Stepwise Refinement in Event-B||CSP
Part 1: Safety

Steve Schneider, Helen Treharne and Heike Wehrheim

March 12\textsuperscript{th} 2011
Stepwise Refinement in Event-B∥CSP
Part 1: Safety

Steve Schneider¹   Helen Treharne¹   Heike Wehrheim²

¹ Department of Computing, University of Surrey
² Institut für Informatik, Universität Paderborn

March 17, 2011

Contents

1 Introduction 3

2 CSP 3
   2.1 Notation ................................. 3
   2.2 Semantic Models .......................... 3
   2.3 Refinement and specification .......... 4
   2.4 Relevant CSP semantics ............... 5
       2.4.1 RUN .................................. 5
       2.4.2 Parallel composition .......... 5
       2.4.3 Hiding ............................. 6
       2.4.4 Renaming ......................... 6
   2.5 Lazy abstraction ...................... 7

3 Event-B 7
   3.1 Machines ............................... 7
   3.2 Events ................................. 8
   3.3 Proof obligations for refinement .... 9

4 CSP semantics for B machines 11
Abstract

This technical report provides the CSP semantic basis for stepwise refinement in Event-B||CSP. It provides the foundation for combining Event-B machines with CSP control processes in the context of refinement. A number of proof rules are presented which are sufficient to establish refinement of an Event-B||CSP combination. This report focuses on traces, both finite and infinite, which allows consideration of safety specifications and also consideration of divergence-freedom. Several refinement steps in Event-B||CSP in the development of a simple bounded retransmission protocol are presented to illustrate the approach.
1 Introduction

Event-B [Abr10] provides a framework for system development through stepwise refinement. Individual refinement steps 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 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 Event-B∥CSP approach aims to combine Event-B machine descriptions with CSP control processes, in order to support a more explicit view of control. This approach is founded on the CSP semantics for action systems, applied to Event-B. CSP also supports an approach to refinement consistent with that of Event-B. The aim of this report is to provide the underlying results to support refinement in Event-B∥CSP. This involves support for individual refinement steps, and results about the resulting refinement chain. We provide results for reasoning about deadlock-freedom, trace refinement, divergence-freedom, and failures refinement. The approach remains faithful to the Event-B approach, which uses ‘anticipated’ events to defer consideration of divergence-freedom. We introduce a way of dealing with the intermediate refinement steps so that the final level of the refinement chain is ensured to be divergence-free.

2 CSP

2.1 Notation

We use $tr$ to refer to finite traces: finite sequence of events. These can also be written explicitly as $(a_1, a_2, \ldots, a_n)$. The empty trace is written $\langle \rangle$. Concatenation of traces is written as $tr_1 \triangleright tr_2$. We use $u$ to refer to infinite traces. Given a set of events $A$, the projections $tr \upharpoonleft A$ and $u \upharpoonleft A$ are the traces restricted to only those events in $A$. Note that $u \upharpoonleft A$ might be finite, if only finitely many $A$ events appear in $u$. Conversely, $tr \upharpoonright A$ and $u \upharpoonright A$ 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. The set $A^*$ is the set of all finite sequences of elements of $A$, and $A^\omega$ is the set of all infinite sequences of elements of $A$.

We use $P$ and $Q$ to refer to CSP processes, and $M$ to refer to Event-B machines.

2.2 Semantic Models

A CSP process $P$ has an alphabet $\alpha P$. 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 consists of four sets \( \langle T, F, D, I \rangle \) which are respectively the 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 \( \alpha P \) that it can engage in.

Traces are finite sequences of events from \( P \)'s alphabet: \( tr \in \alpha P^* \). The set \( \text{traces}(P) \) represents the possible finite sequences of events that \( P \) can perform.

Failures are pairs \( (tr, X) \) consisting of a trace \( tr \) and a set \( X \subseteq \alpha P \). This describes \( P \) performing the sequence of events \( tr \) and then refusing to engage in any of the events in \( X \). The set \( \text{failures}(P) \) is the set of all such pairs corresponding to possible executions of \( P \). For technical reasons it also contains any pair \( (tr, X) \) for which \( tr \) is a divergence.

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. The set \( \text{divergences}(P) \) is the set of all possible divergences for \( P \). In this paper we are generally dealing with divergence-free processes, for which the set \( \text{divergences}(P) \) is empty.

Infinite traces \( u \in \alpha P^\omega \) are infinite sequences of events. The set \( \text{infinites}(P) \) is the set of infinite traces that \( P \) can exhibit. For technical reasons it also contains those infinite traces which have some prefix which is a divergence.

The set of finite and infinite traces of a process is denoted \( t_{\text{inf}}(P) \):

\[
t_{\text{inf}}(P) = \text{traces}(P) \cup \text{infinites}(P)
\]

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

### 2.3 Refinement and specification

In this paper we will use the following refinement relations:

\[
P \sqsubseteq_T Q \iff \text{traces}(Q) \subseteq \text{traces}(P)
\]

\[
P \sqsubseteq_{TDI} Q \iff \text{traces}(Q) \subseteq \text{traces}(P)
\]

\[
\land \text{divergences}(Q) \subseteq \text{divergences}(P)
\]

\[
\land \text{infinites}(Q) \subseteq \text{infinites}(P)
\]

The refinement relation may be used in specification: desired behaviour is expressed as a process \( \text{SPEC} \), and then a requirement on an implementation \( \text{IMP} \) is that \( \text{SPEC} \sqsubseteq \text{IMP} \) in the appropriate semantic model.

Specifications may also be given in terms of predicates. If \( S \) is a predicate on a particular semantic element, then we write \( P \text{ sat } S \) to denote that all relevant elements in the semantics of \( P \) meet the predicate \( S \). For example, if \( S(u) \) is a predicate on infinite traces, then \( P \text{ sat } S(u) \) is equivalent to \( \forall u \in \text{infinites}(P) . S(u) \).
2.4 Relevant CSP semantics

2.4.1 RUN

For a set of events $A$, the process $RUN_A$ is given as follows:

$$\alpha(RUN_A) = A$$

$$\text{traces}(RUN_A) = A^*$$

$$\text{failures}(RUN_A) = \{(tr, \{\} \mid tr \in A^*)\}$$

$$\text{divergences}(RUN_A) = \{\}$$

$$\text{infinites}(RUN_A) = A^\omega$$

2.4.2 Parallel composition

If $P$ has alphabet $\alpha P$ and $Q$ has alphabet $\alpha Q$ then the semantics of parallel composition can be given as follows:

$$\alpha(P \parallel Q) = \alpha P \cup \alpha Q$$

$$\text{traces}(P \parallel Q) = \{tr \mid tr \in (\alpha P \cup \alpha Q)^* \wedge tr \mid \alpha P \in \text{traces}(P) \wedge tr \mid \alpha Q \in \text{traces}(Q)\} \cup \text{divergences}(P \parallel Q)$$

$$\text{failures}(P \parallel Q) = \{(tr, X) \mid tr \in (\alpha P \cup \alpha Q)^* \wedge \exists X_P, X_Q . X_P \cup X_Q = X \wedge (tr \mid \alpha P, X_P) \in \text{failures}(P) \wedge (tr \mid \alpha Q, X_Q) \in \text{failures}(Q)\} \cup \{(tr, X) \mid tr \in \text{divergences}(P \parallel Q)\}$$

$$\text{divergences}(P \parallel Q) = \{tr \triangleright tr' \mid tr \in (\alpha P \cup \alpha Q)^* \wedge tr' \in (\alpha P \cup \alpha Q)^* \wedge (tr \mid \alpha P \in \text{traces}(P) \wedge tr \mid \alpha Q \in \text{divergences}(Q)) \vee (tr \mid \alpha P \in \text{divergences}(P) \wedge tr \mid \alpha Q \in \text{traces}(Q))\}$$

$$\text{infinites}(P \parallel Q) = \{u \mid u \in (\alpha P \cup \alpha Q)^\omega \wedge u \mid \alpha P \in \text{infinites}(P) \cup \text{traces}(P) \wedge u \mid \alpha Q \in \text{infinites}(Q) \cup \text{traces}(Q)\} \cup \{tr \triangleright u \mid tr \in \text{divergences}(P \parallel Q) \wedge u \in (\alpha P \cup \alpha Q)^\omega\}$$
2.4.3 Hiding

For \( A \subseteq \alpha P \), the hiding operator \( P \setminus A \) is defined as follows:

\[
\alpha(P \setminus A) = \alpha P - A
\]

\[
\text{traces}(P \setminus A) = \{ tr \setminus A \mid tr \in \text{traces}(P) \}
\]

\[
\text{failures}(P \setminus A) = \{(tr \setminus A, X) \mid (tr, X \cup A) \in \text{failures}(P) \}
\]

\[
\cup \{(tr, X) \mid tr \in \text{divergences}(P) \}
\]

\[
\text{divergences}(P \setminus A) = \{ tr \setminus A \mid tr \in \text{divergences}(P) \}
\]

\[
\cup \{(u \setminus A) \cap tr \mid u \in \text{infinities}(P) \land \#(u \setminus A) < \infty \land tr \in (\alpha P - A)^* \}
\]

\[
infinities(P \setminus A) = \{ u \setminus A \mid u \in \text{infinities}(P) \land \#(u \setminus A) = \infty \}
\]

2.4.4 Renaming

If \( f \) is a mapping from a set of events \( A \) to a set of events \( B \), then two alphabet renaming operators are defined as follows:

\[
\alpha(f(P)) = f(\alpha(P))
\]

\[
\text{traces}(f(P)) = \{ f(tr) \mid tr \in \text{traces}(P) \}
\]

\[
\text{failures}(f(P)) = \{ (f(tr), X) \mid (tr, f^{-1}(X)) \in \text{failures}(P) \}
\]

\[
\text{divergences}(f(P)) = \{ f(tr) \cap tr' \mid tr \in \text{divergences}(P) \land tr' \in \alpha(f(P))^* \}
\]

\[
infinities(f(P)) = \{ f(u) \mid u \in \text{infinities}(P) \}
\]

\[
\alpha(f^{-1}(P)) = f^{-1}(\alpha(P))
\]

\[
\text{traces}(f^{-1}(P)) = \{ f^{-1}(tr) \mid tr \in \text{traces}(P) \}
\]

\[
\text{failures}(f^{-1}(P)) = \{ (tr, X) \mid (f(tr), f^{-1}(X)) \in \text{failures}(P) \}
\]

\[
\text{divergences}(f^{-1}(P)) = \{ tr \setminus tr' \mid f(tr) \in \text{divergences}(P) \land tr' \in \alpha(f^{-1}(P))^* \}
\]

\[
infinities(f^{-1}(P)) = \{ u \mid f(u) \in \text{infinities}(P) \}
\]

**Lemma 2.2.** If \( P \) is divergence-free, and for any 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. \( \square \)
2.5 Lazy abstraction

To separate out consideration of divergence from reasoning about traces, we will use $P \mid\mid \text{RUN}_N$ as a lazy abstraction operator. In the TDI model $P \mid\mid \text{RUN}_N$ masks all occurrences of $N$ in $P$. We use $P_0 \mid\mid \text{RUN}_N \subseteq P_1$ rather than $P_0 \subseteq P_1 \setminus N$. They both say that $P_1$ with $N$ abstracted is a refinement of $P_0$, but in the hiding case we also need to worry about introducing divergence. This does not arise in the interleaving case.

The following lemmas give the relationship between refinement results using the two forms of abstraction.

**Lemma 2.3.** If $P_0 \mid\mid \text{RUN}_N \subseteq_{\text{TDI}} P_1$ and $N \cap \alpha(P_0) = \{\}$ and $P_1 \setminus N$ is divergence-free, then $P_0 \subseteq_{\text{TDI}} P_1 \setminus N$.

**Lemma 2.4.** If $P_0 \subseteq_{\text{TDI}} P_1 \setminus N$ and $N \cap \alpha(P_0) = \{\}$ then $P \mid\mid \text{RUN}_N \subseteq_{\text{TDI}} P_1$

3 Event-B

Event-B [Abr10, MAV05] is a state-based specification formalism based on set theory. Here we describe the basic parts of an Event-B machine required for this paper; a full description of the formalism can be found in [Abr10].

3.1 Machines

A machine specification usually defines a list of variables, given as $v$. Event-B also in general allows sets $s$ and constants $c$. 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.

There are many clauses that may appear in Event-B machines, and we concentrate on those clauses concerned with the state. Machines in general may also include clauses relating to given sets and constants, as well as clauses which are to support verification, but in this paper we will not include these.

We will therefore describe a machine $M_0$ with a list of state variables $v$, a state invariant $I(v)$, and a set of events $\text{ev}0, \ldots$ to update the state. Initialisation is a special event $\text{init}$. A refinement $M_1$ of $M_0$ will introduce its own state variables, invariant, and events. Its invariant will relate the state of $M_1$ to that of the refined machine $M_0$. It may also include a variant clause, used to show that newly introduced events cannot occur indefinitely.
machine $M_0$
variables $v$
invariant $I(v)$
events $evt_0$, ...
end

machine $M_1$
variables $w$
invariant $J(v, w)$
events $evt_1$, ...
variant $V(w)$
end

A machine $M_0$ or $M_1$ 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.

### 3.2 Events

Central to an Event-B description is the definitions 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.

A machine also has an initialisation event $init$.

An event in a refinement machine can also indicate the event that it refines, in the `refines` clause. It may also include a `status` of `convergent` or `anticipated`, indicating respectively whether it is intended to decrease, or not increase, the machine variant, used for reasoning about divergence.

This gives rise to a mapping $f_{M_1}$ which maps events in $M_1$ to the events they refine in $M_0$. It is defined by $f_{M_1}(b) = a$ where $b$ `refines` $a$ appears in $M_1$. It is a partial function, since not all events in $M_1$ necessarily refine events in $M_0$; some events in $M_1$ may be newly introduced.

Often the mapping $f_{M_1}$ is simply the identity mapping on the events on $M_0$: this is the case when events in $M_0$ are refined by events in $M_1$ of the same name.

The Event-B approach to refinement allows an event to be refined by a number of events. This is called splitting events in [Abr10, Section 14.6.1]. In the general case of $M_0 \subsetneq M_1$, $evt_0$ of $M_0$ may be refined by several events in $M_1$: $evt_1$ `refines` $evt_0$, $evt_1'$ `refines` $evt_0$, ..., $evt_1''$ `refines` $evt_0$. In such cases $f_{M_1}$ will be many-to-one.

We will use the following form for events and their refinements, as the most suitable for the proofs in this paper. This is a slight variation from the form of [MAV05], which included the nondeterminism within the event more explicitly.
Proof obligations on events can be expressed in terms of weakest precondition semantics on statements, where $[S]R$ denotes the weakest precondition for statement $S$ to guarantee to establish postcondition $R$.

Events of the form `when G(v) then S(v) end` can be abbreviated as $G(v) \implies S(v)$.

Weakest preconditions for events of the form "`when G(v) then S(v) end"" are given by considering them as guarded commands:

\[
\left[ \text{when } G(v) \text{ then } S(v) \text{ end} \right] P = G(v) \Rightarrow [S(v)] P
\]

Events in the general form "`when G(v) then \( v : BA(v, v') \) end"" have a weakest precondition semantics as follows:

\[
\left[ \text{when } G(v) \text{ then } v : BA(v, v') \text{ end} \right] P = G(v) \Rightarrow \forall x.(BA(v, x) \Rightarrow P[x/v])
\]

Observe that for the case $P = true$ we have

\[
\left[ \text{when } G(v) \text{ then } v : BA(v, v') \text{ end} \right] true = true
\]

### 3.3 Proof obligations for refinement

A machine $M_0$ is refined by another machine $M_1$, written $M_0 \ll M_1$, if there is a linking invariant (i.e. a predicate) $J$ on the variables of the two machines, which is established by their initialisations, and which is preserved by all events, in the sense that any event of $M_1$ can be matched by an event of $M_0$ (or `skip` for newly introduced events, as described below) to maintain $J$. This is the standard notion of downwards simulation data refinement [DB01]. The standard Event-B proof obligations for each event in a refinement are given in [MAV05] as $FIS\_REF$, $GRD\_REF$, and $INV\_REF$. They express respectively: that the refined event is feasible; that abstract events are enabled when their refinements are; and that the linking invariant is preserved on occurrence of events. We will use the refinement relation $M_0 \ll M_1$ to mean that the three proof obligations $FIS\_REF$, $GRD\_REF$, and $INV\_REF$ hold between $M_0$ and $M_1$.

New events can also be introduced, in which case they are treated as data refinements of `skip`. A variant $V$ must also be introduced. New events must have a status of `convergent` or `anticipated`, and in each case the associated proof
obligation \( WFD_{\text{REF}} \) should be established with respect to the variant \( V \). The new events need not always be enabled, but their execution should maintain the linking relationship to the same abstract state.

Furthermore, any refinement of an anticipated event must have status \emph{convergent} or \emph{anticipated}. Refinements of convergent events, and of unlabelled events, need not be labelled.

If refinement introduces a set of new events \( N \), then we will include \( N \) as a superscript in the refinement relation: \( M_0 \leq^N M_1 \). This means that the four proof obligations \( FIS_{\text{REF}}, GRD_{\text{REF}}, INV_{\text{REF}}, \) and \( WFD_{\text{REF}} \) hold between \( M_0 \) and \( M_1 \).

We 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_{\text{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:

\[
\begin{align*}
I(v) \land J(v, w) \land H(w) \\
\rightarrow \\
\exists w'.BA1(w, w')
\end{align*}
\]

**GRD_{\text{REF}}: Guard Strengthening** This requires that when a concrete event is enabled, then so is the abstract one. The rule is:

\[
\begin{align*}
I(v) \land J(v, w) \land H(w) \\
\rightarrow \\
G(v)
\end{align*}
\]

**INV_{\text{REF}}: Simulation** This ensures that the occurrence of events in the concrete machine can be matched in the abstract one. New events are treated as refinements of \emph{skip}. The rule is:

\[
\begin{align*}
I(v) \land J(v, w) \land H(w) \land BA1(w, w') \\
\rightarrow \\
\exists v'.(BA0(v, v') \land J(v', w'))
\end{align*}
\]
Event-B also allows a variety of further proof obligations for refinement, depending on what is appropriate for the application. The two parts of the variant rule WFD_REF below must hold for all newly-introduced events.

**WFD_REF: Variant** This rule ensure that a 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:

<table>
<thead>
<tr>
<th>$I(v) \land J(v, w) \land H(w) \land BA1(w, w')$</th>
<th>WFD_REF (convergent event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\vdash V(W) \in \mathbb{N} \land V(w') &lt; V(w)$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$I(v) \land J(v, w) \land H(w) \land BA1(w, w')$</th>
<th>WFD_REF (anticipated event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\vdash V(W) \in \mathbb{N} \land V(w') \leq V(w)$</td>
<td></td>
</tr>
</tbody>
</table>

## 4 CSP semantics for B machines

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 that 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 dual to that presented in [Abr10], where traces are considered as sequences of *states* rather than our treatment of traces as sequences of *events*.

The alphabet $\alpha M$ of a machine $M$ is simply its set of *events*.

The CSP semantics is based on the weakest precondition semantics of events, as given above.

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

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

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

11
The failures of a machine $M$ are those pairs $(tr, X)$ for which performing $tr$ followed by refusing $X$ is possible:

$$\text{failures}(M) = \{(tr, X) \mid \neg[\text{init}; tr][[\bigvee_{op \in X} G_{op}(c, v)]])\}$$

In other words, it is not always the case that performance of $tr$ is followed by some event from $X$ being enabled.

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

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

Note that any Event-B machine $M$ with events of the form $evt$ given in Section 3.2 is divergence-free. This is because $[evt]\text{true} = \text{true}$ for such events (and for $\text{init}$), and so $[\text{init}; tr]\text{true} = \text{true}$. Thus no potential divergence $tr$ meets the condition $\neg[\text{init}; tr]\text{true}$.

Infinite Traces 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 $\neg[\text{init}](\neg P_0)$ (i.e. some execution of $\text{init}$ reaches a state where $P_0$ holds), and $P_i \Rightarrow \neg[\text{ui}](\neg P_{i+1})$ for each $i$ (i.e. if $P_i$ holds then some execution of $u_i$ can reach a state where $P_{i+1}$ holds).

$$\text{infinities}(M) = \{u \mid \exists \langle P_i \rangle \in \mathbb{N}. \quad \neg[\text{init}](\neg P_0) \land \forall i . P_i \Rightarrow \neg[\text{ui}](\neg P_{i+1}) \}$$

These definitions give the CSP Failures/Divergences/Infinite Traces semantics of Event-B machines in terms of the weakest precondition semantics of events.

The following lemmas relate results on Event-B refinements to associated results in the CSP framework. They are similar to the results of [But92, Chapter 4], extended to include event renaming.

The following lemma shows the relationship between Event-B refinement and CSP refinement.

**Lemma 4.1.** If $M \preceq^N M'$ then $f_{\text{map}}^{-1}(M) \boxplus RUN_N \subseteq_{TDI} M'$

In this paper we will use without further comment that $M$ is divergence-free: that any individual event $e$ in $M$ does not diverge, i.e. that

$$(I(v) \land G(v)) \Rightarrow [S(v)]\text{true} \quad (1)$$
5 Refinement

5.1 Stepwise Refinement

5.1.1 Refinement steps

The following CSP results will be applicable to Event-B\|CSP combinations. We concentrate on the relationship between refinement levels identified in Lemma 4.1. Theorem 5.1 is concerned with the relationship between processes at different levels of the refinement chain. Theorem 5.11 is concerned with the treatment of convergent and remaining events through the refinement chain.

**Theorem 5.1.** If a sequence of processes $P_i$, mappings $f_i$, and sets $N_i$ are such that

$$f_{i+1}^{-1}(P_i) \parallel \text{RUN}_{N_{i+1}} \subseteq_{TDI} P_{i+1} \quad (2)$$

for each $i$, and $N_{i+1} \cap f_{i+1}^{-1}(\alpha P_i) = \{\}$ then

$$f_n^{-1}(\ldots(f_1^{-1}(P_0))\ldots) \parallel \text{RUN}_{f_n^{-1}(\ldots f_1^{-1}(P_0))} \subseteq_{TDI} P_n$$

**Proof.** Two successive refinement steps combine to provide a relationship between $P_0$ and $P_2$ of the same form as Line 2 above, as follows:

$$f_2^{-1}(f_1^{-1}(P_0) \parallel \text{RUN}_{N_1}) \parallel \text{RUN}_{N_2} \subseteq_{TDI} P_2 \quad \text{(given)}$$

$$f_2^{-1}(f_1^{-1}(P_0)) \parallel \text{RUN}_{N_1} \subseteq_{TDI} P_2 \quad \text{(line 2, transitivity of $\subseteq$)}$$

$$f_2^{-1}(f_1^{-1}(P_0)) \parallel \text{RUN}_{N_1} \subseteq_{TDI} P_2 \quad (N_1 \cap f_1^{-1}(\alpha P_0) = \{\})$$

$$f_2^{-1}(f_1^{-1}(P_0)) \parallel \text{RUN}_{N_1} \subseteq_{TDI} P_2 \quad (N_2 \cap f_2^{-1}(\alpha P_1) = \{\})$$

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

5.1.2 Parallel composition

A CSP lemma:

**Lemma 5.2.** If

1. $f^{-1}(P_0) \parallel \text{RUN}_{\alpha P_1 \cup \alpha P_2} \subseteq_{TDI} P_1$

2. $f^{-1}(Q_0) \parallel \text{RUN}_{\alpha Q_1 \cup \alpha Q_2} \subseteq_{TDI} Q_1$

then

$$f^{-1}(P_0 \parallel Q_0) \parallel \text{RUN}_{((\alpha P_1 \cup \alpha P_2) - (\alpha P_0 \cup \alpha Q_2))} \subseteq_{TDI} (P_1 \parallel Q_1)$$
Lemma 5.2 yields the following result for a combination of a CSP controller and a B machine.

**Corollary 5.3** (Trace refinement). If

1. \((\alpha(M_1) \circ \mathcal{f}) = f_{M_1}\)
2. \(f^{-1}(P_0) \parallel \text{RUN}_{\alpha P_1 - \alpha P_0} \preceq_{\text{TDI}} P_1\)
3. \(M_0 \preceq^N M_1\)

then

\[ f^{-1}(P_0 \parallel M_0) \parallel \text{RUN}_{((\alpha P_1 \cup \alpha M_1) - (\alpha P_0 \cup \alpha M_0))} \preceq_{\text{TDI}} (P_1 \parallel M_1) \]

This theorem covers the general case of refining \(P_0\) and \(M_0\). It allows each component to introduce events that are already in the alphabet of the other. In this case, the resulting combination should not hide such events. Thus in the case of a clash, existing visibility of events takes precedence over the hiding of newly introduced events.

The alphabet conditions are illustrated in Figure 1. Observe that the new process events \(\alpha P_1 - \alpha P_0\) can overlap with the alphabet of \(M_0\), and also with the new machine events \(\alpha M_1 - \alpha M_0\). Also the new machine events can overlap with the alphabet of \(P_0\). The shaded region shows the new events \((\alpha P_1 \cup \alpha M_1) - (\alpha P_0 \cup \alpha M_0)\), which is abstracted through \(\text{RUN}\) to obtain the refinement result.

In a chain of refinement steps, each step \(P_i \parallel M_i\) to \(P_{i+1} \parallel M_{i+1}\) therefore comes with a set of new events, and an event mapping. Theorem 5.1 yields the following key theorem.
Theorem 5.4 (First Refinement Theorem). If \( P_i, M_i, \) and \( f_i \) are sequences of processes, machines, and event mappings, such that for each \( i \),

- \( (\alpha(M_i) \circ f_i) = f_{M_i} \)
- \( f_{i+1}^{-1} (P_i) \parallel RUN(\alpha(P_{i+1}) - \alpha(P_i)) \subseteq TDI \ P_{i+1} \)
- \( M_i \leqslant (\alpha M_{i+1} - \alpha M_i) M_{i+1} \)
- \( N_{i+1} = ((\alpha P_{i+1} \cup \alpha M_{i+1}) - (\alpha P_i \cup \alpha M_i)) \)

then

\[
\begin{align*}
    f_n^{-1} \ldots (f_1^{-1}(P_0 \parallel M_0)) \ldots \parallel \ & RUN_{f_n}^{\ldots} (\ldots f_2^{\ldots} (N_{i+1}) \ldots) \cup \ldots \cup f_1^{\ldots} (N_0) \cup N_n \\
    \subseteq & TDI \ P_n \parallel M_n
\end{align*}
\]

This formalises the relationship between the events in the initial level \( P_0 \parallel M_0 \) and in the final refinement level \( P_n \parallel M_n \).

In the particular case where new events in one component are not already present in the other, we obtain a simpler formulation for the refinement, given in the following corollary.

Corollary 5.5. If

1. \( P_0 \parallel \text{RUN}_N \subseteq TDI \ P_1 \)
2. \( M_0 \leqslant N^* \ M_1 \)
3. \( N' \cap \alpha P_0 = \{\} \)
4. \( N \cap \alpha M_0 = \{\} \)

then

\[
    (P_0 \parallel M_0) \parallel \text{RUN}_{N \cup N^*} \subseteq TDI \ (P_1 \parallel M_1)
\]

Another corollary concerns the case where no new events are introduced.

Corollary 5.6. If

1. \( P_0 \subseteq TDI \ P_1 \)
2. \( M_0 \leqslant M_1 \)

then

\[
P_0 \parallel M_0 \subseteq TDI \ P_1 \parallel M_1
\]
5.2 Convergent and anticipated events

5.2.1 A CSP approach to convergent and anticipated events

Convergent and anticipated events are used in Event-B. In order to deal with them we introduce a predicate $CA$ which expresses a property in CSP that captures the relationship between them: that if convergent events $C$ occur infinitely often, then there must be infinite occurrences of events $R$ that are neither convergent nor anticipated. This is expressed by the predicate $CA(C, R)$.

**Definition 5.7.** The predicate $CA(C, R)$ is defined as follows:

$$CA(C, R)(u) \iff \left( \#(u \upharpoonright C) = \infty \Rightarrow \#(u \upharpoonright R) = \infty \right)$$

There are several immediate consequences of this definition:

**Lemma 5.8.** If $P$ is a CSP process, then

1. If $P$ sat $CA(C, R)$ then $f^{-1}(P)$ sat $CA(f^{-1}(C), f^{-1}(R))$
2. If $P$ sat $CA(C, R)$ and $N \cap C = \{\}$ then $P \parallel RUN_N$ sat $CA(C, R)$
3. If $P$ sat $CA(C, R)$ and $P$ sat $CA(C', C \cup R)$ then $P$ sat $CA(C \cup C', R)$
4. If $P$ sat $CA(C, R)$ and $C \cap R = \{\}$ then $P \setminus C$ is divergence-free

**Proof.**

1. follows directly from the CSP semantics.

2. Consider $u \in \text{infini}\text{tes}(P \parallel RUN_N)$ with $u \upharpoonright C$ infinite. Each $C$ arises from $P$ (since $N \cap C = \{\}$), hence $P$ performs infinitely many $C$ events. Thus it also performs infinitely many $R$ events since $P$ sat $CA(C, R)$, hence $u \upharpoonright R$ is infinite.

3. Consider $u \in \text{infini}\text{tes}(P)$, and assume that $\text{upr}\text{oject}(C \cup C')$ infinite. If $u \upharpoonright C$ is infinite, then $u \upharpoonright R$ is infinite since $CA(C, R)(u)$. Otherwise $u \upharpoonright C$ is finite, in which case $u \upharpoonright C'$ is infinite. But then $u \upharpoonright (C \cup R)$ is infinite since $CA(C', C \cup R)(u)$. It follows that $u \upharpoonright R$ is infinite, since $u \upharpoonright C$ is finite. Hence in all cases it follows that $u \upharpoonright R$ is infinite, from the initial assumption that $u \upharpoonright (C \cup C')$ is infinite. Hence $CA(C \cup C', R)(u)$. Since this is true for any $u \in \text{infini}\text{tes}(P)$, we conclude that $P$ sat $CA(C \cup C', R)$.

4. If $P$ sat $CA(C, R)$ then $P$ is divergence-free (since any divergence trace can be followed by an infinite sequence of $C$ events, violating $CA(C, R)$). Now given any $u \in \text{infini}\text{tes}(P)$, if $u \upharpoonright C$ is finite then $u \setminus C$ is infinite, and if $u \upharpoonright C$ is infinite, then $u \upharpoonright R$ is infinite, and so $u \setminus C$ is infinite. Hence from the semantics of hiding, $P \setminus C$ is divergence-free.

$\square$
The following corollary establishes a result between processes $P_0$ and $P_1$ which are related by the Event-B refinement relation:

**Corollary 5.9.** If

1. $P_0 \text{ sat } CA(C, R)$
2. $f^{-1}(P_0) \parallel\!\!\parallel RUN_N \subseteq_{TDI} P_1$
3. $N \cap f^{-1}(C) = \{\}$

then $P_1 \text{ sat } CA(f^{-1}(C), f^{-1}(R))$

**Proof.** Follows from Lemma 5.8 (1) and (2) \(\square\)

The next lemma shows how to combine CA properties for $P_0$ and $P_1$ into a combined CA property for $P_1$:

**Lemma 5.10.** If

1. $P_0 \text{ sat } CA(C_0, R_0)$
2. $P_1 \text{ sat } CA(C_1, R_1)$
3. $R_1 = f_1^{-1}(C_0) \cup f_1^{-1}(R_0)$
4. $N_1 \cap R_1 = \{\}$
5. $f_1^{-1}(P_0) \parallel\!\!\parallel RUN_{N_1} \subseteq_{TDI} P_1$

then $P_1 \text{ sat } CA(f_1^{-1}(C_0) \cup C_1, f_1^{-1}(R_0))$

**Proof.** This is justified as follows:

\[
\begin{align*}
N_1 \cap f_1^{-1}(C_0) & = \{\} & R_1 & = f_1^{-1}(C_0) \cup f_1^{-1}(R_0) \\
P_1 & \text{ sat } CA(f_1^{-1}(C_0), f_1^{-1}(R_0)) & \text{corollary 5.9} \\
P_1 & \text{ sat } CA(C_1, R_1) & \text{given} \\
P_1 & \text{ sat } CA(C_1, f_1^{-1}(C_0) \cup f_1^{-1}(R_0)) & R_1 & = f_1^{-1}(C_0) \cup f_1^{-1}(R_0) \\& \text{lemma 5.8 (3)}
\end{align*}
\]

with $C'' = C_1$, $C = f_1^{-1}(C_0)$, $R = f_1^{-1}(R_0)$

\(\square\)

We then obtain the following theorem, which obtains a combined CA property for the final process in a refinement chain in which all intermediate processes meet a CA property:
\[ I(v) \land G(v) \land BA1(v, v') \]
\[ \vdash V(v) \in \mathbb{N} \land V(v') < V(v) \]

Figure 2: CNV: Convergence in Event-B machines

\[ I(v) \land G(v) \land BA1(v, v') \]
\[ \vdash V(v) \in \mathbb{N} \land V(v') \leq V(v) \]

Figure 3: ANT: Anticipation in Event-B machines

**Theorem 5.11.** If a sequence of processes \( P_i \), mappings \( f_i \), and sets \( C_i \), \( R_i \), and \( N_i \) meet the following conditions

1. \( P_i \) sat \( CA(C_i, R_i) \)
2. \( f_{i+1}^{-1}(R_i \cup C_i) = R_{i+1} \)
3. \( N_i \cap R_i = \{\} \)
4. \( f_{i+1}^{-1}(P_i) \parallel \text{RUN}_{N_i+1} \supseteq_{TDI} P_{i+1} \)

then

\( P_n \) sat \( CA( (f_n^{-1}(\ldots f_1^{-1}(C_0)\ldots) \cup \ldots f_n^{-1}(C_{n-1}) \cup C_n) , f_n^{-1}(\ldots f_1^{-1}(R_0)\ldots) ) \)

**Proof.** Two successive steps combine the results of Lemma 5.10 to obtain a combined \( CA \) property, as follows: if \( P_0 \), \( P_1 \) and \( P_2 \) meet the conditions, then

\[ \begin{align*}
P_2 & \text{ sat } CA(f_2^{-1}(C_1) \cup C_2 , f_2^{-1}(R_1)) & \text{ lemma 5.10} \\
P_2 & \text{ sat } CA(f_2^{-1}(C_1) \cup C_2 , f_2^{-1}(f_1^{-1}(R_0)) \cup f_2^{-1}(f_1^{-1}(C_0))) & R_1 = f_1^{-1}(R_0 \cup C_0) \\
P_2 & \text{ sat } CA(f_2^{-1}(f_1^{-1}(C_0)) , f_2^{-1}(f_1^{-1}(R_0))) & \text{ corollary 5.9 twice} \\
P_2 & \text{ sat } CA(f_2^{-1}(f_1^{-1}(C_0)) \cup f_2^{-1}(C_1) \cup C_2 , f_2^{-1}(f_1^{-1}(R_0))) & \text{ lemma 5.8 (3)} 
\end{align*} \]

Hence the whole chain of refinement steps can be collected together, yielding the result. \( \square \)

### 5.2.2 Application to Event-B∥CSP

**Definition 5.12.** A set of events \( E \) converges in machine \( M \) with variant \( V \) if the proof obligation CNV (Figure 2) is true for all events in \( E \).
**Definition 5.13.** A set of events $E$ is anticipated in machine $M$ with variant $V$ if the proof obligation $\text{ANT}$ (Figure 3) holds for all events in $E$.

We obtain the following lemma with respect to convergent and anticipated events.

**Lemma 5.14.** If $M$ has anticipated events $A$, convergent events $C$ and remaining events $R = \alpha M - (C \cup A)$ then $M$ sat $CA(C, R)$.

**Proof.** Consider an infinite trace $u$ with $\# u \upharpoonright C = \infty$. Now consider the value of the variant $V$ during the execution of $u$. It decreases infinitely often, since $u \upharpoonright C$ is infinite. Hence it must increase infinitely often. Events in $C \cup A$ do not increase $V$, hence there must be infinitely many occurrences of events other than $C \cup A$. Hence $u \setminus (C \cup A)$ is infinite, i.e. $u \upharpoonright R$ is infinite.

Now we consider the treatment of convergent and remaining events in a chain of controlled components $P_i \parallel M_i$. To do this, we need to identify the sets $C_i$, $R_i$ and $\alpha_i$ for the parallel combinations, which meet the conditions required for Theorem 5.11.

We are interested in the sets $C_i$ and $R_i$ for which $P_i \parallel M_i$ sat $CA(C_i, R_i)$. Those are the sets that we will be able to say are convergent and remaining sets for a combination.

We obtain a general theorem to handle the combination of $P$ and $M$.

**Theorem 5.15.** If

- $M$ sat $CA(C, R)$ (with $C \cap R = \{\}$)
- $P$ sat $CA(C', R')$ (with $C' \cap R' = \{\}$)
- $C' \cap R = \{\}$

then $(P \parallel M)$ sat $CA(C \cup C', (R \cup R') \setminus C)$.

**Proof.** Consider $u \in \text{infinites}(P \parallel M)$, with $u \upharpoonright (C \cup C')$ infinite. We aim to show that $u \upharpoonright (R \cup R') \setminus C$ is infinite. There are two cases to consider.

**Case** $u \upharpoonright C$ **infinite.** Let $u_1 = u \upharpoonright \alpha M \in \text{infinites}(M)$. Then $\# u_1 \upharpoonright C = \# u \upharpoonright C = \infty$, so by the definition of $CA$ we have $\# u_1 \upharpoonright R$ is infinite. Since $C \cap R = \{\}$, we obtain $u \upharpoonright (R \cup R') \setminus C$ is infinite, as required.

**Case** $u \upharpoonright C$ **finite** Then $u \upharpoonright C'$ is infinite. Hence $(u \upharpoonright R')$ is infinite, and so $u \upharpoonright (R' \setminus C)$ is infinite, thus $u \upharpoonright (R \cup R') \setminus C$ is infinite, as required.

\[\blacksquare\]
Corollary 5.16. If

- \( P_1 \) sat \( CA(C, R) \)
- \( P_2 \) sat \( CA(C', R') \)
- \( C \cap R' = \{\} \)
- \( C' \cap R = \{\} \)

then \( P_1 \parallel P_2 \) sat \( CA(C \cup C', R \cup R') \).

These results yield the following theorem:

Theorem 5.17 (Second Refinement Theorem). If \( P_i \parallel M_i \) is a chain of controlled components, with event mappings \( f_i \), such that:

1. \( \alpha(M_i) \circ f_i = f_M \)
2. \( f_{i+1}^{-1}(P_i) \parallel RUN(\alpha(P_{i+1}) - \alpha(P_i)) \equiv TDL P_{i+1} \)
3. \( M_i \leq (\alpha M_{i+1} - \alpha M_i) M_{i+1} \)
4. \( M_i \) has convergent events \( C_i' \), anticipated events \( A_i' \), and remaining events \( R_i' = \alpha(M_i) - (C_i' \cup A_i') \).
5. \( P_i \) sat \( CA(C_i'', R_i'') \)
6. \( C_i'' \cap R_i' = \{\} \)

Then

\( P_n \parallel M_n \) sat

\[ CA( f_n^{-1}(\ldots f_1^{-1}(C_0) \ldots) \cup \ldots f_{n-1}^{-1}(C_{n-1}) \cup C_n) \cup f_n^{-1}(\ldots f_1^{-1}(R_0) \ldots) \]

where \( C_i = C_i' \cup C_i'' \), and \( R_i = (R_i' \cup R_i'') \setminus C_i' \) for each \( i \).

Proof. Conditions (4), (5) and (6) provide the conditions for Theorem 5.15 to apply, yielding \( (P_i \parallel M_i) \) sat \( CA(C_i'' \cup C_i', (R_i'' \cup R_i') \setminus C_i') \). The sequence of controlled components \( P_i \parallel M_i \) thus meets the conditions of Theorem 5.11, from which the result follows. \( \square \)

Lemma 5.8 (4) yields the following corollary:

Corollary 5.18. \( (P_n \parallel M_n) \setminus (f_n^{-1}(\ldots f_1^{-1}(C_0) \ldots) \cup \ldots f_{n-1}^{-1}(C_{n-1}) \cup C_n)) \) is divergence-free.

If there are no anticipated events in \( P_n \parallel M_n \) then all the events introduced during the refinement are in \( (f_n^{-1}(\ldots f_1^{-1}(C_0) \ldots) \cup \ldots f_{n-1}^{-1}(C_{n-1}) \cup C_n)) \), and so it follows that hiding all the events so introduced preserves divergence-freedom.
5.2.3 Devolved events

Observe that in Theorem 5.17, $C_i = C_i' \cup C_i''$, so events convergent in $M_i \parallel P_i$ are those convergent in $M_i$ together with those convergent in $P_i$. Since $C_i'' \cap R_i' = \{\}$, any events convergent in $P_i$ are either convergent in $M_i$ or anticipated in $M_i$. The inclusion of such events in $C_i$ means that their refinements in $M_{i+1}$ do not need to have a status of convergent or anticipated, since that requirement holds only for events that are anticipated (and hence not in $C_i$) within $P_i \parallel M_i$.

We propose a new status ‘devolved’ for such events to replace ‘anticipated’ where appropriate. Proof obligations on devolved events require that they behave as anticipated events (thus not increasing the variant), and that they are convergent in $P_i$. Discharging these proof obligations means that any refinement of a devolved event in $M_{i+1}$ does not need to have a status.

5.2.4 Establishing CA for CSP processes

The application of Theorem 5.17 requires that the CSP controllers $P_i$ sat $CA(C_i', R_i')$ for some $C_i'$ and $R_i'$. The following lemmas provide ways of establishing such properties.

The first lemma gives a default CA property that a process $P$ will meet.

**Lemma 5.19.** For any process $P$, $P$ sat $CA(\{\}, \alpha_P)$

Using this CA property will always discharge conditions (5) and (6) of Theorem 5.17. It corresponds to the situation where the CSP controller does not have any responsibility for convergence of any events.

The next lemma gives the case where the controller ensures that a set $C$ is convergent:

**Lemma 5.20.** If $P \setminus C$ is divergence-free, then $P$ sat $CA(C, \alpha_{P - C})$

Finally, we can establish CA properties more generally using model-checking in a tool such as FDR[For]. In order to do this, we identify finite-state approximations $CA_n$ to $CA$, as follows:

**Definition 5.21.** Let $CA_n$ be defined as follows:

$$CA_n(C, R)(tr) \equiv tr = tr_0 \land \#(tr' \mid C) > n \Rightarrow \#(tr' \mid R) > 0$$

$CA_n$ holds if there must be at least one occurrence of an event from $R$ for every $n$ occurrences of events from $C$. Then we have a sufficient condition for establishing $CA(C, R)$:

**Lemma 5.22.** If $P$ sat $CA_n(C, R)$ for some $n$, then $P$ sat $CA(C, R)$
This follows from the fact that $CA_n(C, R)(u) \Rightarrow CA(C, R)(u)$. □

Lemma 5.22 is useful because $CA_n$ can be formulated as a finite state process, and hence used as a refinement specification. Define

$$REM(i, n, C, R) = \begin{cases} x : R \rightarrow REM(n, n, C, R) & \text{(} i > 0 \text{)} \\ (i = 0) \land y : C \rightarrow REM(n, n, C, R) & \end{cases}$$

Then we obtain:

**Lemma 5.23.** $P \text{ sat } CA_n(C, R) \text{ iff } (\text{RUN}_A \parallel R) \Rightarrow \square_P$

### 6 Bounded Retransmission Protocol Example

We present a case study illustrating a refinement chain. The case study is inspired by Abrial’s treatment of the Bounded Retransmission Protocol [Abri10], which in turn was based on [GvdP96]. Our approach uses CSP rather than control variables in Event-B to manage the control flow of events in an explicit and visible way.

The case study illustrates the transfer of a file by sending data packets over an unreliable medium. CSP is used to describe the repetitious behaviour in the sender (repeated transmission, and progress through the file) and the receiver (progressive receipt of the data packets), whereas the Event-B part of the model focuses on the state. For the purposes of the case study we focus only on the unreliability of the transmission medium, allowing reliable acknowledgements.

The events introduced through the development are shown in Figure 4.

**Level 0**

In the initial level, given in Figure 5, we see the CSP controller split into a sender controller and a receiver controller. We begin with Abrial’s model, with a single sender and a single receiver event. The event $\text{brp}$ occurs after the protocol has completed.

**Level 1**

In the first refinement step the progress events are split into success and failure events, and an additional requirement on the relationship between the
Figure 4: Events introduced through the development

\begin{align*}
&D_{MN} \\
&D_{progress} \\
&D_{rcv} \\
&D_{timeout} \\
&S_{progress} \\
&S_{rcv} \\
&S_{timeout} \\
&R_{progress} \\
&R_{rcv} \\
&R_{timeout} \\
&S_{success} \\
&S_{fail} \\
&R_{success} \\
&R_{fail} \\
&\text{brp} \\
&f_1 \\
&f_2 \\
&f_3 \\
&f_4 \\
\end{align*}

Figure 5: Level 0: Machine $M_0$ events and control process $P_0$

\[
\begin{align*}
P_0 &= S_0 \parallel R_0 \\
S_0 &= \text{SND}_\text{progress} \rightarrow \text{brp} \rightarrow \text{STOP} \\
R_0 &= \text{RCV}_\text{progress} \rightarrow \text{brp} \rightarrow \text{STOP}
\end{align*}
\]
\begin{align*}
P_1 & = S_1 \parallel R_1 \\
S_1 & = (\text{SND\_success} \rightarrow \text{brp} \rightarrow \text{STOP}) \square (\text{SND\_failure} \rightarrow \text{brp} \rightarrow \text{STOP}) \\
R_1 & = (\text{RCV\_success} \rightarrow \text{brp} \rightarrow \text{STOP}) \square (\text{RCV\_failure} \rightarrow \text{brp} \rightarrow \text{STOP})
\end{align*}

**Figure 6:** Level 1: Machine $M_1$ events and control process $P_1$
sender’s and the receiver’s final state is introduced. The resulting machine and controller are given in Figure 6. The associated renaming function is

\[
\begin{align*}
 f_1(SND\_success) &= f_1(SND\_failure) = SND\_progress \\
 f_1(RCV\_success) &= f_1(RCV\_failure) = RCV\_progress \\
 f_1(brp) &= brp
\end{align*}
\]

There are no new events at this level.

Then \( P_0 \sqsubseteq_T f_1(P_1) \). Also each event \( a \) of \( M_1 \) has that \( a \) refines \( f_1(a) \). Hence

\[
P_0 \parallel M_0 \sqsubseteq_T f_1(P_1 \parallel M_1)
\]

**Level 2**

In the second refinement step, we introduce the data file \( p : 1..n \rightarrow D \) to be transferred. Reception of data packets will be modelled with a new convergent event in the receiver part of the description, an adjustment to \texttt{RCV\_success}, with all other events remaining unchanged. A loop is introduced into the CSP controller. Observe that in this example it is the convergence of the \texttt{B} event that ensures that the new event cannot occur indefinitely.
$N_2$ is the set of events that have been newly introduced at this level. There is only one such event:

\[ N_2 = \{ RCV_{\text{rcv\_current\_data}} \} \]

No event renaming has occurred, so $f_2$ will be the identity function and can be ignored. This will be the case with all subsequent refinement levels.

The new event introduced for $M_2$, and the event strengthened from $M_1$ and $M_2$, are given in Figure 7, along with the control process $P_2$.

Then $P_1 \parallel RUN(N_2) \sqsubseteq_T P_2$.

Hence $(P_1 \parallel M_1) \parallel RUN(N_2) \sqsubseteq_T (P_2 \parallel M_2)$.

### Level 3

In the third refinement step, we make use of the new status for events in controlled components: ‘devolved’. We introduce new events into the sender controller: a devolved event, a convergent event, and an anticipated event. We also refine two of the receiver events. These are given in Figure 8. All other events remain unchanged. We also introduce a data channel $db$ which is set and reset by the sender when sending data.

The CSP controller, shown in Figure 9, is used to manage the flow of events in the sender. In the pure Event-B version [Abr10], an additional control variable is needed to manage the interaction between the sender events. Here, the relationship between their occurrence is given explicitly in $S_3$.

The requirement $M_2 \preceq M_3$ requires that \texttt{SND\_rcv\_curr\_ack} decreases the variant $V_3$, that \texttt{SND\_timeout} does not increase $V_3$, and that the strengthened receiver events are appropriate refinements. We must also show that the devolved event \texttt{SND\_snd\_data} does not increase $V_3$.

Then $P_2 \parallel RUN(N_3) \sqsubseteq_T P_3$, where

\[ N_3 = \{ \texttt{SND\_snd\_data}, \texttt{SND\_rcv\_curr\_ack}, \texttt{SND\_timeout} \} \]

Observe also that $P_3 \setminus D_3$ is divergence-free, where $D_3 = \{ \texttt{SND\_snd\_data} \}$.

### Level 4

In the final refinement step, we refine the anticipated event \texttt{SND\_timeout} by a convergent event. This is achieved by introducing a counter variable $c$ which places a bound on the number of times the \texttt{SND\_timeout} event can occur without receiving an acknowledgement.
Snd_snd_data
status
devolved
when
s_st = working
then
d := p(s + 1)
db := TRUE
end

Snd_recv_curr_ack
status
convergent
when
s_st = working
s + 1 < n
r = s + 1
then
s := s + 1
db := FALSE
end

Snd_timeout
status
anticipated
when
TRUE
then
skip
end

Rcv_rev_current_data
when
r_st = working
r + 1 < n
r = s
db = TRUE
then
r := r + 1
g := g \cup \{r + 1 \mapsto d\}
end

Rcv_success
when
r_st = working
r + 1 = n
r = s
then
r_st := success
r := r + 1
g := g \cup \{r + 1 \mapsto d\}
end

invariant:
J_3 : g = \{1..r\} \triangleleft p
variant:
V_3 : (n - s)

Figure 8: Level 3: Machine M_3 new and changed events

\begin{align*}
P_3 &= S_3 \parallel R_3 \\
S_3 &= \text{Snd_snd_data} \rightarrow \text{Snd_recv_curr_ack} \rightarrow S_3 \\
& \quad \square \text{Snd_success} \rightarrow \text{brp} \rightarrow \text{STOP} \\
& \quad \square \text{Snd_fail} \rightarrow \text{brp} \rightarrow \text{STOP} \\
& \quad \square \text{Snd_timeout} \rightarrow S_3 \\
R_3 &= R_2
\end{align*}

Figure 9: Level 3: Control process P_3
variant: $V_4 : (MAX - c) + \#(\{FALSE\} - \{db\})$

$P_4 = P_3 || RUN(N_4)$
where
$N_4 = \{DMN\_data\_channel\}$

Figure 10: Level 4: Machine $M_4$ new and changed events, and control process $P_4$
We also model the unreliability of the data channel by introducing the new event DMN_data_channel corresponding to loss of data. The new event and the changed events are given in Figure 10.

At this level, the timeout is refined to a convergent event. Also, the new event DMN_data_channel, which resets the data channel db, is convergent. All events in M3 are refined by their corresponding events in M4. Hence $M_3 \lessdot M_4$.

**Refinement chain**

Finally, we consider the whole chain of refinements from $P_0 \parallel M_0$ to $P_4 \parallel M_4$.

The set of all new events introduced is given by $N = N_2 \cup N_3 \cup N_4$. By Theorem 5.4 the relationship between the initial and final levels is:

$$P_0 \parallel M_0 \sqsubseteq_T f_1((P_4 \parallel M_4) \setminus N)$$

Further, there are no anticipated events left in $M_4$. Hence by Corollary 5.18, $(P_4 \parallel M_4) \setminus N$ is divergence-free.

**7 Discussion**

This report has presented a number of results, based on the CSP semantic models, for Event-B||CSP refinements. The relationship between the initial level in the refinement chain and the final level has been captured as a CSP refinement. Additional conditions on the steps through the refinement allow divergence-freedom properties to be established. The theory also underpins an extension to the treatment of new events in Event-B: there we have that new events, and refinements of anticipated events, must have a status of convergent or anticipated. By considering the combination with CSP, we find that anticipated events which are convergent in the CSP controller can be refined by events without any status. We have introduced a third status, devolved, for such events.

This paper has not considered liveness, in the form of failures refinement or deadlock-freedom. These will be the subject of a separate paper.

We have illustrated the approach with the development of a simple bounded retransmission protocol in Event-B||CSP through a chain of refinement steps. Each step illustrates a refinement rule underpinned by the Event-B||CSP semantics. The result is a description of the protocol with a clear relationship to the original specification. Further, though not considered explicitly in this paper, the protocol transmitting the file is also deadlock-free.

Our example has been chosen in part to enable comparison with the pure Event-B approach taken in [Abr10]. In our approach, inclusion of the CSP controllers
alongside the Event-B description has allowed a clearer and more natural expression of the flow of control of events, particularly with respect to the timeout and repeated transmission of the data. It also allows for simpler event descriptions in the Event-B machine, since control variables in event guards and assignments can be removed where their effect is now taken care of by the CSP controller. In our view the overall behaviour of the system is easier to understand. The cost of this benefit is the need to reconcile two formalisms, and some overhead in ensuring consistency between them.

In terms of tool support available for the approach, one notable model-checking tool that checks combinations of CSP with Event-B (and also classical B) is ProB [LB08], which allows Event-B machines with CSP controllers to be explored for consistency. Results from this form of model-checking augment our approach, since it supports the verification of machine invariants under CSP controllers, even if the machine in isolation is not consistent. Our rules for establishing consistency do not yet cover this case, since they require consistency of the Event-B machine. ProB also supports refinement checking of combinations, though currently this is practicable only on small examples. Alongside ProB, support for the approach will also come from Event-B tools such as the RODIN platform [ABH+10], and from CSP tools such as FDR [For].

References


A Full proof rules from [Abr10]

The key Event-B proof obligations for refinement are given in [Abr10, Section 5.2: Proof Obligation Rules]. We describe each of them in turn:

A.1 Feasibility: FIS and WFIS ([Abr10, p.191,202])

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.

\[
\begin{align*}
\text{evt} & \text{ refines } \text{evt0} \\
\text{any } x \text{ where } & G(s, c, v, x) \\
\text{then } & \text{act } : v : | \text{BA}(s, c, v, x, v') \\
\text{end} & \\
\end{align*}
\]

The rule for feasibility of an event is:

\[
\begin{array}{c|c}
A(s, c) \land I(s, c, v) \land G(s, c, v, x) \\
\quad \vdash \quad \exists v'. \text{BA}(s, c, v, x, v') & \text{FIS}
\end{array}
\]

The rule for feasibility of a concrete event is:
A.2 Guard strengthening: GRD ([Abr10, p.193])

This requires that when a concrete event is enabled, then so is the abstract one.

\[
\begin{align*}
\text{evt} & \text{ refines } \text{evt0} \\
\text{any } x \text{ where} & \quad \text{any } y \text{ where} \\
G(s, c, v, x) & \quad H(y, s, c, w) \\
\ldots & \quad \text{with} \\
\text{then} & \quad x : W(x, s, c, w, y) \\
\ldots & \quad \text{then} \\
\text{end} & \quad \ldots \\
\end{align*}
\]

The rule is then:

\[
\begin{align*}
A(s, c) \land I(s, c, v) \land J(s, c, v, w) \land \\
H(y, s, c, w) \land BA2(s, c, w, y, w') \\
\vdash \quad \exists x. W(x, s, c, w, y, w')
\end{align*}
\]

A.3 Simulation: SIM ([Abr10, p.194-5])

This ensures that the occurrence of events in the concrete machine can be matched in the abstract one. New events are treated as refinements of \text{skip}.

\[
\begin{align*}
\text{evt} & \text{ refines } \text{evt0} \\
\text{any } x \text{ where} & \quad \text{any } y \text{ where} \\
 & \quad H(y, s, c, w) \\
\ldots & \quad \text{with} \\
\text{then} & \quad x : W1(x, s, c, w, y, w') \\
\ldots & \quad v' : W2(v', s, c, w, y, w') \\
\text{then} & \quad w : BA2(s, c, w, y, w') \\
\text{end} & \quad \text{end}
\end{align*}
\]

The rule is then:
A.4 Variant rules NAT and VAR ([Abr10, p.198–200])

\[
\begin{align*}
A(s, c) \land I(s, c, v) \land J(s, c, v, w) \land H(y, s, c, w) \land W_1(x, s, c, w, y, w') & \land W_2(v', s, c, w, y, w') \\
\vdash & \\
BA_2(s, c, w, y, w') & \\
\end{align*}
\]

SIM

The first rule ensures that the proposed variant \( n(s, c, v) \) is a natural number:

\[
\begin{align*}
A(s, c) \land I(s, c, v) & \land G(s, c, v, x) \\
\vdash & \\
n(s, c, v) \in \mathbb{N} & \\
\end{align*}
\]

NAT

The second rule, on all convergent events, ensures that they decrease the variant:

\[
\begin{align*}
A(s, c) \land I(s, c, v) & \land G(s, c, v, x) \land BA(s, c, v, x, v') \\
\vdash & \\
n(s, c, v') & < n(s, c, v) \\
\end{align*}
\]

VAR

B Proof rules from [MAV05]

B.1 Feasibility: FIS_REF

\[
\begin{align*}
P(s, c) & \land I(s, c, v) \land J(s, c, v, w) \land H(s, c, w) \\
\vdash & \\
\exists w'. S(s, c, w, w') & \\
\end{align*}
\]

FIS_REF
### B.2 Guard strengthening: GRD\_REF

\[
\begin{array}{c|c}
| P(s, c) \land I(s, c, v) \land J(s, c, v, w) \land H(s, c, w) & \text{GRD\_REF} \\
| \quad \vdash G(s, c, v) & \\
\end{array}
\]

### B.3 Simulation: INV\_REF

\[
\begin{array}{c|c}
| P(s, c) \land I(s, c, v) \land J(s, c, v, w) \land H(s, c, w) \land S(s, c, w, w') & \text{INV\_REF} \\
| \quad \vdash \exists v'. (R(s, c, v, v') \land J(s, c, v', w')) & \\
\end{array}
\]