Under consideration for publication in Formal Aspects of Computing

External and Internal Choice with Event Groups in Event-B

Michael Butler

Electronics and Computer Science
University of Southampton, UK

Abstract.
Abrial's Event-B formalism for refinement-based system development is influenced by Back's action system approach. Morgan has defined a CSP-like failures-divergence semantics for action systems that distinguishes internal and external choice of actions. Morgan's semantics has the characteristic that the choice between enabled actions is external while internal choice is represented less directly through nondeterministic effect of actions. Practical experience with Event-B has demonstrated the need to be able to represent both internal and external choice between enabled events more explicitly. In this paper, Morgan's failures semantics for action systems is modified to allow both internal and external choice to be represented directly. This is achieved by grouping events so that external choice is between event groups and internal choice is within event groups. This leads to a refinement rule for preservation of choice between event groups while allowing for reduction of choice within event groups. We also provide a refinement rule for splitting event groups in order to increase external choice. The refinement rules are justified in terms of failures refinement.

1. Introduction

Abrial's Event-B is a formalism for refinement-based development of discrete event systems [Abr10]. Its deployment is supported by the Rodin toolset which includes proof obligation generation and verification through a collection of mechanical provers [ABH+10]. An Event-B machine consists of a collection of variables, invariants on those variables and a collection of guarded events that may update the machine variables. An Event-B development consists of a collection of machines linked by refinement and each event of a refined machine must either refine some event of its abstraction or must refine *skip*. Refinement between events is defined using the standard notion of refinement for programs based on preservation of a gluing invariant between abstract and concrete variables.

An action system consists of a collection of guarded atomic actions acting on some state variables as defined by Back [Bac90]. At each execution step, an enabled action is nondeterministically chosen. No distinction is made between whether the action to be executed is chosen internally or externally. Action system behaviour is modelled by the possible state traces a system may perform and refinement is defined semantically in terms of state trace inclusion: any trace of a refined model must be a trace of the abstract model [BvW94]. As in Event-B, the refinement proof rule for action systems requires that each action of a refined system refines some abstract action or refines *skip*. The refinement rule also requires that

Correspondence and offprint requests to: Michael Butler e-mail: mjb@ecs.soton.ac.uk. This work is supported by the FP7 DEPLOY Project (ICT 214158), www.deploy-project.eu

the disjunction of all abstract action guards entails the disjunction of the concrete guards under a gluing invariant.

Hoare's failures-divergence model for CSP distinguishes internal and external choice of events [Hoa85]. In this model, external choice must be preserved by refinement while internal choice may be reduced or may be made external through refinement. Morgan has defined a CSP-like failures-divergence semantics for action systems that distinguishes internal and external choice of actions [Mor90]. Morgan's semantics has the the characteristic that the choice between enabled actions is external while internal choice is represented less directly through the nondeterministic effect of actions. Woodcock and Morgan have defined proof rules for action system refinement that ensure failures-divergence refinement at the semantic level [WM90]. The enabledness preservation parts of those proof rules are more fine-grained than in Back's rule: they require that the guard of every abstract action should entail the guard of its corresponding refined action. This is stronger than Back's rule which only requires entailment between the disjunction of all abstract action and the disjunction of all refined events.

Although it is possible to model internal choice between events indirectly through nondeterministic actions, practical experience with Event-B has demonstrated the need to be able to represent both internal and external choice between enabled events more explicitly. We will illustrate this in Section 2 with an example of a database transaction whose completion is modelled by two events: *Update* and *Abort*. These two events represent alternative outcomes of a transaction. In an abstract model, both the *Update* and *Abort* events have the same guard. The reason for this is that it is convenient to abstract away from the cause of the transaction failure and to focus on the effect of success or failure at the specification level. The refined transaction model in Section 2 explicitly represents the occurrence of a fault and the guards of the *Update* and *Abort* events are strengthened accordingly. However, under the failures interpretation, this is not a refinement of the specification since the choice between *Update* and *Abort* is external in the abstract model whereas it is internal in the refined model. Clearly the choice should not be external in the abstraction either: if we abstract away from the cause of failure within the system, then we cannot offer the choice between success or failure to the external environment. It would be more natural to view the choice between the abstract completion events as internal. The transaction is representative of a range of existing case studies that have been undertaken in Event-B including an electronics funds transfer system [BY08], a replicated database system [YB06], multi-agent systems [BB09], a flash-based file system [DB09], a data management system for a satellite [FRB11] and a gear control system for a car [SB11]. All these cases involve treatment of faults with the specification focusing on the effect of success or failure and abstracting away from the causes of failure. Occurrence of failure is introduced in later refinements.

In this paper, Morgan's failures semantics for action systems is modified to allow both internal and external choice to be represented directly by taking account of event grouping. We also modify the enabledness-preservation part of the refinement rule of [WM90] so that there is entailment between disjunctions of event guards within groups. Our new refinement rule is shown to be sound w.r.t. failures refinement. Although the results presented in this paper are motivated by experience with Event-B, the results apply to action systems in general. We use Event-B as the notation in practice but what is important from the point of view of this paper is that we treat events[1] as weakest precondition predicate transformers. It is possible to treat events as relations in order to give machines a failures-divergence semantics as found in Josephs [Jos88] and He [He89]. An advantage of predicate transformers over relations is that they provide a uniform treatment of nontermination; termination is embodied in the *wp* semantics and does not need to be treated explicitly, for example, using a special *bottom* observation. Having said that, in this paper we will not be concerned with nontermination and will focus instead on the distinction between internal and external choice. Thus we will only work with failures of Event-B machines but not divergences. Recently Schneider et al [STW11] have given a CSP semantics directly to Event-B in a similar style to [Mor90] that treats traces and divergences but not failures.

It is worth pointing out a further more recent contribution by Back and von Wright, that of distinguishing demonic and angelic choice in action systems [BvW00]. This is very elegant work that defines angelic choice as the dual of demonic choice. While demonic choice is the same as CSP internal choice, angelic choice is more general than CSP external choice: angelic choice can be widened in refinement and the angel will avoid a non-terminating choice if possible. Nondeterminism in Event-B is demonic so that treating angelic and demonic choice would require a major change to the language. As we will demonstrate, treating CSP-style

[1] Back uses the term 'action' while Abrial uses the term 'event'.

internal and external choice can easily be accommodated in Event-B through finessing of the enabledness-preservation proof obligations.

None of the existing work on giving a CSP semantics to action systems/Event-B supports the explicit treatment of internal choice within an event group; that is the novel contribution of this paper. After presenting motivating examples in the Section 2, we outline Morgan's existing definition of action system failures in Section 3. Section 4 presents the new definition of failures using event groups and proves some validating theorems about the definition. Section 5 presents a revised data refinement rule that takes account of event groups while Section 6 presents a simple rule showing that syntactically splitting event groups is a valid refinement step.

Author's note: the work in [Mor90] served as the starting point for my own DPhil thesis [But92] (supervised by Carroll Morgan at Oxford). A number of extensions were made to the semantic definitions and refinement of action systems in my DPhil: better treatment of unbounded nondeterminism through the definition of an infinite traces model, treatment of internal events and convergence, treatment of parameterised actions that distinguishes input and output parameters and treatment of synchronised parallel composition of action systems. These issues will not be addressed in this paper however. Besides providing the starting points for my DPhil, Carroll always encouraged me to follow his typical style of presenting proofs in a clear calculational way (following Dijkstra) and I have endeavoured to follow that style here.

2. Motivating Examples

The distinction between internal and external choice is illustrated by the two Event-B models of drinks vending machines in Fig. 1. In the machine $VM1$, initially the only event enabled is the *Coin* event. Execution of the *Coin* event leads to a state where control variable $m1 = vend$ and thus both the *Tea* and *Coffee* events are enabled. Using the failures-divergence semantics of Morgan, the choice between these events would be external, that is, the choice between *Tea* and *Coffee* would be resolved by the environment of $VM1$. In the second machine, $VM2$, the control variable $m2$ is nondeterministically assigned either the value *tea* or the value *coffee* when the *Coin* event is executed. Execution of the *Coin* event will either lead to a state in which *Tea* is enabled or to a state in which *Coffee* is enabled. Under the failures semantics, we assume that the environment of an Event-B machine observes the execution of the machine events but not the values of the machine variables. Thus the environment cannot directly observe or control the nondeterministic effect on variable $m2$ of the *Coin* event in $VM2$. This gives rise to internal choice between *Tea* or *Coffee*; that is, the choice between offering the *Tea* event or the *Coffee* event to the environment is made internally by $VM2$ and the environment has no control over that choice.

It is possible to show that each event of $VM2$ refines its corresponding event in $VM1$. In addition, $VM2$ *does* preserve overall enabledness of the events, i.e., we can construct a gluing invariant in which the disjunction of the abstract guards entails the disjunction of the concrete guards. However $VM2$ *does not* preserve the enabledness of individual events, i.e., it is not possible to construct a gluing invariant in which the guard of *Tea* in $VM1$ entails the guard of *Tea* in $VM2$ (and similarly for the *Coffee* events). This is as expected since $VM2$ does not retain the external choice between *Tea* and *Coffee* and is therefore not a failures refinement of $VM1$. From the point of view of a customer using a vending machine, $VM1$ is a better machine than $VM2$ since $VM1$ allows the customer to choose between tea and coffee whereas $VM2$ chooses the drink for the customer nondeterministically. Thus it is appropriate here to use a notion of refinement in which $VM2$ is not a valid refinement of $VM1$.

Although it is possible to model internal choice between events indirectly through nondeterministic actions, as illustrated by $VM2$ of Fig. 1, practical experience with Event-B has demonstrated the need to be able to represent both internal and external choice between enabled events more explicitly. This is exemplified by the transaction system in the machines of Fig. 2. The left-hand machine, $Transaction1$, is a simple model of a transaction that might succeed or abort. In the case that the transaction succeeds (*Update* event), the database is updated. In the case that the transaction aborts (*Abort* event), the database remains unchanged. In $Transaction1$, both the *Update* and *Abort* events have the same guard. This is because we abstract away from the cause of the transaction failure and focus on the effect of success or failure. In the machine $Transaction2$ of Fig. 2, the flag f is introduced to represent occurrence of a fault and the guards of the *Update* and *Abort* events are strengthened accordingly. However, under the failures interpretation, $Transaction2$ is not a refinement of $Transaction1$ since the choice between *Update* and *Abort* is external in $Transaction1$ whereas it is internal in $Transaction2$. It would be more natural to view the choice between

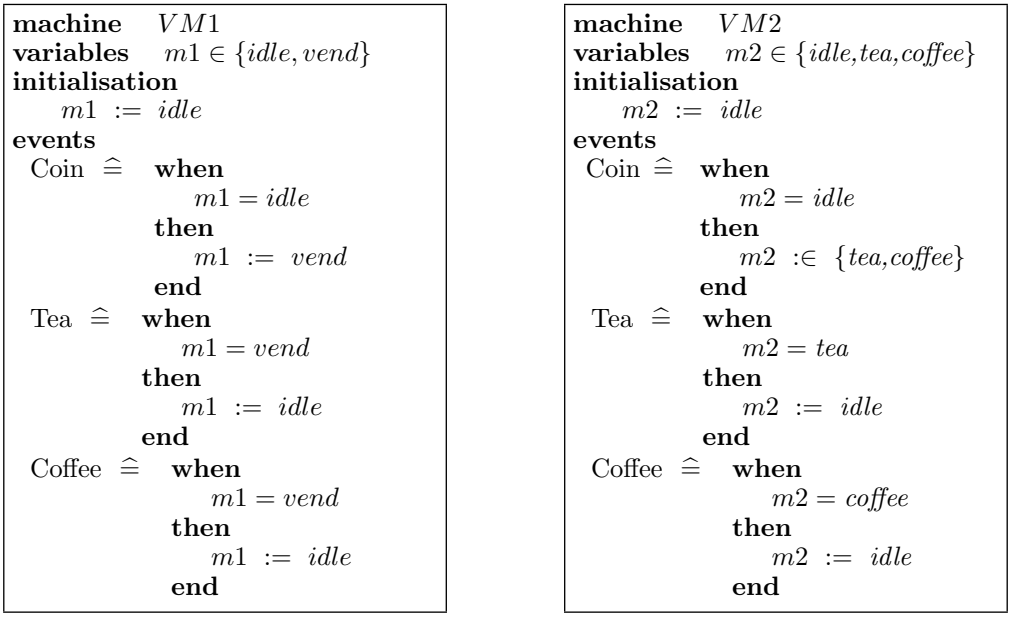

Fig. 1. Simple vending machines with external (VM1) and internal (VM2) choice

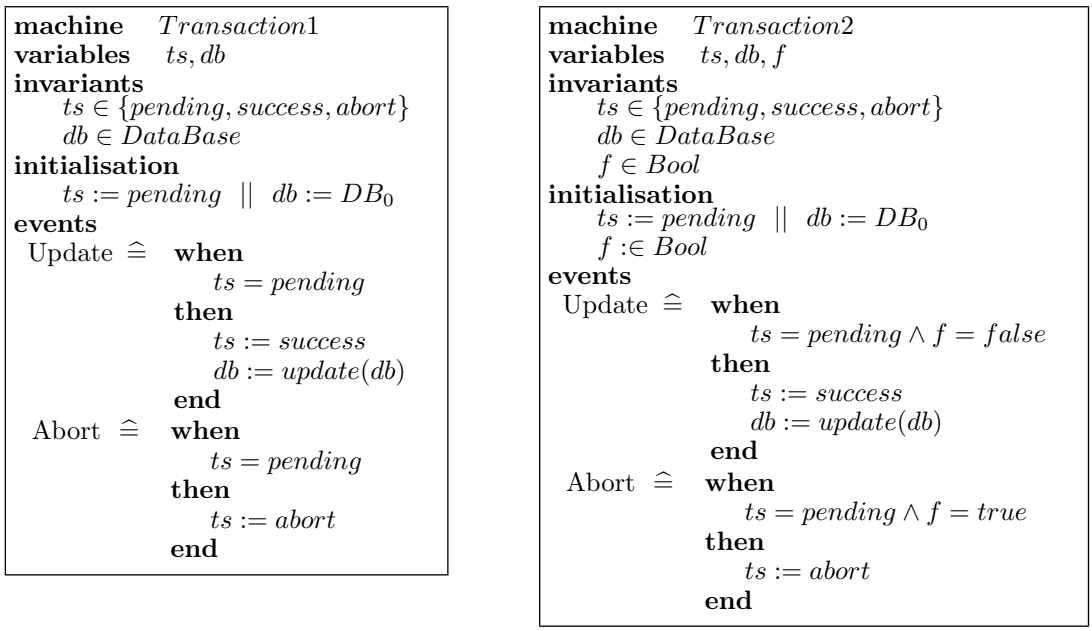

Fig. 2. Simple transaction system and its refinement in Event-B

the completion events in *Transaction*1 as internal as well. So while the failures semantic model is appropriate for the vending machine example, it seems to be less appropriate for the transaction example.

One solution would be to use the enabledness preservation rule of Back for the transaction example which involves entailment between disjunctions of guards. This would work for the example of Fig. 2: we can prove that the disjunction of the guards of *Update* and *Abort* in the abstraction entails the disjunction of the concrete guards. However that would lose more than we want. We still wish to retain external choice for other events. For example, if the transaction model also included an event for reading the database, then we would like to specify that the environment can chose between attempting to update the database or reading

the database. The solution we adopt is to syntactically group the *Update* and *Abort* events together and to place the *Read* event into a separate group. Our interpretation is that the choice between enabled events in different groups is external whereas the choice between enabled events within a group is internal.

3. Background definitions

We outline the CSP failures model of Hoare as well as Morgan's definition of the failures of an action system. In CSP, a failure is a pair consisting of an event trace and a set of refusal events. If A is the alphabet of events that a system M can engage in, then a failure of M is a pair of the form (s, X) where $s \in A^*$ is a finite trace of events and $X \subseteq A$ is a refusal set. The interpretation of the pair is that M may engage in the trace of events s after which it may refuse to engage in any of the events in the refusal set X. For example, $VM1$ of Fig. 1 can refuse $Coin$ immediately after it executes $Coin$ thus it containts the failure $(\langle Coin\rangle, \{Coin\})$. $VM1$ can refuse neither *Tea* nor *Coffee* after *Coin* thus it does *not* contain the failures $(\langle Coin\rangle, \{Tea\})$ nor $(\langle Coin\rangle, \{Coffee\})$. Because of internal choice, $VM2$ may refuse *Tea* after *Coin*. $VM2$ has the failures,

$$(\langle Coin\rangle, \{Tea\}) \quad and \quad (\langle Coin\rangle, \{Coffee\}).$$

$VM2$ does *not* have the failure $(\langle Coin\rangle, \{Tea, Coffee\})$ since it cannot refuse both *Tea* and *Coffee* after *Coin*.

The event labels (e.g., *Coin*, *Tea*, *Coffee*) are used to construct the traces and the refusal sets. We write A for the (finite) set of labels of the events of machine M and write $wp_M(a, Q)$ for the weakest precondition guaranteeing that the event with label $a \in A$ of machine M will establish postcondition Q. In Event-B, an event operates on a list of machine variables v. An event labelled a from machine M has a canonical form defined in terms of a guard and a before-after predicate as follows [Abr10]:

$$a \;\hat{=}\; \textbf{when } G(v) \textbf{ then } v :| \; BA(v, v') \textbf{ end}.$$

The weakest precondition of this canonical form is,

$$wp_M(a, Q) \;\hat{=}\; G(v) \Rightarrow (\forall v' \cdot BA(v, v') \Rightarrow Q[v'/v]).$$

Event-B has a *feasibility* proof obligation that requires that $BA(v, v')$ is satisfiable for some v' whenever $G(v)$ holds.

Similar to [Mor90], we write $wp_M(s, Q)$ for the weakest precondition of the sequential composition of the events of M labelled by trace s We write $t; s$ for concatenation of traces and have that:

$$wp_M(\ s; t,\ Q) \;=\; wp_M(s, wp_M(t, Q)).$$

Traces and failures represent possible behaviour whereas weakest preconditions represent guaranteed behaviour, i.e, the postcondition is guaranteed no matter what path is followed during execution of a non-deterministic event. To represent possible behaviour, Morgan defined the conjugate weakest precondition as follows:

$$\overline{wp}_M(s, Q) \;\hat{=}\; \neg wp_M(s, \neg Q).$$

$\overline{wp}_M(s, Q)$ represents the weakest precondition under which execution of s *might* lead to a state satisfing Q. We write $grd_M(a)$ for the guard of an event $a$of machine M and say that a is enabled whenever it is possible to reach some state by executing a, thus,

$$grd_M(a) \;\hat{=}\; \overline{wp}(a, true).$$

The feasibility obligation of Event-B means that for the canonical form of an event, $grd_M(a) = G(v)$.

For any set of labels $X \subseteq A$, we write $grd_M(X)$ for the *disjunction* of the guards of actions labelled from X. We write i for the special initialisation event of M. We write F_M for the set of failures of M and, following Morgan [Mor90], a pair (s, X) is a failure of machine M as follows:

$$(s, X) \in F_M \quad \hat{=} \quad \overline{wp}_M(\ i; s,\ \neg grd_M(X)).$$

That is, (s, X) is a failure of M if it is possible for M to execute the sequence of events $i; s$ and reach a state in which none of the events in X is enabled.

4. The new definition of failures

As explained in the introduction, we want to group events in order to specify that the choice between enabled events within a group is internal rather than external. As outlined in the previous section, the definition of failures in [Mor90] means that an event may be refused when it is not enabled. We want to broaden the condition under which an event may be refused to take account of other events in the same event group. For example, assume that the *Update* and *Abort* events of the abstract transaction model, $Transaction1$, were to be grouped together, then it should be possible for *Update* to be refused either when *Update* is disabled or when *Abort* is enabled. More generally, assume that a is part of a group of events G, then the condition under which a may be refused are,

$$\neg grd_M(a) \ \vee \ grd_M(G \setminus \{a\}).$$

That is, a may be refused when a is disabled or when some event other than a in the same group G is enabled. This condition can be re-written as an implication:

$$grd_M(a) \ \Rightarrow \ grd_M(G \setminus \{a\}).$$

In order to define the failures, we need to find the refusal condition for a set of events X. We do this by factoring X into its groups so that $X \cap G$ will be those events of X that are in a group G while $G \setminus X$ will be the events of G that are not in X. Generalising the above refusal condition for a single event to the events of X from G gives the following:

$$grd_M(X \cap G) \ \Rightarrow \ grd_M(G \setminus X).$$

Experience with proofs similar to those presented later in this paper led to the observation that this can be re-written as follows:

$$\begin{array}{lll} & grd_M(X \cap G) \Rightarrow grd_M(G \setminus X) & \\ = & (grd_M(X \cap G) \Rightarrow grd_M(G \setminus X)) \ \wedge \ (grd_M(G \setminus X) \Rightarrow grd_M(G \setminus X)) & \textit{tautology} \\ = & (grd_M(X \cap G) \vee grd_M(G \setminus X)) \ \Rightarrow \ grd_M(G \setminus X) & \textit{factorisation} \\ = & grd_M(G) \ \Rightarrow \ grd_M(G \setminus X) & (X \cap G) \cup (G \setminus X) = G \end{array}$$

The condition $grd_M(G) \ \Rightarrow grd_M(G \setminus X)$ is easier to reason with since X only appears once.

We assume that a machine M has a set of event group labels, written grp_M, that the set of events in a group labelled $g \in grps_M$ is given by $evts_M(g)$ and that every event in A is in at least one group. For example, if we grouped the *Tea* and *Coffee* events of $VM1$ in Fig. 1, then we would have,

$$grp_{VM1} = \{G_1, G_2\}, \qquad evts_{VM1}(G_1) = \{Coin\}, \qquad evnts_{VM1}(G_2) = \{Tea, Coffee\}.$$

When setting out on the work described in this paper our intention was that the events would be partitioned by the groups, i.e., the event groups are disjoint, as this conforms to the usage we encountered in practice. However, disjointness is not required for any of the proofs that we do. The only requirement is that every event is in at least one group (otherwise it will always be refused according to our definition).

We are now in a position to give a more precise definition for the refusal condition for a set X:

Definition 1. For Event-B machine M with event alphabet A and event set $X \subseteq A$, the refusal predicate of X, written $ref_M(X)$ is defined as follows:

$$ref_M(X) \ \hat{=} \ \bigwedge_{g \in grp_M} grd_M(evts_M(g)) \ \Rightarrow \ grd_M(evts_M(g) \setminus X).$$

This in turn leads to our new definition of machines failures:

Definition 2. For Event-B machine M with event alphabet A, $(s, X) \in A^* \times \mathbb{P}(A)$ is a failure of M, written F_M, as follows:

$$(s, X) \in F_M \quad \hat{=} \quad \overline{wp}_M(\ i; s, \ ref_M(X)\ ).$$

To illustrate the definition of ref_M, we calculate some refusal predicates for example $VM1$ (Fig.1) with event groups $\{Coin\}$ and $\{Tea, Coffee\}$; that is, the choice between *Tea* and *Coffee* is internal since they are placed in the same group. The conditions under which the set $\{Tea\}$ is refused is calculated as follows:

$$
\begin{array}{lll}
 & ref(\{Tea\}) & \\
= & (\ grd(\{Coin\}) \Rightarrow grd(\{Coin\})\ ) \ \wedge \ (\ grd(\{Tea, Coffee\}) \Rightarrow grd(Coffee)\ ) & \textit{Definition 1} \\
= & true \ \wedge \ (\ (m1 = vend \vee m1 = vend) \Rightarrow m1 = vend\ ) & \textit{Fig.1, definition of VM1} \\
= & true &
\end{array}
$$

Thus it is always possible for $VM1$ to refuse $\{Tea\}$ (and similary $\{Coffee\}$). However $VM1$ can refuse both *Tea* and *Coffee* only when it is not in the vending state:

$$
\begin{array}{ll}
 & ref(\{Tea, Coffee\}) \\
= & (\ grd(\{Coin\}) \Rightarrow grd(\{Coin\})\ ) \ \wedge \ (\ grd(\{Tea, Coffee\}) \Rightarrow grd(\{\})\ ) \\
= & true \ \wedge \ (\ (m1 = vend \vee m1 = vend) \Rightarrow false\ ) \\
= & m1 \neq vend
\end{array}
$$

Finally it is never possible for $VM1$ to refuse all three events:

$$
\begin{array}{ll}
 & ref(\{Coin, Tea, Coffee\}) \\
= & (\ grd(\{Coin\}) \Rightarrow grd(\{\})\ ) \ \wedge \ (\ grd(\{Tea, Coffee\}) \Rightarrow grd(\{\})\ ) \\
= & (\ m1 = idle \Rightarrow false\ ) \ \wedge \ (\ (m1 = vend \vee m1 = vend) \Rightarrow false\ ) \\
= & m1 \neq idle \ \wedge \ m1 \neq vend \\
= & false
\end{array}
$$

Hoare has defined several well-formedness conditions for the failures model of CSP as follows [Hoa85]:

$$(\langle\rangle, \{\}) \in F \tag{1}$$

$$(s; t, X) \in F \ \Rightarrow \ (s, \{\}) \in F \tag{2}$$

$$(s, X) \in F \ \wedge \ Y \subseteq X \ \Rightarrow \ (s, Y) \in F \tag{3}$$

$$(s, X) \in F \ \wedge \ a \in A \ \wedge \ a \notin X \ \Rightarrow \ (s, X \cup \{a\}) \in F \ \vee \ (s; a, \{\}) \in F \tag{4}$$

To validate his definition of failures, Morgan shows that that these conditions follow as theorems from the definition [Mor90]. We prove the same for our new definition. Proving Conditions (1) and (2) is independent of our definition of ref_M so we concentrate on Conditions (3) and (4). For Condition (3) we will use the following lemma that follows easily from the fact that $grd_M(X)$ is defined as a disjunction:

Lemma 1. For $Y \subseteq X$, we have that $grd_M(Y) \Rightarrow grd_M(X)$.

Next we prove the following lemma concerning closure of refusal conditions:

Lemma 2. For $Y \subseteq X$, we have that $ref_M(X) \Rightarrow ref_M(Y)$.

Proof:

$$
\begin{array}{lll}
 & ref_M(X) & \\
= & \bigwedge g \in grp_M \cdot grd_M(G) \Rightarrow grd_M(G \setminus X) & \textit{Definition 1, writing } G \textit{ for } evts_M(g) \\
\Rightarrow & \bigwedge g \in grp_M \cdot grd_M(G) \Rightarrow grd_M(G \setminus Y) & \textit{by Lemma 1 and } (G \setminus X) \subseteq (G \setminus Y) \\
= & ref_M(Y) & \textit{Definition 1}
\end{array}
$$

Theorem 1. $(s, X) \in F_M \ \wedge \ Y \subseteq X \ \Rightarrow \ (s, Y) \in F_M$.

Proof:

$$
\begin{array}{lll}
 & (s, X) \in F_M & \\
= & \overline{wp}_M(\ i; s,\ ref_M(X)\ ) & \textit{Definition 2} \\
\Rightarrow & \overline{wp}_M(\ i; s,\ ref_M(Y)\ ) & \textit{by Lemma 2 , } Y \subseteq X \textit{ and monotonicity of } \overline{wp} \\
= & (s, Y) \in F_M & \textit{Definition 2}
\end{array}
$$

Proof that Condition (4) of the failures model is satisfied requires a case split. We proceed with the theorem and its proof and then introduce a required lemma:

Theorem 2. $(s, X) \in F_M \ \wedge \ a \in A \ \wedge \ a \notin X \ \Rightarrow \ (s, X \cup \{a\}) \in F_M \ \vee \ (s; a, \{\}) \in F_M$.

Proof:

$$
\begin{array}{lll}
& (s, X) \in F_M & \\
= & \overline{wp}_M(\ i; s,\ ref_M(X)\ ) & \textit{Definition 2} \\
= & \overline{wp}_M(\ i; s,\ ref_M(X) \wedge (\neg grd_M(a) \vee grd_M(a))\ ) & \textit{introduce cases} \\
= & \overline{wp}_M(\ i; s,\ ref_M(X) \wedge \neg grd_M(a)\ ) \ \vee \ \overline{wp}_M(\ i; s,\ ref_M(X) \wedge grd_M(a)\ ) & \overline{wp}\ \textit{is disjunctive} \\
\Rightarrow & \overline{wp}_M(\ i; s,\ ref_M(X \cup \{a\})\ ) \ \vee \ \overline{wp}_M(\ i; s,\ ref_M(X) \wedge grd_M(a)\ ) & \textit{Lemma 3} \\
\Rightarrow & \overline{wp}_M(\ i; s,\ ref_M(X \cup \{a\})\ ) \ \vee \ \overline{wp}_M(\ i; s,\ \overline{wp}_M(a, true)\ ) & grd_M(a) = \overline{wp}_M(a, true) \\
= & (s, X \cup \{a\}) \in F_M \ \vee \ (s; a, \{\}) \in F_M & \textit{Definition 2}
\end{array}
$$

Lemma 3. For $a \in A$, $a \notin X$ we have that $ref_M(X) \wedge \neg grd_M(a) \ \Rightarrow \ ref_M(X \cup \{a\})$.

$$
\begin{array}{lll}
& ref_M(X) \wedge \neg grd_M(a) & \\
= & (\bigwedge g \in G \cdot grd_M(G) \Rightarrow grd_M(G \setminus X)) \ \wedge \ \neg grd_M(a) & \textit{Definition 1, writing } G \textit{ for } evts_M(g) \\
\Rightarrow & \bigwedge g \in grp_M \cdot grd_M(G) \ \Rightarrow \ (grd_M(G \setminus X) \ \wedge \ \neg grd_M(a)) & \textit{Logic}
\end{array}
$$

Case $a \in G$:

$$
\begin{array}{lll}
= & \bigwedge g \in grp_M \cdot grd_M(G) \ \Rightarrow \ ((grd_M((G \setminus (X \cup \{a\})) \cup \{a\}) \ \wedge \ \neg grd_M(a)) & a \in G,\ a \notin X \\
= & \bigwedge g \in grp_M \cdot grd_M(G) \ \Rightarrow \ ((grd_M(G \setminus (X \cup \{a\})) \vee grd_M(a)) \ \wedge \ \neg grd_M(a)) & \textit{Defn of } grd_M \\
\Rightarrow & \bigwedge g \in grp_M \cdot grd_M(G) \ \Rightarrow \ grd_M(G \setminus (X \cup \{a\})) & \textit{Logic} \\
= & ref_M(X \cup \{a\}) &
\end{array}
$$

Case $a \notin G$ is trivial to prove since in that case $G \setminus X = G \setminus (X \cup \{a\})$.

It is important to bear in mind that because of nondeterminism in events, a single sequence of events can give rise to multiple execution paths. For example, the event sequence *i;Coin* in *VM*2 (Fig. 1) may result in variable $m2 = tea$ or $m2 = coffee$. Condition $wp_M(\ i; s,\ Q\ )$ holds when *all* execution paths of the sequence $i; s$ result in a state satisfying Q. This includes the case where $i; s$ has no execution paths, i.e., is not a trace in the failures model. Morgan's definition of system failures (Section 3) means that if an event is guaranteed to be enabled via all execution paths of a sequence $i; s$, then that event cannot be refused after that trace:

$$
\begin{array}{lll}
wp_M(\ i; s,\ grd_M(a)\ ) & = & \neg\ \overline{wp}_M(\ i; s,\ \neg grd_M(a)\ ) \\
& = & (s, \{a\}) \notin F_M \qquad \textit{by Morgan's definition}
\end{array}
$$

If two events a and b are guaranteed to be enabled after $i; s$ and $i; s$ is a trace, then neither a nor b can be refused after $i; s$, i.e., the choice between a and b after $i; s$ is external. The equivalent result with our definition is at the level of event groups rather than individual events. If G is the set of all events in a single group, then it can be shown that $ref_M(G) = \neg grd_M(G)$, thus,

$$
\begin{array}{lll}
wp_M(\ i; s,\ grd_M(G)\ ) & = & \neg\ \overline{wp}_M(\ i; s,\ \neg grd_M(G)\ ) \\
& = & \neg\ \overline{wp}_M(\ i; s,\ ref_M(G)\ ) \\
& = & (s, G) \notin F_M \qquad \textit{by our definition}
\end{array}
$$

That is, if execution of event sequence $i; s$ is guaranteed to lead to a state in which some event in G is enabled, then G cannot be refused after s. By Property (3) above, if $(s, G) \notin F_M$ then no set larger than G can be refused after s either. So, as expected, our definition means that the choice *between* event groups is external.

5. Data refinement and event groups

Data refinement is the standard development technique in Event-B and proving data refinement requires the use of a gluing invariant linking abstract and concrete variables. Abrial formulates the proof obligation for

refinement in Event-B as a condition on the canonical form of events [Abr10]. Assume we are data-refining abstract event a_M operating on abstract variables v by concrete event a_N operating on concrete variables w. The respective events have the following canonical form:

$$a_M \;\hat{=}\; \textbf{when } G(v) \textbf{ then } v :| S(v, v') \textbf{ end}$$
$$a_N \;\hat{=}\; \textbf{when } H(w) \textbf{ then } w :| R(w, w') \textbf{ end}$$

The modeller needs to provide a gluing invariant $I(v, w)$ that relates the abstract and concrete variables and verify the following proof obligation for each pair of corresponding abstract and refined events:

$$I(v, w) \;\wedge\; H(w) \;\wedge\; R(w, w') \;\Rightarrow\; G(v) \;\wedge\; \exists v' \cdot S(v, v') \wedge I(v', w') \tag{5}$$

In this paper, we assume a one-to-one correspondence between abstract and concrete events. In general, in Event-B, each abstract event may be refined by one or more concrete events and there may be concrete events that refine *skip*.

Abrial also defines an enabledness-preservation obligation, similar to Back's condition on guards, requiring that the disjunction of abstract guards, $grd_M(A)$, entails the disjunction of concrete guards, $grd_N(A)$, under the gluing invariant:

$$I(v, w) \;\wedge\; grd_M(A) \;\Rightarrow\; grd_N(A) \tag{6}$$

We use a different formulation of event refinement in terms of wp (as found in work on refinement such as [GM91, Mor89, von94]). Rather than having a gluing predicate I, we use a special kind of predicate transformer called a *representation transformer* that maps predicates on v to predicates on w. Given representation transformer rep, data refinement is formulated as follows:

$$rep(wp_M(a, Q)) \;\Rightarrow\; wp_N(a, rep(Q)), \quad \textit{for all postconditions } Q \textit{ on } v \tag{7}$$

If $rep(Q)$ is defined as $(\exists v \cdot I(v, w) \wedge Q)$ then it can be shown that (5) entails (7) [GM91, Mor89, von94]. This definition of rep is disjunctive which is an assumption we will exploit below. We use a representation transformer rather than gluing predicate as it is algebraically simpler and thus simplifies our proofs, i.e., we use $rep(Q)$ rather than $(\exists v \cdot I(v, w) \wedge Q)$.

In the CSP failures model, refinement is defined as failures inclusion, that is, M is refined by N when $F_N \subseteq F_M$. Simulation is a standard technique for verifying data refinement between state-based systems. Woodcock and Morgan define forwards and backwards simulation rules for action system that are shown to be sound and jointly complete in the failure-divergences model [WM90]. Here we focus on the forwards simulation rule, modifying it slightly to take account of the event groupings. The motivation for introducing rep in [GM91] is that it captures both forwards and backwards simulation in a single definition; disjunctive rep corresponds to forwards simulation while conjunctive rep corresponds to backwards simulation. However, since we assume that rep is disjunctive, we are restricting our treatment of refinement to forwards simulation. Our forward simulation rule is as follows:

Definition 3. Let machines M and N both have the same event alphabet A and the same event groupings $(grp_M = grp_N)$ and let that rep be a disjunctive representation transformer linking the states of M and N. M is forward simulated by N under rep provided:

(i) $wp_M(i, Q) \;\Rightarrow\; wp_N(i, rep(Q)), \quad$ *for initialisation event* i*, all postconditions* Q *on* v
(ii) $rep(wp_M(a, Q)) \;\Rightarrow\; wp_N(a, rep(Q)), \quad$ *for all* $a \in A$*, all postconditions* Q *on* v
(iii) $rep(grd_M(\; evt_M(g)\;)) \;\Rightarrow\; grd_N(\; evt_M(g)\;) \quad$ *for all* $g \in grp_M$

As outlined above, Conditions (i) and (ii) in this rule are entailed by the data refinement rule in Event-B. Condition (iii) concerns enabledness preservation between the guards of corresponding event groups. This third condition is the essential difference between our simulation rule and those of the forward simulation rule of [WM90] and Abrial's rule represented by (6) above. Our rule is in between both of those: in [WM90], the enabledness obligation is at the level of individual events rather than event groups while (6) treats all events as a single group.

Our simulation rule allows us to verify that $VM1$ is refined by $VM2$ (Fig. 1) provided *Tea* and *Coffee* are in the same event group. It also allows us to verify that $Transaction1$ is refined by $Transaction2$ (Fig. 2) provided *Update* and *Abort* are in the same event group.

We formulate and prove the soundness of our simulation rule w.r.t. our failures semantics:

Theorem 3. If M is forward simulated by N under rep, then $F_N \subseteq F_M$.

Proof of this theorem will use the following lemma:

Lemma 4. If M is forward simulated by N under rep, then for all $s \in A^*$

$$wp_M(\ i; s,\ Q\ ) \ \Rightarrow\ wp_N(\ i; s,\ rep(Q)\ ).$$

This lemma is easily proved using induction over traces and Conditions (i) and (ii) of Definition 3.

Proof of Theorem 3 is by contrapositive and proceeds as follows:

$$\begin{array}{lll}
 & (s, X) \notin F_M & \\
= & wp_M(\ i; s,\ \neg ref_M(X)\ ) & \textit{Definition 2} \\
\Rightarrow & wp_N(\ i; s,\ rep(\neg ref_M(X))\ ) & \textit{Lemma 4 with } \neg ref_M(X) \textit{ for } Q \\
\Rightarrow & wp_N(\ i; s,\ \neg ref_N(X)\ ) & \textit{Lemma 5} \\
\Rightarrow & (s, X) \notin F_N & \textit{Definition 2}
\end{array}$$

Lemma 5. If M is forward simulated by N under rep, then for any $X \subseteq A$:

$$rep(\neg ref_M(X)) \ \Rightarrow\ \neg ref_N(X).$$

Proof:

$$\begin{array}{lll}
 & rep(\neg ref_M(X)) & \\
= & rep(\ \neg \bigwedge g \in grp_M \cdot grd_M(G) \Rightarrow grd_M(G \setminus X)\ ) & \textit{Definition 1, writing } G \textit{ for } evts_M(g) \\
= & \bigvee g \in grp_M \cdot rep(\ \neg(grd_M(G) \Rightarrow grd_M(G \setminus X))\ ) & \textit{rep is disjunctive} \\
\Rightarrow & \bigvee g \in grp_M \cdot rep(grd_M(G)) \ \wedge\ rep(\neg grd_M(G \setminus X)) & \textit{Logic, rep is monotonic} \\
\Rightarrow & \bigvee g \in grp_M \cdot grd_N(G) \ \wedge\ rep(\neg grd_M(G \setminus X)) & \textit{Condition (iii) of simulation rule} \\
\Rightarrow & \bigvee g \in grp_M \cdot grd_N(G) \ \wedge\ \neg grd_N(G \setminus X) & \textit{Lemma 7} \\
= & \neg \bigwedge g \in grp_M \cdot grd_N(G) \Rightarrow grd_N(G \setminus X) & \textit{Logic} \\
= & \neg ref_N(X) & \textit{Definition 1, } grp_M = grp_N
\end{array}$$

Note that the proof of this lemma makes use of group enabledness preservation (Condition (iii) of the simulation rule) but also makes use of the fact that refinement between events entails guard strengthening (Lemma 7) which is the opposite direction to Condition (iii). According to our definition of the refusal predicate, an abstract event a may be refused when some other event b in its group is enabled. For the concrete model to be a valid failures refinement, a should not be refused in more cases than in the abstract model thus b should not be more enabled in the concrete model. Event refinement (Condition (ii) of simulation) ensures that the concrete version of b is not more enabled (Lemma 6):

Lemma 6. If M is forward simulated by N under rep, then we have for $a \in A$:

$$rep(\neg grd_M(a)) \ \Rightarrow\ \neg grd_N(a)$$

Proof:

$$\begin{array}{lll}
 & rep(\neg grd_M(a)) & \\
= & rep(wp(a_M, false)) & \textit{Definition of } grd_M(a) \\
\Rightarrow & wp(a_N, rep(false)) & \textit{Condition (ii) of simulation rule} \\
= & wp(a_N, false) & \textit{rep is disjunctive so } rep(false) = false \\
= & \neg grd_N(a) & \textit{Definition of } grd_N(a)
\end{array}$$

We use this to prove Lemma 7:

Lemma 7. If M is forward simulated by N under rep, then for any $Y \subseteq A$, we have:

$$rep(\neg grd_M(Y)) \ \Rightarrow\ \neg grd_N(Y)$$

Proof:

$$
\begin{array}{lll}
 & rep(\neg grd_M(Y)) & \\
= & rep(\neg \bigvee a \in Y \cdot grd_M(a)) & \textit{Definition of } grd_M(Y) \\
= & rep(\bigwedge a \in Y \cdot \neg grd_M(a)) & \textit{Logic} \\
\Rightarrow & \bigwedge a \in Y \cdot rep(\neg grd_M(a)) & \textit{rep is monotonic} \\
\Rightarrow & \bigwedge a \in Y \cdot \neg grd_N(a) & \textit{Lemma 6} \\
\Rightarrow & \neg \bigvee a \in Y \cdot grd_N(a) & \textit{Logic} \\
= & \neg grd_N(Y) & \textit{Definition of } grd_N(Y)
\end{array}
$$

6. Refinement by group subsetting

In the previous section, we presented a simulation rule that required verifying preservation of enabledness between corresponding event groups. In this section, we show that another, very simple, method of machine refinement is to break event groups into smaller groups. This has the effect of converting internal choice into external choice. For example, suppose we started with a version of *VM1* (Fig. 1) with two event groupings,

$$G1 = \{Coin\}, \quad G2 = \{Tea, Coffee\}$$

so that the choice between *Tea* and *Coffee* is internal. Let us refer to this as *VM1a*. Now we construct another version, *VM1b* say, where the group $G2$ is split so *VM1b* has event groups,

$$H1 = \{Coin\}, \quad H2 = \{Tea\}, \quad H3 = \{Coffee\}$$

and now the choice between *Tea* and *Coffee* is external. We would expect that *VM1a* is refined by *VM1b*. We present a theorem that shows that such a rearrangement of event groupings that increases external choice is always a valid refinement. In practice, we would expect to partition groups but disjointness is not a necessity.

We say that machines M and N are *equivalent modulo event groupings* if they differ only in the way events are grouped so that their variables and events are the same. We assume the re-mapping of groups from M to N is defined by a function $gmap \in grp_M \rightarrow \mathbb{P}(grp_N)$. For the example above,

$$gmap(G1) = \{H1\}, \quad gmap(G2) = \{H2, H3\}.$$

The following definition characterises the conditions under which such a re-mapping will preserve enabledness:

Definition 4. Assume M and N are equivalent modulo their event grouping. A re-mapping of event groups of M to N, $gmap \in grp_M \rightarrow \mathbb{P}(grp_N)$, is *enabledness preserving* if it satisfies the following conditions:

(i) $(\bigcup g \in grp_M \cdot gmap(g)) \;=\; grp_N$
(ii) $(\bigcup h \in gmap(g) \cdot evt_N(h)) \;=\; evt_M(g), \qquad$ for all $g \in grp_M$

Re-mapping the event groupings using an enabledness-preseving *gmap* is refinement-preserving under the failures model:

Theorem 4. If *gmap* is an enabledness-preserving remapping of event groups from M to N, then $F_N \subseteq F_M$.

Proof: Since M and N are equivalent modulo their event grouping, it is easy to show that $wp_N(i; s, Q) = wp_M(i; s, Q)$ for any trace s and predicate Q (proof omitted). The proof then continues as follows:

$$
\begin{array}{lll}
 & (s, X) \in F_N & \\
= & \overline{wp}_N(\ i; s,\ ref_N(X)\ ) & \textit{Definition 2} \\
= & \overline{wp}_M(\ i; s,\ ref_N(X))\ ) & wp_N(i; s, Q) = wp_M(i; s, Q) \\
\Rightarrow & \overline{wp}_M(\ i; s,\ ref_M(X))\ ) & \textit{Lemma 8} \\
= & (s, X) \in F_M & \textit{Definition 2}
\end{array}
$$

The main impact of the re-mapping of groups is that ref_M and ref_N are different. Instead ref_M and ref_N are related as follows:

Lemma 8. If M and N are equivalent modulo their event grouping and *gmap* is an enabledness-preserving remapping of event groups, then for any $X \subseteq A$, we have:

$$ref_N(X) \;\Rightarrow\; ref_M(X)$$

$$\begin{array}{lll}
& ref_N(X) & \\
= & \bigwedge h \in grp_N \cdot grd_N(evt_N(h)) \Rightarrow grd_N(evt_N(h) \setminus X) & \textit{Definition of } ref_N(X) \\
= & \bigwedge g \in grp_M \cdot \bigwedge k \in gmap(g) \cdot grd_N(evt_N(k)) \Rightarrow grd_N(evt_N(k) \setminus X) & \textit{Condition (i)} \\
\Rightarrow & \bigwedge g \in grp_M \cdot (\bigvee k \in gmap(g) \cdot grd_N(evt_N(k))) \;\Rightarrow & \\
 & \qquad (\bigvee k \in gmap(g) \cdot grd_N(evt_N(k) \setminus X)) & \textit{Logic} \\
= & \bigwedge g \in grp_M \cdot grd_N(\bigcup k \in gmap(g) \cdot evt_N(k)) \;\Rightarrow & \\
 & \qquad grd_N(\; (\bigcup k \in gmap(g) \cdot evt_N(k) \setminus X) \;) & \textit{Definition of } grd_M \\
= & \bigwedge g \in grp_M \cdot grd_N(\bigcup k \in gmap(g) \cdot evt_N(k)) \;\Rightarrow & \\
 & \qquad grd_N(\; (\bigcup k \in gmap(g) \cdot evt_N(k)) \;\setminus X) & \setminus \textit{ distributes through } \cup \\
= & \bigwedge g \in grp_M \cdot grd_M(evt_M(g)) \;\Rightarrow\; grd_M(evt_M(g) \setminus X) & \textit{Condition (ii), } grd_M = grd_N \\
= & ref_M(X) & \textit{Definition of } ref_M(X)
\end{array}$$

7. Concluding

We introduced the notion of event groups for the Event-B formalism in order to distinguish internal choice of enabled events from external choice. At the semantic level, this distinction comes from the CSP failures model and was applied to Event-B by modifying Morgan's definition of the failures model for action systems. External choice should be maintained by refinement while internal choice may be reduced. The refinement theorems give us what we had set out to achieve: a semantic justification for a proof rule that requires enabledness preservation between groups of events (rather than between all events at one extreme or between individual events at the other extreme) as well as a justification for re-mapping of event groups.

An attraction of the proof rules is that they fit well with the existing proof rules for Event-B as embodied in the Rodin tool. Although there is no single behavioural semantic model for Event-B, the basic proof rules for event refinement and enabledness preservation apply to several semantic models including sequential and concurrent programs. This is explored further by Hallerstede in [Hal11].

There are several concepts in Event-B for which we have yet to explore how they relate to event groups with internal choice. These include new events in a refinement, machine composition and event parameterisation. The ability to distinguish explicitly between internal and external choice should be especially useful for distinguishing input and output parameters of events. As is the case in CSP, the choice of value for input parameters is best treated as external choice: the environment chooses the value of an input and the choice of possible input values should be maintained through refinement. Where there is more than one choice of value for an output parameter in a particular state, then the choice is best treated as internal: the environment does not choose the value of a parameter output to it and the choice of output value may be reduced (though not removed completely) in refinement.

References

[ABH+10] J.-R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010.

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

[Bac90] R.-J.R. Back. Refinement calculus II: Parallel and reactive systems. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume LNCS 430. Springer–Verlag, 1990.

[BB09] E. Ball and M. Butler. Event-B patterns for specifying fault-tolerance in multi-agent interaction. In *Methods, Models and Tools for Fault Tolerance*, volume 5454 of *LNCS*, pages 104–129. Springer, 2009.

[But92] M. Butler. *A CSP Approach To Action Systems.* D.Phil. Thesis, Programming Research Group, Oxford University, 1992.

[BvW94] R.-J.R Back and J. von Wright. Trace refinement of action systems. In *CONCUR*, volume 836 of *LNCS*, pages 367–384. Springer, 1994.

[BvW00] R.-J.R Back and J. von Wright. Contracts, games, and refinement. *Inf. Comput.*, 156(1-2):25–45, 2000.

[BY08] M. Butler and D. Yadav. An incremental development of the Mondex system in Event-B. *Formal Asp. Comput.*, 20(1):61–77, 2008.

[DB09] K. Damchoom and M. Butler. Applying event and machine decomposition to a flash-based filestore in Event-B. In *SBMF 2009*, volume 5902, pages 134–152. Springer LNCS, 2009.

[FRB11] A.S. Fathabadi, A. Rezazadeh, and M. Butler. Applying atomicity and model decomposition to a space craft system in Event-B. In *NASA Formal Methods*, volume 6617 of *LNCS*, pages 328–342. Springer, 2011.

[GM91] P.H.B. Gardiner and C.C. Morgan. Data refinement of predicate transformers. *Theoretical Computer Science*, 87:143–162, 1991.

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

[He89] J. He. Process refinement. In J. McDermid, editor, *The Theory and Practice of Refinement*. Butterworths, 1989.

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

[Jos88] M.B. Josephs. A state-based approach to communicating sequential processes. *Distrib. Comput.*, 3:9–18, 1988.

[Mor89] J.M. Morris. Laws of data refinement. *Acta Informatica*, 26:287–308, 1989.

[Mor90] C.C. Morgan. Of wp and CSP. In W.H.J. Feijen, A.J.M. van Gasteren, D. Gries, and J. Misra, editors, *Beauty is our business: a birthday salute to Edsger W. Dijkstra*. Springer–Verlag, 1990.

[SB11] M.R. Sarshogh and M. Butler. Specification and refinement of discrete timing properties in Event-B. In *AVoCS 2011*, September 2011.

[STW11] S. Schneider, H. Treharne, and H. Wehrheim. A CSP Account of Event-B Refinement. In *Refine*, volume 55 of *EPTCS*, pages 139–154, 2011.

[von94] J. von Wright. The lattice of data refinement. *Acta Informatica*, 31(2):105–135, 1994.

[WM90] J.C.P. Woodcock and C.C. Morgan. Refinement of state-based concurrent systems. In D. Bjørner, C.A.R. Hoare, and H. Langmaack, editors, *VDM '90*, volume LNCS 428. Springer–Verlag, 1990.

[YB06] D. Yadav and M. Butler. Rigorous design of fault-tolerant transactions for replicated database systems using Event B. In *RODIN Book*, volume 4157 of *LNCS*, pages 343–363. Springer, 2006.