

Infinites Let $u \in infinites(P_1 \setminus N_2)$. By semantics of hiding there is some $u' \in infinites(P_1)$ such that $u' \setminus N_2 = u$ and $\#(u' \setminus N_2) = \infty$. By (1) $u' \in infinites(P_0 \mid\mid\mid RUN_{N_1 \cup N_2})$. By (2) and (3) $u' \setminus N_2 = u \in infinites(P_0 \mid\mid\mid RUN_{N_1})$.

$\square$

This result can be used in a way similar to the usage of Lemma 4.1.

Lemma 7.5. Let $M_0 \preccurlyeq M_1$ with events refined according to function f_1. Then

$$f_1^{-1}(M_0) \mid\mid\mid RUN_{N_1 \cap (A_1 \cup O_1)} \sqsubseteq_{TDI} M_1 \setminus (N_1 \cap C_1)$$

Proof: By Lemmas 5.5 (4) and 5.4 we get $M_1 \setminus C_1$ divergence-free. By Lemma 5.1 we furthermore have $f_1^{-1}(M_0) \mid\mid RUN_{N_1} \sqsubseteq_{TDI} M_1$. We now distribute N_1 into disjoint partitions $N_1 \cap (A_1 \cup O_1)$ and $N_1 \cap C_1$ and can now apply Lemma 7.4 which gives us the result.

$\square$

Again similar to the previous section, we need a result about the CA-predicate in refinement chains constructed according to strategy II. The following lemma can be seen as the generalisation of Lemma 5.6 to strategy II. This time we have to take into account that anticipated events can be refined straight into ordinary events during a refinement step, and also ordinary events can be introduced as new events in a refinement step. Hence the O' referred to in the CA predicate which the refinement machine needs to satisfy must also take these new sets into account.

Lemma 7.6. Let $M \preccurlyeq M'$ be constructed according to strategy II with an associated refinement function f' and let M **sat** $CA(C, O)$. Then M' **sat** $CA(f'^{-1}(C) \cup C'\ ,\ f'^{-1}(O) \cup (f'^{-1}(A) \cap O') \cup (N' \cap O'))$.

Proof: Assume $u \in infinites(M')$ and $\#(u \upharpoonright (f'^{-1}(C) \cup C')) = \infty$. We aim to establish that $\#(u \upharpoonright f'^{-1}(O) \cup (f'^{-1}(A) \cap O') \cup (N' \cap O')) = \infty$. We have $\#(u \upharpoonright f'^{-1}(C)) = \infty$ or $\#(u \upharpoonright C') = \infty$.

In the former case, Lemma 5.1 yields that $f'(u \upharpoonright f^{-1}(\alpha M)) \in infinites(M)$. Then

$$\begin{array}{rl}
\#(u \upharpoonright f'^{-1}(C)) = \infty & \text{(given)} \\
\#(f'(u \upharpoonright f'^{-1}(C)) \upharpoonright C) = \infty & \text{(since renaming preserves length)} \\
\#(f'(u \upharpoonright f'^{-1}(\alpha M)) \upharpoonright C) = \infty & \text{(since } C \subseteq \alpha M) \\
\#(f'(u \upharpoonright f'^{-1}(\alpha M)) \upharpoonright O) = \infty & \text{(by } M \textbf{ sat } CA(C, O)) \\
\#(u \upharpoonright f'^{-1}(\alpha M)) \upharpoonright f'^{-1}(O) = \infty & \text{(since renaming preserves length)} \\
\#(u \upharpoonright f'^{-1}(O)) = \infty & \text{(since } O \subseteq \alpha M)
\end{array}$$

In the latter case Lemma 5.4 yields that $\#(u \upharpoonright O') = \infty$. Then we have $\#(u \upharpoonright f'^{-1}(C \cup O) \cup (f'^{-1}(A) \cap O') \cup (N' \cap O')) = \infty$ (since O' is by definition of strategy II $f'^{-1}(C \cup O) \cup (f'^{-1}(A) \cap O') \cup (N' \cap O')$). Hence $\#(u \upharpoonright f'^{-1}(C)) = \infty$ or $\#(u \upharpoonright f'^{-1}(O)) = \infty$ or $\#(u \upharpoonright f'^{-1}(A) \cap O') = \infty$ or $\#(u \upharpoonright (N' \cap O')) = \infty$. The second, third and fourth cases make up the desired result, the first is the case already treated above.

$\square$

For our next theorem we first require a lemma:

Lemma 7.7. If $M_0 \preccurlyeq M_1 \preccurlyeq \ldots \preccurlyeq M_n$ then for each $i \leqslant n$:

$$O_i = f_i^{-1}(INT_{i-1}) \cup IF_i \cup g_{1,i}^{-1}(O_0)$$

Proof: By induction on n.

Induction base Assume $n = 0$. This case reduces to $O_0 = O_0$, which establishes the case.

Induction step Assume the result for i, we aim to establish it for $i+1$. The inductive hypothesis gives:

$$O_i = f_i^{-1}(INT_{i-1}) \cup IF_i \cup g_{1,i}^{-1}(O_0)$$

We have

$$\begin{aligned}
O_{i+1} &= (f_{i+1}^{-1}(O_i \cup C_i \cup A_i) \cap O_{i+1}) \cup (N_i \cap O_i) \\
&= f_{i+1}^{-1}(O_i) \cup f_{i+1}^{-1}(C_i) \cup f_{i+1}^{-1}(A_i) \cap O_{i+1} \cup (N_{i+1} \cap O_{i+1}) \\
&= f_{i+1}^{-1}(f_i^{-1}(INT_{i-1}) \cup IF_i \cup g_{1,i}^{-1}(O_0)) \cup f_{i+1}^{-1}(C_i) \cup f_{i+1}^{-1}(A_i) \cap O_{i+1} \cup (N_{i+1} \cap O_{i+1}) \\
&= f_{i+1}^{-1}((f_i^{-1}(INT_{i-1}) \cup C_i) \cup (f_{i+1}^{-1}(IF_i) \cup f_{i+1}^{-1}(A_i) \cap O_{i+1} \cup (N_{i+1} \cap O_{i+1}) \cup (f_{i+1}^{-1}(g_{1,i}^{-1}(O_0))) \\
&= f_{i+1}^{-1}(INT_i) \cup IF_{i+1} \cup g_{1,i+1}^{-1}(O_0)
\end{aligned}$$

which establishes the case. □

We now obtain the following result on refinement chains:

Theorem 7.8. If $M_0 \preccurlyeq M_1 \preccurlyeq \ldots \preccurlyeq M_n$ then

$$M_n \ \mathbf{sat}\ CA(INT_n\ ,\ g_{1,n}^{-1}(O_0) \cup IF_n)$$

Proof: By induction on n.

Induction base Assume $n = 0$. In this case $INT_0 = \{\}$, and $g_{1,0}^{-1}(O_0) \cup IF_0 = O_0$. The result immediately follows since $g_{1,0}$ is the identify, and M_0 **sat** $CA(\{\}, O_0)$ holds vacuously, establishing the case.

Induction step Assume the result for i, we aim to establish it for $i+1$. The inductive hypothesis gives:

$$M_i \ \mathbf{sat}\ CA(INT_i, g_{1,i}^{-1}(O_0) \cup IF_i)$$

and from Lemma 5.4 we have:

$$M_{i+1} \ \mathbf{sat}\ CA(C_{i+1}, O_{i+1})$$

Now for $M_i = M$ and $M_{i+1} = M'$ we define

$$\begin{aligned}
C &= INT_i \\
O &= g_{1,i}^{-1}(O_0) \cup IF_i \\
C' &= C_{i+1} \\
O' &= O_{i+1} \\
X &= (f_{i+1}^{-1}(A_i) \cap O_{i+1}) \cup (N_{i+1} \cap O_{i+1})
\end{aligned}$$

Now

$$\begin{aligned}
O' &= O_{i+1} \\
&= (f_{i+1}^{-1}(O_i \cup C_i \cup A_i) \cap O_{i+1}) \cup (N_{i+1} \cap O_{i+1}) \\
&= f_{i+1}^{-1}(O_i) \cup f_{i+1}^{-1}(C_i) \cup ((f_{i+1}^{-1}(A_i) \cap O_{i+1}) \cup (N_{i+1} \cap O_{i+1})) \\
&= f_{i+1}^{-1}(f_i^{-1}(INT_{i-1}) \cup IF_i \cup g_{1,i}^{-1}(O_0)) \cup f_{i+1}^{-1}(C_i) \cup X && \text{(by Lemma 7.7)} \\
&= f_{i+1}^{-1}(f_i^{-1}(INT_{i-1}) \cup f_{i+1}^{-1}(IF_i) \cup f_{i+1}^{-1}(g_{1,i}^{-1}(O_0)) \cup f_{i+1}^{-1}(C_i) \cup X \\
&= f_{i+1}^{-1}(f_i^{-1}(INT_{i-1})) \cup f_{i+1}^{-1}(C_i) \cup f_{i+1}^{-1}(IF_i) \cup f_{i+1}^{-1}(g_{1,i}^{-1}(O_0)) \cup X && \text{(reordering terms)} \\
&= f_{i+1}^{-1}(INT_i) \cup f_{i+1}^{-1}(IF_i \cup (g_{1,i}^{-1}(O_0))) \cup X \\
&= f_{i+1}^{-1}(C) \cup f_{i+1}^{-1}(O) \cup X
\end{aligned}$$

It follows from Lemma 5.6 that

$$M_{i+1} \ \mathbf{sat}\ CA(f_{i+1}^{-1}(INT_i) \cup C_{i+1}\ ,\ f_{i+1}^{-1}(g_{1,i}^{-1}(O_0) \cup IF_i) \cup (f_{i+1}^{-1}(A_i) \cap O_{i+1}) \cup (N_{i+1} \cap O_{i+1}))$$

By the definitions of INT_{i+1} and IF_{i+1} this reduces to

$$M_i \ \mathbf{sat}\ CA(INT_{i+1}, g_{1,i+1}^{-1}(O_0) \cup IF_{i+1})$$

```
machine $Basket3.II$
refines $Basket2$
variables $tot, state, scanning$
invariant $scanning \in true, false$
variant $if\ scanning = true\ then\ tot\ else\ 0$
events
  init $\widehat{=}$
     $state := state\_empty \mid\mid scanning := false \mid\mid tot := 0$   checkout $\widehat{=}$
      when $tot > 0$
      then $state := state\_complete \mid\mid scanning := true$ end
  empty $\widehat{=}$
      when $tot = 0$
      then $state := state\_empty \mid\mid scanning := false$ end
  add $\widehat{=}$
      status : ordinary
      when $scanning \neq true$
      then $tot := tot + 1 \mid\mid state := state\_changing$ end
  remove $\widehat{=}$
      status : ordinary
      when $tot > 0 \wedge scanning \neq true$
      then $tot := tot - 1 \mid\mid state := state\_changing$ end
  scan $\widehat{=}$
      status : convergent
      refines $remove$
      when $tot > 0 \wedge scanning = true$
      then $tot := tot - 1 \mid\mid state := state\_changing$ end
end
```

Fig. 14. Alternative development of the basket following approach 2

which establishes the case.

The result follows by induction. □

Since $INT \cap (g_{2,n}(O_0) \cup IF) = \{\}$ we furthermore get $M_n \setminus INT$ divergence-free. Together with Lemma 7.4 we finally obtain

Theorem 7.9. Let $M_0 \preccurlyeq M_1 \preccurlyeq \ldots \preccurlyeq M_n$ be a chain of refinement steps following strategy II, refining events according to functions f_i, and let IF and INT be the two sets calculated above. Then

$$g_{1,n}^{-1}(M_0) \mid\mid\mid RUN_{IF} \sqsubseteq_{TDI} M_n \setminus INT$$

In Figure 14 we see an alternative development of $Basket2$ following this second development strategy. Instead of making the anticipated event *add* of $Basket2$ convergent (or keep it anticipated), we put it into the interface and make it ordinary. The event *remove* of $Basket2$ is now split into an ordinary event *remove* and a convergent event *scan*. The machine thus ends up with having an interface $\{checkout, empty, add, remove\}$ while having only one internal event which is *scan*. Thus for the development $Basket0 \preccurlyeq Basket1 \preccurlyeq Basket2 \preccurlyeq Basket3.II$ with event sets and event status as shown in Figure 15 we obtain $IF = \{add, remove\}$ (the extension of the interface) and $INT = \{scan\}$, and thus by Theorem 7.9 we get

$$f_{3.II}^{-1}(f_2^{-1}(f_1^{-1}(Basket0))) \mid\mid\mid RUN_{\{add, remove\}} \sqsubseteq_{TDI} Basket3.II \setminus \{scan\}$$

For example, one trace of $Basket3.II \setminus \{scan\}$ is the trace

$$\langle add, add, remove, checkout, scan, empty\rangle \setminus \{scan\} = \langle add, add, remove, checkout, empty\rangle$$

This can be separated out into

$$\langle checkout, empty\rangle \in traces(f_{3.II}^{-1}(f_2^{-1}(f_1^{-1}(Basket0))))$$

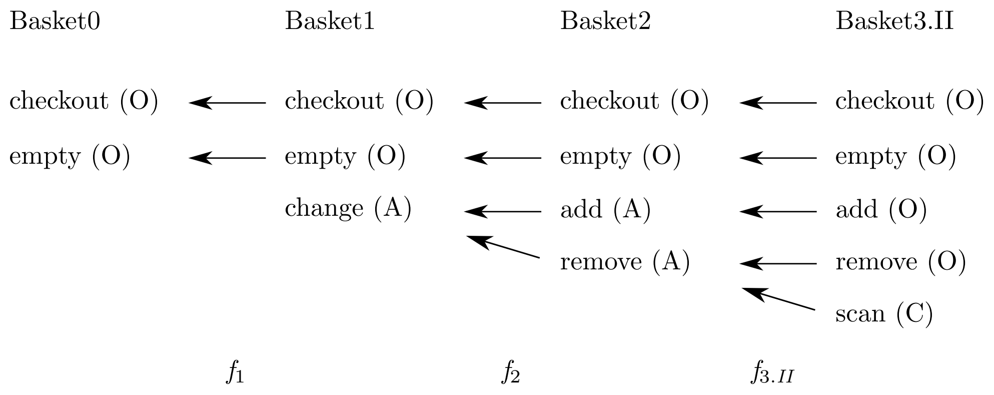


Fig. 15. Machines and events in the development according to strategy II

and

$\langle add, add, remove\rangle \in traces(RUN_{\{add,remove\}})$

This shows that the trace is also a trace of the interleaving of these two component processes, $f_{3.II}^{-1}(f_2^{-1}(f_1^{-1}(Basket0))) \mid\mid\mid RUN_{\{add,remove\}}$, in line with Theorem 7.9.

This completes the basket example for which we now have two different developments and the corresponding results in terms of CSP refinement.

8. Conclusion

In this paper, we have given a behavioural semantics account of Event-B refinement. The approach builds on Butler's semantics for action systems [But92]. Butler's refinement rules allow new convergent events to be introduced into action systems, so that a refinement step from machine M_i to machine M_{i+1}, introducing new events N_{i+1}, satisfies $M_i \sqsubseteq_{TDI} (M_{i+1} \setminus N_{i+1})$, thus ensuring that hiding the new events does not introduce divergence. Abrial's approach to Event-B refinement [Abr10] generalises this approach, allowing new events to be *anticipated* (deferring making it internal) as well as *convergent* (to be considered as internal) , and also allowing splitting of events. The approach requires that anticipated events become convergent at some refinement step, so that all events introduced during refinement are shown at some point to be convergent. This corresponds to strategy I described in Section 6. The Rodin tool for Event-B [BH07, EB11, ABH$^+$10] generalises Abrial's approach in supporting a more liberal treatment of anticipated events, and does not require that they are eventually made convergent but allows them to be added to the interface instead. In this approach treatment of an event as anticipated therefore defers the decision as to whether it will ultimately be internal or external. This is strategy II as described in Section 7. Rodin does have some superficial differences with strategy II, notably that it allows refinements of convergent events to also be labelled as convergent, whereas strategy II requires them to be ordinary. In fact they are treated as ordinary by Rodin, since the proof obligations generated do not require them to decrease the variant. Rodin allows such events to be labelled as convergent to allow the developer to record that they were shown to be convergent at some stage.

Our approach to refinement using CSP semantics reflects the introduction of anticipated events and splitting, and thus extends Butler's approach, in order to encompass these different forms of event treatment in Event-B refinement. We obtain a clear statement of the relationship between the first and the last machines in a refinement chain, which follows from the transitivity of refinement along the chain, together with the additional requirements on the first and last machines: that the first should contain only ordinary events, and that the last should not contain any anticipated events. For strategy I the relationship is given by Theorem 6.5, and for strategy II by Theorem 7.9. We do not yet handle merging events, and this is the subject of current research. The handling of divergence in refinement, though not in an Event-B setting, has also been investigated in [BD09]. Proofs of divergence freedom for particular events also come into play when studying liveness properties of Event-B machines. This has recently been investigated in [HA11], looking at properties specified in temporal logic.

Recently, an Event-B$\|$CSP approach has been introduced [STW10]. It aims to combine Event-B machine descriptions with CSP [Sch99] control processes, in order to support a more explicit view of control. In this, it follows previous works on integration of formal methods [But00, WC02, DS03, OW05, ST05, Ili09], which aim at complementing a state-based specification formalism with a process algebra.

The account of refinement presented here provides the basis for a flexible refinement framework in Event-B$\|$CSP, and this is presented in [STW11c]. The semantics justifies the introduction of a new status of *devolved*, for refinement events which are anticipated in the Event-B machine but convergent in the CSP controller. This approach has been applied to an initial Event-B$\|$CSP case study of a Bounded Retransmission Protocol [STW11a]. We aim to investigate further case studies. We are in particular interested in finding out whether the work of showing divergence-freedom (and also deadlock-freedom) can be divided onto the Event-B and CSP part such that for some events convergence is guaranteed by showing the corresponding proof obligations in Event-B while for others we just look at divergence-freedom of the CSP process. The latter part could then be supported by model checking tools for CSP, like FDR [For]. Another strand of future work would be an investigation of *decomposition* techniques for Event-B [But09, HA10, SHWI11] and their impact on the CSP semantics.

Acknowledgement. Many thanks to Thai Son Hoang for numerous discussions on Event-B development strategies. Thanks also to the anonymous referees for their careful reading of the paper and for their constructive suggestions.

References

[ABH$^+$10] 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.

[ABHV08] J-R. Abrial, M. J. Butler, S. Hallerstede, and L. Voisin. A Roadmap for the Rodin Toolset. In E. Börger, M. J. Butler, J. P. Bowen, and P. Boca, editors, *ABZ*, volume 5238 of *Lecture Notes in Computer Science*, page 347. Springer, 2008.

[Abr05] Jean-Raymond Abrial. *The B-book - assigning programs to meanings*. Cambridge University Press, 2005.

[Abr10] J-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[BD02] C. Bolton and J. Davies. Refinement in Object-Z and CSP. In M. Butler, L. Petre, and K. Sere, editors, *IFM 2002: Integrated Formal Methods*, number 2335 in LNCS, pages 225–244, 2002.

[BD09] E. A. Boiten and J. Derrick. Modelling divergence in relational concurrent refinement. In Leuschel and Wehrheim [LW09], pages 183–199.

[BH07] M.J. Butler and S. Hallerstede. The Rodin formal modelling tool. In *BCS-FACS Christmas 2007 Meeting — Formal Methods In Industry*, 2007.

[But92] M. J. Butler. *A CSP approach to Action Systems*. DPhil thesis, Oxford University, 1992.

[But00] M. J. Butler. csp2B: A practical approach to combining CSP and B. In *FACS*, pages 182–196, 2000.

[But09] M. J. Butler. Decomposition Structures for Event-B. In Leuschel and Wehrheim [LW09], pages 20–38.

[But12] Michael Butler. External and internal choice with event groups in event-b. *Formal Aspects of Computing*, 24(4-6):555–567, 2012.

[BvW98] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus: A Systematic Introduction*. Graduate Texts in Computer Science. Springer, 1998.

[DB01] J. Derrick and E.A. Boiten. *Refinement in Z and Object-Z*. Springer-Verlag, 2001.

[DB03] J. Derrick and E.A. Boiten. Relational concurrent refinement. *Formal Aspects of Computing*, 15(2-3):182–214, November 2003.

[DS03] J. Derrick and G. Smith. Structural Refinement of Systems Specified in Object-Z and CSP. *Formal Asp. Comput.*, 15(1):1–27, 2003.

[EB11] Event-B.org. Rodin platform version 2.2.2, release date 2011/06/01. `http://www.event-b.org/`, 2011.

[For] Formal Systems (Europe) Ltd. The FDR model checker. `http://www.fsel.com/` (accessed 8/3/11).

[HA10] T.S. Hoang and J-R. Abrial. Event-B Decomposition for Parallel Programs. In M. Frappier, U. Glässer, S. Khurshid, R. Laleau, and S. Reeves, editors, *ABZ*, volume 5977 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2010.

[HA11] T.S. Hoang and J-R. Abrial. Reasoning about liveness properties in Event-B. In Shengchao Qin and Zongyan Qiu, editors, *ICFEM*, volume 6991 of *Lecture Notes in Computer Science*, pages 456–471. Springer, 2011.

[Hal11] Stefan Hallerstede. On the purpose of event-b proof obligations. *Formal Asp. Comput.*, 23(1):133–150, 2011.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Ili09] A. Iliasov. On Event-B and Control Flow. Technical Report CS-TR-1159, School of Computing Science, Newcastle University, August 2009.

[Jac02] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.

[LW09] M. Leuschel and H. Wehrheim, editors. *Integrated Formal Methods, 7th International Conference, IFM 2009, Düsseldorf, Germany, February 16-19, 2009. Proceedings*, volume 5423 of *Lecture Notes in Computer Science*. Springer, 2009.

[MAV05] C. Métayer, J.-R. Abrial, and L. Voisin. Event-B language, 2005. RODIN Project Deliverable 3.2, `http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf`, accessed 25/5/10.

[Mor88] C.C. Morgan. The specification statement. *ACM Trans. Program. Lang. Syst.*, 10(3):403–419, 1988.

[Mor90] C.C. Morgan. Of wp and CSP. *Beauty is our business: a birthday salute to E. W. Dijkstra*, pages 319–326, 1990.

[OW05] E-R. Olderog and H. Wehrheim. Specification and (property) inheritance in CSP-OZ. *Sci. Comput. Program.*, 55(1-3):227–257, 2005.

[Ros98] A.W. Roscoe. *Theory and Practice of Concurrency.* Prentice-Hall, 1998.

[Sch99] S. Schneider. *Concurrent and Real-time Systems: The CSP approach.* Wiley, 1999.

[SHWI11] R.A. Silva, T.S. Hoang, W. Wei, and A. Iliasov. A Survey on Event-B Decomposition. In *Workshop on Automated Verification of Critical Systems (AVOCS 2011)*, 2011.

[ST05] S. Schneider and H. Treharne. CSP theorems for communicating B machines. *Formal Asp. Comput.*, 17(4):390–422, 2005.

[STW10] S. Schneider, H. Treharne, and H. Wehrheim. A CSP approach to control in Event-B. In *IFM*, pages 260–274, 2010.

[STW11a] S. Schneider, H. Treharne, and H. Wehrheim. Bounded retransmission in Event-B‖CSP: a case study. In *Workshop B 2011*, ENTCS, 2011.

[STW11b] S. Schneider, H. Treharne, and H. Wehrheim. A CSP account of Event-B refinement. In J. Derrick, E.A. Boiten, and S. Reeves, editors, *Refine 2011*, volume 55 of *EPTCS*, pages 139–154, 2011.

[STW11c] S. Schneider, H. Treharne, and H. Wehrheim. Stepwise refinement in Event-B‖CSP. Technical Report CS-11-03, University of Surrey, 2011.

[WC02] J. Woodcock and A. Cavalcanti. The Semantics of Circus. In D. Bert, J. P. Bowen, M. C. Henson, and K. Robinson, editors, *ZB 2002*, volume 2272 of *Lecture Notes in Computer Science*, pages 184–203. Springer, 2002.

[WD96] J. C. P. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Prentice Hall, 1996.