A CSP account of Event-B refinement

Steve Schneider
Department of Computing, University of Surrey
S.Schneider@surrey.ac.uk

Helen Treharne
Department of Computing, University of Surrey
H.Treharne@surrey.ac.uk

Heike Wehrheim
Department of Computer Science, University of Paderborn
wehrheim@uni-paderborn.de

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 moreover possibly be anticipated or convergent. All such steps are accompanied with precise proof obligations. Still, it remains unclear what the exact relationship - in terms of a *behaviour-oriented semantics* - between an Event-B machine and its refinement is. In this paper, we give a CSP account of Event-B refinement, with a treatment for the first time of splitting events and of anticipated 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.

1 Introduction

Event-B [1] 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 framework allows for a great deal of flexibility as to cover a broad range of system developments. The recent book [1] comprising case studies from rather diverse areas shows that this goal is actually met. The flexibility is 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*. Anticipated and convergent events both introduce new details into the machine specification. Convergent events must not be executed forever, while for anticipated events this condition is deferred to later refinement steps. All of these steps come with precise proof obligations; appropriate tool support helps in discharging these [3, 2]. Event-B is essentially a *state-based* specification technique, and proof obligations therefore reason about predicates on states.

Like Event-B, CSP comes with a notion of refinement. In order to understand their relationship, these two refinement concepts need to be set in a single framework. Both formalisms moreover 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

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

To appear in EPTCS.

semantic domains of traces, failures and divergences. The aim of this paper is to give a precise account of Event-B refinement in terms of CSP's behaviour-oriented process refinement. This will also provide the underlying results that support refinement in the combined formalism Event-B‖CSP. Our work is thus in line with previous studies relating state-based with behaviour-oriented refinement (see e.g. [5, 9, 4]). 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 moreover also deals with the Event-B approach of anticipated events as a means to defer consideration of divergence-freedom. Our results involves support for individual refinement steps as well as for the resulting refinement chain.

The paper is structured as follows. The next section introduces the necessary background on Event-B and CSP. 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 Section 5 will then set these definitions in relation. It turns out that the appropriate refinement concept of CSP in this combination with Event-B is infinite-traces-divergences refinement. The last section concludes.

2 Background

We start with a short introduction to CSP and Event-B. For more detailed information see [17] and [1] respectively.

2.1 CSP

CSP, Communicating Sequential Processes, introduced by Hoare [11] 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 $a_1, a_2, \ldots$ or $evt0, evt1, \ldots$. 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* ($P_1 \,|||\, P_2$), 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 $\mathscr{U}$ in [16] or FDI in [17]. The semantics of a process can be understood in terms of four sets, T, F, D, I, 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 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. Failures will not 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. 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 traces which have some prefix which is a divergence.

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

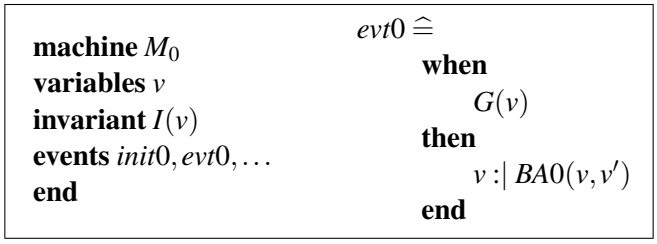

Figure 1: Template of an Event-B machine and an event.

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 only finitely many A events appear 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. As a first observation, we get the following.

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 2.3 *Follows immediately from the semantics of the hiding operator.*

Later, we furthermore use *specifications* on traces or, more generally, on CSP processes. Specifications are given in terms of predicates. If S is a predicate on a particular semantic element, then we write P **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 **sat** $S(u)$ is equivalent to $\forall u \in infinites(P) \,.\, S(u)$.

2.2 Event-B

Event-B [1, 13] 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 [1].

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. 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 $evt0, \ldots$ to update the state (see left of Fig.1). Initialisation is a special event $init0$.

A machine M_0 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).

3 CSP semantics for Event-B machine

Event-B machines are particular instances of action systems, so Morgan's CSP semantics for action systems [14] 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 [6] 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. 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.

Note that the notion of *traces* for machines is different to that presented in [1], where traces are considered as sequences of *states* rather than our treatment of traces as sequences of *events*.

The CSP semantics is based on the weakest precondition semantics of events. Let S be a statement (of an event). Then $[S]R$ denotes 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 by considering them as guarded commands:

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

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

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

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

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

Based on the weakest precondition, 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*:

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

Here, the weakest precondition on a sequence of events is the weakest precondition of the sequential composition of those events: $[\langle a_1, \ldots, a_n \rangle]P$ 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

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

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

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

Infinite Traces The technical definition of infinite traces is given in [6], 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).

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

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

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 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:

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

Refinement in Event-B also allows for the possibility of introducing new events. To capture this aspect in CSP, we need a way of incorporating this into process refinement. As a first idea, we could *hide* the new events in the refining process. This potentially introduces divergences, namely, when there is an infinite sequence of new events in the infinite traces. In order to separate out consideration of divergence from reasoning about traces, we will use $P \,|||\, RUN_N$ as a lazy abstraction operator instead. RUN_N defines a divergence free process capable of executing any order of events from the set N. This will enable us to characterise Event-B refinement introducing new events in CSP terms. The following lemma gives the relationship between refinement involving interleaving, and refinement involving hiding.

Lemma 4.1 *If $P_0 \,|||\, RUN_N \sqsubseteq_{TDI} P_1$ 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) $P_0 \,|||\, RUN_N \sqsubseteq_{TDI} P_1$, (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.

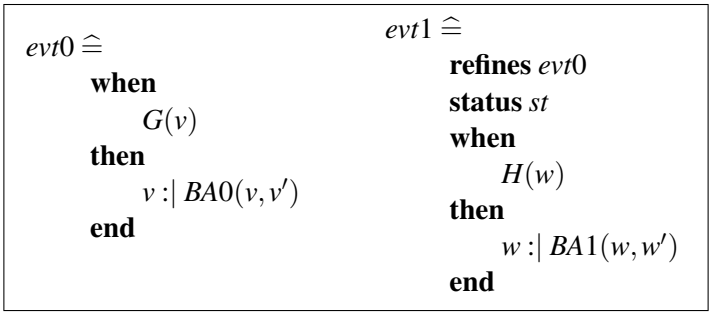

Figure 2: An event and its refinement

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

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

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

□

4.2 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 to take the following form:

machine M_0
variables v
invariant $I(v)$
events $init0, evt0, \ldots$
end

machine M_1
refines M_0
variables w
invariant $J(v,w)$
events $init1, evt1, \ldots$
variant $V(w)$
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, in the sense that any event of M_1 can be matched by an event of M_0 (or *skip* for newly introduced events) to maintain J. This is the standard notion of downwards simulation data refinement [8]. We next look at this in more detail, and in particular give the proof obligations associated to these conditions.

First of all, we need to look at events again. Figure 2 gives the shape of an event and its refinement. We see that an event in the refinement now also gets a *status*. The status can be ordinary (also called *remaining*), or *anticipated* or *convergent*. Convergent events are those which must not be executed forever, and anticipated events are those that will be made convergent at some later refinement step. New events must either have status anticipated or convergent. Both of these 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.

We now describe each of the proof obligations in turn. We have simplified them from their form in [13] by removing explicit references to sets and constants. Alternative forms of these proof obligations are given in [1, 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:

$$\frac{I(v) \wedge J(v,w) \wedge H(w) \vdash \exists w'.BA1(w,w')}{\textbf{FIS\_REF}}$$

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

$$\frac{I(v) \wedge J(v,w) \wedge H(w) \vdash G(v)}{\textbf{GRD\_REF}}$$

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). New events are treated as refinements of *skip*. The rule is:

$$\frac{I(v) \wedge J(v,w) \wedge H(w) \wedge BA1(w,w') \vdash \exists v'.(BA0(v,v') \wedge J(v',w'))}{\textbf{INV\_REF}}$$

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 respectively for all convergent and anticipated events, including all newly-introduced 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:

$$\frac{I(v) \wedge J(v,w) \wedge H(w) \wedge BA1(w,w') \vdash V(w) \in \mathbb{N} \wedge V(w') < V(w)}{\textbf{WFD\_REF (convergent event)}}$$

$$\frac{I(v) \wedge J(v,w) \wedge H(w) \wedge BA1(w,w') \vdash V(w) \in \mathbb{N} \wedge V(w') \leqslant V(w)}{\textbf{WFD\_REF (anticipated event)}}$$

We will use the refinement relation $M_0 \preccurlyeq M_1$ to mean that the four proof obligations *FIS_REF*, *GRD_REF*, *INV_REF*, and *WFD_REF* hold between abstract machine M_0 and concrete machine M_1.

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 remaining events R (neither anticipated nor convergent). The alphabet of M, the set of all possible events, is thus given by $\alpha M = A \cup C \cup R$. In the CSP refinement, these will take different roles.

Now consider an Event-B Machine M_0 and its refinement M_1: $M_0 \preccurlyeq M_1$. The machine M_0 has anticipated events A_0, convergent events C_0, and remaining events R_0, and M_1 similarly has event sets A_1, C_1, and R_1. Each event ev_1 in M_1 either refines a single event ev_0 in M_0 (indicated by the clause 'refines ev_0' in the description of ev_1) or does not refine any event of M_0. The set of new events N_1 is those events which are not refinements of events in M_0.

$M_0 \preccurlyeq M_1$ thus induces a partial surjective function $f_1 : \alpha M_1 \twoheadrightarrow \alpha M_0$ where $f_1(ev_1) = ev_0 \Leftrightarrow ev_1$ refines ev_0. Observe that αM_1 is partitioned by $f_1^{-1}(\alpha M_0)$ and N_1. The rules for refinement between events in Event-B impose restrictions on these sets:

1. each event of M_0 is refined by at least one event of M_1;
2. each new event in M_1 is either anticipated or convergent;
3. each event in M_1 which refines an anticipated event of M_0 is itself either convergent or anticipated;
4. refinements of convergent or remaining events of M_0 are remaining in M_1, i.e. they are not given a status.

The conditions imposed by the rules are formalised as follows:

1. $ran(f_1) = A_0 \cup C_0 \cup R_0$;
2. $N_1 \subseteq A_1 \cup C_1$;
3. $f_1^{-1}(A_0) \subseteq A_1 \cup C_1$;
4. $f_1^{-1}(C_0 \cup R_0) = f_1^{-1}(C_0) \cup f_1^{-1}(R_0) = R_1$.

These relationships between the classes of events are illustrated in Figure 3.

5.1 New events

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.

Lemma 5.1 *If $M_0 \preccurlyeq M_1$ and the refinement introduces new events N_1 and uses the mapping f_1, then $f_1^{-1}(M_0) \,|||\, RUN_{N_1} \sqsubseteq_{TDI} M_1$.*

Proof: We assume state variables of M_0 and M_1 named as given above, i.e. state variables of M_0 are v and of M_1 are w. Let $tr = \langle a_1, \ldots, a_n \rangle \in traces(M_1)$. We need to show that $tr \in traces(f_1^{-1}(M_0) \,|||\, RUN_{N_1})$. First of all note that the interleaving operator merges the traces of two processes together, i.e., the traces of $f_1^{-1}(M_0) \,|||\, RUN_{N_1}$ are simply those of $f_1^{-1}(M_0)$ with new events arbitrarily inserted. The proof proceeds by induction on the length of the trace.

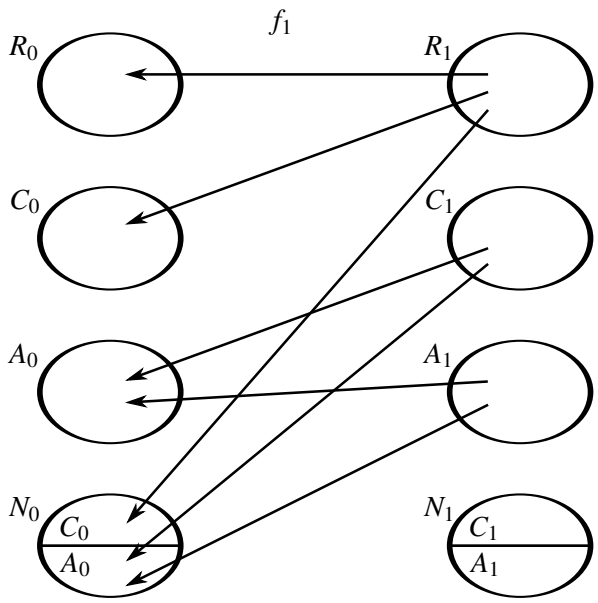

Figure 3: Relationship between events in a refinement step: f_1 maps events in M_1 to events in M_0 that they refine.

Induction base Assume $n = 0$, i.e., $tr = \langle\rangle$. By definition this means that the initialisation event *init*1 has been executed bringing the machine M_1 into a state w_1. By INV_REF (using init as event), we find a state v_1 such that $J(v_1, w_1)$ and furthermore $\langle\rangle \in traces(M_0)$ and hence also in $traces(f_1^{-1}(M_0) \,|||\, RUN_{N_1})$.

Induction step Assume that for a trace $tr = \langle a_1, \ldots, a_{j-1}\rangle \in traces(M_1)$ we have already shown that $tr \in traces(f_1^{-1}(M_0) \,|||\, RUN_{N_1})$ and this has led us to a pair of states v_{j-1}, w_{j-1} such that $J(v_{j-1}, w_{j-1})$. Now two cases need to be considered:

1. $a_j \notin N_1$: Assume a_j in M_1 to be of the form

   $$\textbf{when } H(w) \textbf{ then } w :| BA1(w, w') \textbf{ end}$$

   and $f_1(a_j)$ in M_0 of the form

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

   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 $BA1(w_{j-1}, w_j)$ we find – by INV_REF – a state v_j such that $J(v_j, w_j)$ and $BA(v_{j-1}, v_j)$. Hence $tr^\frown\langle a_j\rangle \in traces(f_1^{-1}(M_0) \,|||\, RUN_{N_1})$.
2. $a_j \in N_1$: Similar to the previous case. Here, a_j refines skip and thus $v_j = v_{j-1}$ and the event a_j is coming from RUN_{N_1}.

In the same way we can carry out a proof for infinite traces. For divergences it is even simpler as $divergences(M_1) = \{\}$. □

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*

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

for each i, then

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

Proof: Two successive refinement steps combine to provide a relationship between P_0 and P_2 of the same form as Line 1 above, as follows:

$$\begin{array}{rlll}
f_2^{-1}(P_1) \,|||\, RUN_{N_2} & \sqsubseteq_{TDI} & P_2 & \text{(given)} \\
f_2^{-1}(f_1^{-1}(P_0) \,|||\, RUN_{N_1}) \,|||\, RUN_{N_2} & \sqsubseteq_{TDI} & P_2 & \text{(line (1), transitivity of } \sqsubseteq) \\
f_2^{-1}(f_1^{-1}(P_0)) \,|||\, RUN_{f_2^{-1}(N_1)} \,|||\, RUN_{N_2} & \sqsubseteq_{TDI} & P_2 & \text{(Law: } f^{-1}(P \,|||\, Q) = f^{-1}(P) \,|||\, f^{-1}(Q)) \\
f_2^{-1}(f_1^{-1}(P_0)) \,|||\, RUN_{f_2^{-1}(N_1)\cup N_2} & \sqsubseteq_{TDI} & P_2 & \text{(Law: } RUN_A \,|||\, RUN_B = RUN_{A\cup B})
\end{array}$$

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

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 rules 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

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

Intuitively, this states that all infinite traces having infinitely many convergent (C) events also have infinitely many (R) remaining 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 R with $C \cap R = \{\}$. M* does not diverge on C events *if M* **sat** $CA(C,R)$.

Convergent events in Event-B machines only come into play during refinement. Thus a plain, single Event-B machine has no convergent events ($C = \{\}$) and thus trivially satisfies the specification predicate.

Lemma 5.4 *If $M_0 \preccurlyeq M_1$, and M_1 has convergent, anticipated, and remaining events C_1, A_1, and R_1 respectively, then M_1* **sat** $CA(C_1,R_1)$

Proof: We prove this by contradiction. Assume $\neg M_1$ **sat** $CA(C_1,R_1)$. Then there is some $u \in infinites(M_1)$ such that $\#(u \upharpoonright C_1) = \infty$ and $\#(u \upharpoonright R_1) < \infty$. Then there must be some tr_0, u' such that $u = tr_0 \frown u'$ with $u' \in (C_1 \cup A_1)^\omega$ (i.e. tr_0 is a prefix of u containing all the R_1 events). Moreover, $\#u' \upharpoonright C_1 = \infty$.

Now since $M_0 \preccurlyeq M_1$ 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 $M_0 \preccurlyeq M_1$ 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_1 or C_1 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_1) 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 $C, C', R \subseteq \alpha P$ nonempty finite sets of events.*

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 $P \,|||\, RUN_N$ **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. 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** $CA(C,R)$ we have $\#(f(u) \upharpoonright R) = \infty$ and hence $\#(u \upharpoonright f^{-1}(R)) = \infty$.
2. Let $u \in \mathit{infinites}(P \,|||\, RUN_N)$ 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** $CA(C,R)$ we get $\#((u \setminus N) \upharpoonright R) = \infty$, hence $\#(u \upharpoonright R) = \infty$.
3. Let $u \in \mathit{infinites}(P)$ such that $\#(u \upharpoonright (C \cup C')) = \infty$. Both C and C' are finite sets hence either $\#(u \upharpoonright C) = \mathit{infty}$ or $\#(u \upharpoonright C') = \infty$ (or both). In the first case we get $\#(u \upharpoonright R) = \infty$ by P **sat** $CA(C,R)$. In the second case it follows that $\#(u \upharpoonright (C \cup R)) = \infty$ and hence again $\#(u \upharpoonright C) = \infty$ or directly $\#(u \upharpoonright R) = \infty$.
4. First of all note that if P **sat** $CA(C,R)$ 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 $C \cap R = \{\}$ — $\#(u \upharpoonright R) \neq \infty$ which contradicts P **sat** $CA(C,R)$.

□

The most interesting of these properties is probably the last 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.

This gives us some results about the specification predicate for single Event-B machines and CSP processes. Next, we would like to apply this to refinements. First, we again consider just two machines.

Lemma 5.6 *Let $M_0 \preccurlyeq M_1$ with an associated refinement function f_1 and let M_0 **sat** $CA(C_0, R_0)$. Then M_1 **sat** $CA(f_1^{-1}(C_0) \cup C_1\ ,\ f_1^{-1}(R_0))$.*

Proof: Assume $u \in \mathit{infinites}(M_1)$ and $\#(u \upharpoonright (f_1^{-1}(C_0) \cup C_1) = \infty$. We aim to establish that $\#(u \upharpoonright f_1^{-1}(R_0)) = \infty$. We have $\#(u \upharpoonright f_1^{-1}(C_0)) = \infty$ or $\#(u \upharpoonright C_1) = \infty$.

In the former case, Lemma 5.1 yields that $f_1(u \upharpoonright f^{-1}(\alpha M_0)) \in \mathit{infinites}(M_0)$. Then

$$
\begin{array}{rl}
\#(u \upharpoonright f_1^{-1}(C_0)) = \infty & \text{(given)} \\
\#(f_1(u \upharpoonright f^{-1}(C_0)) \upharpoonright C_0) = \infty & \text{(since renaming preserves length)} \\
\#(f_1(u \upharpoonright f^{-1}(\alpha M_0)) \upharpoonright C_0) = \infty & \text{(since } C_0 \subseteq \alpha M_0\text{)} \\
\#(f_1(u \upharpoonright f^{-1}(\alpha M_0)) \upharpoonright R_0) = \infty & \text{(by } M_0 \textbf{ sat } CA(C_0, R_0)\text{)} \\
\#(u \upharpoonright f^{-1}(\alpha M_0)) \upharpoonright f^{-1}(R_0) = \infty & \text{(since renaming preserves length)} \\
\#(u \upharpoonright f_1^{-1}(R_0)) = \infty & \text{(since } R_0 \subseteq \alpha M_0\text{)}
\end{array}
$$

In the latter case Lemma 5.4 yields that $\#(u \upharpoonright R_1) = \infty$. Then

$$\begin{array}{rl} \#(u \upharpoonright R_1) = \infty & \\ \#(u \upharpoonright f_1^{-1}(R_0 \cup C_0)) = \infty & (\text{since } R_1 = f_1^{-1}(C_0 \cup R_0)) \\ \#(u \upharpoonright f_1^{-1}(R_0)) = \infty \vee \#(u \upharpoonright f_1^{-1}(C_0)) = \infty & \end{array}$$

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

□

Note that by Lemma 5.5 (4) the above result implies that the machine M_1 does not diverge on $f_1^{-1}(C_0) \cup C_1$, in particular $M_0 \setminus (f_1^{-1}(C_0) \cup C_1)$ is divergence-free.

Similar to the previous case, we can lift this to chains of refinement steps. Consider the last result with respect to two refinement steps $M_0 \preccurlyeq M_1 \preccurlyeq M_2$:

$$\begin{array}{rlll} M_0 & \textbf{sat} & CA(C_0, R_0) & (\text{given}) \\ f^{-1}(M_0) & \textbf{sat} & CA(f^{-1}(C_0), f^{-1}(R_0)) & (\text{lemma 5.5 (1)}) \\ f^{-1}(M_0) \,|||\, RUN_{N_1} & \textbf{sat} & CA(f^{-1}(C_0), f^{-1}(R_0)) & (\text{lemma 5.5 (2), since } f_1^{-1}(C_0) \cap N_1 = \{\}) \\ M_1 & \textbf{sat} & CA(f^{-1}(C_0), f^{-1}(R_0)) & (\text{lemma 5.1}) \\ f_2^{-1}(M_1) & \textbf{sat} & CA(f_2^{-1}(f^{-1}(C_0)), f_2^{-1}(f^{-1}(R_0))) & (\text{lemma 5.5 (1)}) \\ f_2^{-1}(M_1) \,|||\, RUN_{N_2} & \textbf{sat} & CA(f_2^{-1}(f^{-1}(C_0)), f_2^{-1}(f^{-1}(R_0))) & (\text{lemma 5.5 (2)}) \\ M_2 & \textbf{sat} & CA(f_2^{-1}(f^{-1}(C_0)), f_2^{-1}(f^{-1}(R_0))) & (\text{lemma 5.1}) \\ M_2 & \textbf{sat} & CA(C_2 \cup f_2^{-1}(C_1)\,,\, f_2^{-1}(R_1)) & (\text{lemma 5.6}) \end{array}$$

Then by applying Lemma 5.5(3) to the final two lines, with $R = f_2^{-1}(f_1^{-1}(R_0))$, $C = f_2^{-1}(f_1^{-1}(C_0))$, and $C' = C_2 \cup f_2^{-1}(C_1)$, we obtain

$$M_2 \quad \textbf{sat} \quad CA(C_2 \cup f_2^{-1}(C_1) \cup f_2^{-1}(f_1^{-1}(C_0))\,,\, f_2^{-1}(f_1^{-1}(R_0))$$

Thus if

$$M_0 \preccurlyeq M_1 \preccurlyeq \ldots \preccurlyeq M_n$$

then collecting together all the steps yields that

$$M_n \quad \textbf{sat} \quad CA((f_n^{-1}(\ldots f_1^{-1}(C_0)\ldots) \cup \ldots f_n^{-1}(C_{n-1}) \cup C_n)\,, f_n^{-1}(\ldots f_1^{-1}(R_0)\ldots)) \tag{2}$$

Finally, we would like to put together these results into one result relating the initial machine M_0 to the final 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 $M_n \setminus \{\mathit{new\ events}\}$ via infinite-traces-divergences refinement – show that Event-B refinement actually does not introduce divergences on new events. For such chains of refinement steps we always assume that $A_0 = C_0 = \{\}$ (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:

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

Then noting the disjointness of the union, by repeated application of

$$C_j \uplus A_j \uplus R_j \quad = \quad f_j^{-1}(C_{j-1} \uplus A_{j-1} \uplus R_{j-1}) \uplus N_j$$

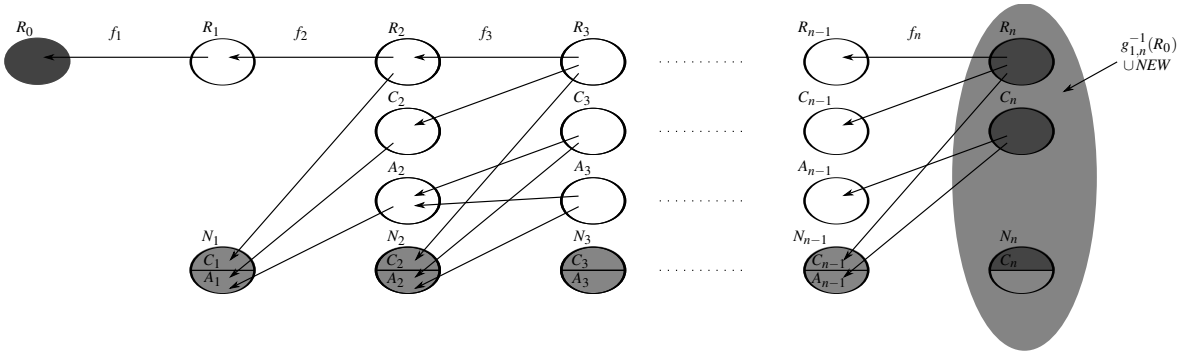

Figure 4: Constructing *NEW*

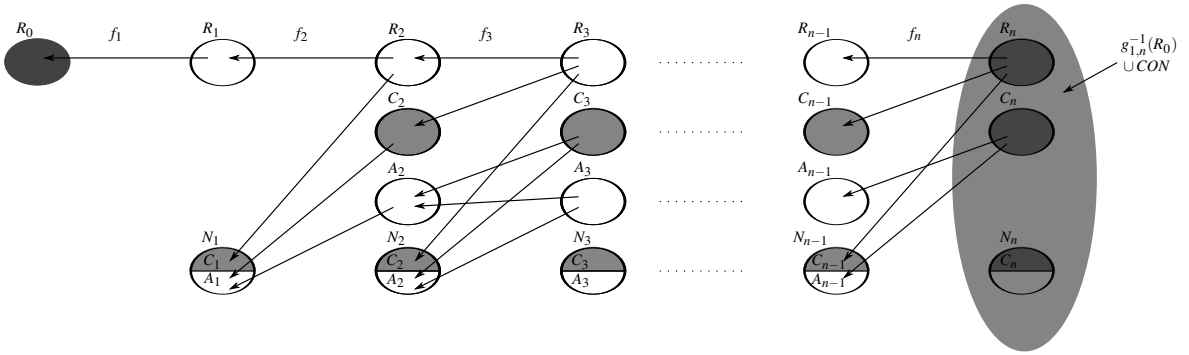

Figure 5: Constructing *CON*

we obtain

$$C_j \uplus A_j \uplus R_j \quad = \quad g^{-1}_{1,j}(C_0 \uplus A_0 \uplus R_0) \uplus g^{-1}_{2,j}(N_1) \uplus \ldots \uplus g^{-1}_{j,j}(N_{j-1}) \uplus N_j \tag{3}$$

Observe that this is a partition of $C_j \uplus A_j \uplus R_j$. Also, by repeated application of

$$R_j \quad = \quad f_j^{-1}(R_{j-1}) \uplus f_j^{-1}(C_{j-1})$$

we obtain

$$R_j \uplus C_j \quad = \quad g^{-1}_{1,j}(R_0) \uplus g^{-1}_{1,j}(C_0) \uplus g^{-1}_{2,j}(C_1) \uplus \ldots \uplus g^{-1}_{j,j}(C_{j-1}) \uplus C_j \tag{4}$$

Observe that this is a partition of $C_j \uplus R_j$.

In a full refinement chain $M_0 \preccurlyeq \ldots \preccurlyeq M_n$ we have that $A_0 = \{\}$,$C_0 = \{\}$, and $A_n = \{\}$. Define:

$$NEW \quad = \quad g^{-1}_{2,n}(N_1) \uplus \ldots \uplus g^{-1}_{n,n}(N_{j-1}) \uplus N_n$$

$$CON \quad = \quad g^{-1}_{1,n}(C_0) \uplus \ldots \uplus g^{-1}_{n,n}(C_{j-1}) \uplus C_n$$

These constructions are illustrated in Figures 4 and 5.

Then from Equation 3 above with $j = n$, and using $A_0 = C_0 = A_n = \{\}$ we obtain

$$C_n \uplus R_n \quad = \quad g^{-1}_{1,n}(R_0) \uplus NEW$$

From Equation 4 above with $j = n$ we obtain

$$C_n \uplus R_n \quad = \quad g_{1,n}^{-1}(R_0) \uplus CON$$

Hence $NEW = CON$. From Theorem 5.2 and Line (2) above respectively we obtain that

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

$$\text{and} \qquad M_n \textbf{ sat } CA(CON\,,\, f_n^{-1}(\ldots f_1^{-1}(R_0)\ldots)\,)$$

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

$$f_n^{-1}(\ldots(f_1^{-1}(M_0))\ldots) \quad \sqsubseteq_{TDI} \quad M_n \setminus NEW \tag{5}$$

or, equivalently, that the following theorem holds true.

Theorem 5.7 *Let $M_0 \preccurlyeq M_1 \preccurlyeq \ldots \preccurlyeq M_n$ be a chain of refinement steps such that $A_0 = C_0 = \{\}$ and $A_n = \{\}$, refining events according to functions f_i, and let NEW be the set of events as calculated above. Then*

$$M_0 \sqsubseteq_{TDI} f_1(f_2(\ldots f_n(M_n \setminus NEW)\ldots))$$

Proof: This follows from the result in Line 5 above, using the CSP law $f(f^{-1}(P)) = P$. □

This result guarantees that Event-B refinement (a) does neither introduce "new traces on old events" nor (b) does it introduce divergences on new events. This gives us the precise account of Event-B refinement in terms of CSP which we were aiming at.

6 Conclusion

In this paper, we have given a CSP account of Event-B refinement. The approach builds on Butler's semantics for action systems [6]. Butler's refinement rules allow new convergent events to be introduced into action systems, so that refinement steps satisfy $M_i \sqsubseteq_{TDI} (M_{i+1} \setminus N_{i+1})$, and hiding new events does not introduce divergence. Abrial's approach to Event-B refinement generalises this approach, allowing new events to be *anticipated* as well as *convergent*, and also allowing splitting of events. Our approach to refinement using CSP semantics reflects this generalisation and thus extends Butler's, in order to encompass these different forms of event treatment in Event-B refinement. We do not yet handle merging events, and this is the subject of current research.

Recently, an Event-B‖CSP approach has been introduced [19]. It aims to combine Event-B machine descriptions with CSP [17] control processes, in order to support a more explicit view of control. In this, it follows previous works on integration of formal methods [7, 22, 15, 18, 12], 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 [21]. 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 [20]. We aim to develop 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 [10].

References

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

[2] J-R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta & L. Voisin (2010): *Rodin: an open toolset for modelling and reasoning in Event-B*. *STTT* 12(6), pp. 447–466, doi:10.1007/s10009-010-0145-y.

[3] J-R. Abrial, M. J. Butler, S. Hallerstede & L. Voisin (2008): *A Roadmap for the Rodin Toolset*. In E. Börger, M. J. Butler, J. P. Bowen & P. Boca, editors: *ABZ*, *Lecture Notes in Computer Science* 5238, Springer, p. 347, doi:10.1007/978-3-540-87603-8.

[4] E. A. Boiten & J. Derrick (2009): *Modelling Divergence in Relational Concurrent Refinement*. In Michael Leuschel & Heike Wehrheim, editors: *IFM*, *Lecture Notes in Computer Science* 5423, Springer, pp. 183–199, doi:10.1007/978-3-642-00255-7.

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

[6] M. J. Butler (1992): *A CSP approach to Action Systems*. DPhil thesis, Oxford University.

[7] M. J. Butler (2000): *csp2B: A Practical Approach to Combining CSP and B*. In: *FACS*, pp. 182–196.

[8] J. Derrick & E. A. Boiten (2001): *Refinement in Z and Object-Z*. Springer-Verlag, doi:10.1007/978-1-4471-0257-1.

[9] J. Derrick & E.A. Boiten (2003): *Relational Concurrent Refinement*. *Formal Aspects of Computing* 15(2-3), pp. 182–214, doi:10.1007/s00165-003-0007-4.

[10] Formal Systems (Europe) Ltd.: *The FDR Model Checker*. `http://www.fsel.com/` (accessed 8/3/11).

[11] C.A.R. Hoare (1985): *Communicating Sequential Processes*. Prentice-Hall.

[12] A. Iliasov (2009): *On Event-B and Control Flow*. Technical Report CS-TR-1159, School of Computing Science, Newcastle University.

[13] C. Métayer, J.-R. Abrial & L. Voisin (2005): *Event-B Language*. RODIN Project Deliverable 3.2, `http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf`, accessed 25/5/10.

[14] C. Morgan (1990): *Of wp and CSP*. In: *Beauty is our business: a birthday salute to Edsger W. Dijkstra*, Springer, pp. 319–326.

[15] E-R. Olderog & H. Wehrheim (2005): *Specification and (property) inheritance in CSP-OZ*. *Sci. Comput. Program.* 55(1-3), pp. 227–257, doi:10.1016/j.scico.2004.05.017.

[16] A.W. Roscoe (1998): *Theory and Practice of Concurrency*. Prentice-Hall.

[17] S. Schneider (1999): *Concurrent and Real-time Systems: The CSP approach*. Wiley.

[18] S. Schneider & H. Treharne (2005): *CSP theorems for communicating B machines*. *Formal Asp. Comput.* 17(4), pp. 390–422, doi:10.1007/s00165-005-0076-7.

[19] S. Schneider, H. Treharne & H. Wehrheim (2010): *A CSP Approach to Control in Event-B*. In Dominique Méry & Stephan Merz, editors: *IFM*, *Lecture Notes in Computer Science* 6396, Springer, pp. 260–274, doi:10.1007/978-3-642-16265-7.

[20] S. Schneider, H. Treharne & H. Wehrheim (2011): *Bounded Retransmission in Event-B||CSP: a Case Study*. Technical Report CS-11-04, University of Surrey.

[21] S. Schneider, H. Treharne & H. Wehrheim (2011): *Stepwise refinement in Event-B||CSP*. Technical Report CS-11-03, University of Surrey.

[22] J. Woodcock & A. Cavalcanti (2002): *The Semantics of Circus*. In D. Bert, J. P. Bowen, M. C. Henson & K. Robinson, editors: *ZB*, *Lecture Notes in Computer Science* 2272, Springer, pp. 184–203. Available at `http://link.springer.de/link/service/series/0558/bibs/2272/22720184.htm`.