Working Paper Series
ISSN 1170-487X

LSB - Live and Safe B
Alternative semantics for Event B

Steve Reeves and David Streader

Working Paper: 08/2006
July 10, 2006

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

LSB - Live and Safe B
Alternative semantics for Event B

Steve Reeves and David Streader
University of Waikato,Hamilton, New Zealand
{dstr,stever}@cs.waikato.ac.nz

July 10, 2006

Abstract

We define two lifted, total relation semantics for Event B machines: Safe B for safety-only properties and Live B for liveness properties. The usual Event B proof obligations, Safe, are sufficient to establish Safe B refinement. Satisfying Safe plus a simple additional proof obligation ACT_REF is sufficient to establish Live B refinement. The use of lifted, total relations both prevents the ambiguity of the unlifted relational semantics and prevents operations being clairvoyant.

Keywords: process refinement, Event B, Live B, Safe B, LSB

1 Introduction

An Event B tool will generate proof obligations that, when satisfied, are sufficient to guarantee the correctness of a machine refinement. But Event B has a partial correctness semantics that can model only safety properties, and although on the surface it seems that Event B could be extended to model liveness properties by the addition of simple proof obligations, this we show not to be the case.

The problem arises because of the underlying semantic model. It is well-known from Spivey's work on Z [1] that to use partial relations to model what is called total correctness in the state-based world and liveness in the event-based world, can result in operations being clairvoyant when operations may be guarded. This introduction of clairvoyant operations has also been found by Nelson [2] when extending Dijkstra's calculus (with a weakest precondition semantics). In [2] Nelson states "we require that the implementation of our commands be clairvoyant enough to find a computation that succeeds". We consider clairvoyance more thoroughly in Section 1.1.

In this paper we will define two lifted, total relation semantics for Event B:

Safe B semantics gives a partial correctness interpretation, as does Event B. We show that Event B's normal proof obligations, here called Safe, imply refinement defined on the new Safe B semantics;

Live B semantics gives a total correctness interpretation and differs from Safe B semantics in how the relation is totalised. Satisfying the proof obligations Safe plus one additional proof obligation is sufficient to imply refinement on Live B semantics, i.e.refinement on the total correctness semantics.

In Section 2 we review Event B semantics with the slightly generalised form of operations found in [3]. We motivate our definition of refinement by reference to a **natural** notion of refinement (Section 2.4) that is common in the literature.

In Section 3 we review the close relation between state-based formalisms and event-based formalisms and formalise the previously mentioned natural notion of refinement in Definition 3. In Section 4 we define our Safe B semantics and show that applying the event-based definition of refinement gives the expected results. In Section 5 we define our Live B semantics and show that applying the event-based definition of refinement gives the result that we would expect from the event-based literature.

We illustrate our alternative semantics with a simple example system where the operations of one version, C, are indeed refinements of the operations of another version, A. We then use this example to show that had we given the partial relations a total correctness interpretation then, quite at odds with our expectations, the machine C is not a refinement of A (using the previously given natural notion of refinement).

A consequence of this is that simply adding an additional proof obligation while using the usual partial relation semantics of Event B would result in operations behaving *clairvoyantly* (see next section) and refinement not giving the results we expect.

1.1 Clairvoyant operations

The first reference we have found to clairvoyant operations is in Nelson's work [2] extending Dijkstra's calculus. The semantics used by both Nelson and Dijkstra is not based on partial relations but on weakest preconditions. What the partial relation and weakest precondition semantics have in common is that neither have an explicit representation of nontermination.

If we compose the two operations in Fig. 1 as relations, we get the result shown. The point here is that W ; U is defined to always take us from a to a. So, somehow, W must "know" not to allow the move from a to b, which leads us into being blocked, i.e. into a state which U cannot get us back to a from. But this means that W must "know" that it will pass on "control" to U.

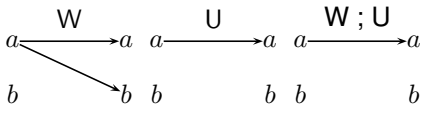

Figure 1: Operation W and U and their clairvoyant composition

Hence we say that W must be "clairvoyant" because it must be able to see into the future (of its uses). Indeed, sequencing in general will require clairvoyance to work according to relational composition. This we take to be completely unacceptable.

The problem illustrated by the two operations W and U was referred to by Spivey as giving operations a meaning that *"differs from the meaning that would be be natural in a programming language"*[1, p136].

It is well known that partial relation semantics are ambiguous in that the region outside the domain of the relation can be interpreted in several ways: as guarded [4],

as chaos (persistently undefined) [5] or as undefined [6].

Further, relations with no explicit representation of nontermination, whether partial or total, are ambiguous. They can be interpreted as either:

partial correctness semantics (see [7, 8]) - the relation defines the behaviour **only if** the operation terminates, or;

total correctness semantics (see [5, 1, 9]) - the relation defines when the operation terminates and what state it terminates in.

Under the partial correctness interpretation W and U (Fig. 1) do not behave clairvoyantly as W ; U quite naturally specifies that if W ; U terminates then it will terminate in state a. The clairvoyance only occurs if a total correctness interpretation is applied.

2 B and Event B Machines and semantics

In this section we briefly review B [9] and Event B [8] machines and their semantics. To be *well-formed* the machine must satisfy some tool generated "proof obligations". B and Event B tools also automatically generate proof obligations that are *sufficient* to imply refinement.

In B [9] operations are given a partial relation semantics. The domain is partitioned into three disjoint sets. The states that do not satisfy the pre-condition Pre of the operation are *undefined*. The states that satisfy Pre can be either pre-states not related to any post-state, referred to as *magic* states, or pre-states that are related to a post-state, referred to as *active* states.

Event B was introduced in *Extending B without changing it* [10] to model operations that could be guarded in the process algebraic sense. In Event B operations (now called events) cannot be undefined on any part of their domain. Like Dunne and Conroy [3], however, we permit both guards G and Pre to condition generalised substitutions R as there is no theoretical reason to limit the operations to being totally defined:

$$E \triangleq \textbf{PRE } Pre \textbf{ SELECT } G \textbf{ THEN } R \textbf{ END}$$

In this paper the states that do not satisfy the pre-condition Pre of the operation are *undefined*, characterised by $\neg Pre$. The states that satisfy Pre can be either pre-states not related to any post-state, referred to as *blocked* states characterised by $Pre \wedge \neg G$, or pre-states that are related to a post-state, referred to as *active* states characterised by $Pre \wedge G$.

When **SELECT** does not appear, assume $G = TRUE$. Similarly when **PRE** does not appear, assume $Pre = TRUE$. We add a generalised substitution $stop$ with before and after predicate $FALSE$ as sugar for:

$$stop \triangleq \textbf{SELECT } FALSE \textbf{ THEN } skip \textbf{ END}$$

Event B specifies only safety properties. Thus doing nothing satisfies any specification because it "does nothing wrong".

2.1 Set theoretic semantics

The usual B syntax for generalised substitutions defines three predicates. The first defines a relation R and the two others define subsets of the pre-states: G is true of the states in $domain(R)$ in which the operation is not blocked; and Pre is true of the states in $domain(R)$ for which the operation is defined.

Let S be restricted to evaluations that satisfy the invariant, $S \triangleq \{v|I(v)\}$. The partial relation $\mathsf{rel} \subseteq S \times S$ for "extended" Event B operation A can be split into three parts, based on combinations of the predicates I, Pre and G. The first is the *active* part:

$$\mathsf{a_A} \triangleq \{v \mapsto v' | I(v) \wedge G(v) \wedge Pre(v) \wedge R(v, v')\}$$

All three predicates hold in $\mathsf{a_A}$. The second part $\mathsf{m_A}$ is not active, so G does not hold but Pre does still hold for it, so it is the *blocked* part of the relation (and so contributes nothing in the partial relation rel above):

$$\mathsf{m_A} \triangleq \emptyset.$$

The third part, states for which Pre does not hold, is the *undefined* part:

$$\mathsf{u_A} \triangleq \{v \mapsto x | I(v) \wedge \neg Pre(v)\}.$$

States of $\mathsf{u_A}$ are related to all other states (x is unconstrained, though from S). The relational semantics of A is the union of these three parts:

$$\mathsf{rel_A} \triangleq \mathsf{a_A} \cup \mathsf{m_A} \cup \mathsf{u_A}$$

This relational semantics is ambiguous (Section 1.1). It can be interpreted as giving either a partial correctness semantics or a total correctness semantics. Usually the interpretation is stated informally, i.e. not as part of the formal semantics, and also is usually fixed in the initial discussion and never changed. But we are interested in both interpretations, and even in using a set of operations where some are to have one interpretation and the rest to have the other interpretation. We wish to formally capture the semantic difference between total and partial correctness. To do this we will lift and totalise the partial relations in different ways, as we shall see.

2.2 Proof obligations of Event B machines

An Event B machine is initialised to start in the states for which init holds. The following three proof obligations constitute a *well-formedness* condition on Event B machines:

$\mathsf{init}(v) \Rightarrow I(v)$	INIT
$I(v) \wedge G(v) \wedge Pre(v) \Rightarrow \exists v'.R(v, v')$	FIS
$I(v) \wedge G(v) \wedge Pre(v) \wedge R(v, v') \Rightarrow I(v')$	INV

Proof obligation INIT guarantees that the initialisation of a machine must satisfy its invariant. FIS guarantees that a, the active part of rel, is a total relation and finally INV guarantees that the postcondition of any operation must satisfy the machine invariant.

That $G(v) \wedge Pre(v) \subseteq domain(R(v, v'))$ is guaranteed by the FIS proof obligation.

All Event B machines must be well-formed, i.e. must satisfy these three proof obligations, prior to the refinement of the machine. Let $\mathbb{B}$ be the set of all well-formed machines.

2.3 Refinement

Refinement of the Event B operation A (see Fig. 2) is given by

$$\vdash G_A \Rightarrow \mathsf{A} \sqsubseteq_E \mathsf{C} \triangleq \mathsf{rel_C} \subseteq \mathsf{rel_A}.$$

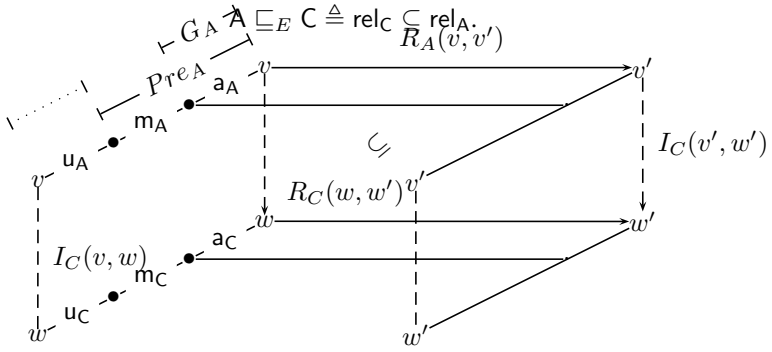

Figure 2: Refinement $\mathsf{A} \sqsubseteq_E \mathsf{C} \iff \mathsf{rel_C} \subseteq \mathsf{rel_A}$

Machine refinement has been defined in various ways in the literature and small changes in a definition can have far reaching consequences.

In Event B [8] machine refinement is defined as a property quantified over all operations whereas in B [9] machine refinement is defined as a property quantified over all programs. The review of the event-based testing semantics in Section 3 provides an answer to the obvious question: what is a good definition of refinement?

2.4 A natural but informal definition of refinement

Although informal the following definition of refinement is very natural and appears in many places in the literature [7, 5, 9, 11, 12, 4]:

> The concrete machine $\mathsf{C} \in \mathbb{B}$ is a refinement of an abstract machine $\mathsf{A} \in \mathbb{B}$ where no *user* of A could observe if they were given C in place of A.

The user of Event B machine A first places it in an Event B **context** $\mathsf{x} \in \mathbb{B}$. This we write as $[\![\mathsf{A}]\!]_{\mathsf{x}}$. The user then **observes** the whole system, M, built from A and x. Thus to formalise this we need to define both Event B contexts and observations.

As B machines can be used only by programs, the contexts in which a B machine can be placed are only programs. Consequently it is natural to define B machine refinement by quantifying over all programs.

Event B machines, unlike B machines, are not used by programs, they are used by other machines. This is also true of the event-based definition of processes that appears in CSP [13], ACP [14] and the Algebraic Theory of Processes [15]. Because of these different users we would expect a Event B refinement to be quantified over all machines, just as testing refinement [15] is quantified over all processes. Although in Event B [8] refinement is quantified over all operations, in [3] they define forward and backward Event B refinement quantified over all programs.

2.5 Decomposition and contexts

The Event B methodology is based on a top-down approach where machine M can be *decomposed* into a pair of component machines A and x, written $(\mathsf{A}, \mathsf{x}) \in decompose(\mathsf{M})$. In performing such a decomposition the new machines may contain both *internal* (normal) operations, denoted by op for example, and *external* operations, denoted by $\overline{\mathsf{op}}$. These new machines can then be *recomposed* so that $recompose(\mathsf{A}, \mathsf{x}) = \mathsf{M}$.

Because of this top-down construction we would start with process M and construct $(\mathsf{A}, \mathsf{x}) \in decompose(\mathsf{M})$ hence $[\![\mathsf{A}]\!]_{\mathsf{x}} = recompose(\mathsf{A}, \mathsf{x}) = \mathsf{M}$.

We do not need to give the details of Event B decomposition but we remind the reader that recomposition takes the union of the *internal* operations of A and x. The only synchronisation of operations is between two operations with the same name where one is internal and the other is external. The external operation is simply dropped in the construction of $recompose(\mathsf{A}, \mathsf{x})$.

The proof obligations of $decompose$ guarantee that the external operations of one process are more abstract than the internal operations with the same name in the other process, so for any op:

$$\overline{\mathsf{op}} \sqsubseteq \mathsf{op}. \tag{1}$$

Thus the contexts that A can be placed in must satisfy (1).

We postpone defining what can be observed of an Event B machine until after we review the observation of processes in the next section.

3 Machines are processes

This section introduces nothing new but will review some ideas from the event-based world of processes and the close relation between the labelled transition system semantics of processes and the state-based relational semantics of operations.

CSP [16, 13] was one of the first process calculus formalisms and uses *failure* semantics to model liveness properties and trace semantics to model safety properties. Failure semantics has been characterised by an elegant **must** testing semantics and trace semantics characterised by **may** testing semantics [17, 15]. When the interactions formalised by these testing semantics is the same as the interactions of real processes with their environment (contexts) then an engineer can have some confidence that it is appropriate to use these testing refinements.

Subsequently a wide range of testing semantics [18, 19] has been designed and the process-theoretic approach has been applied to processes that interact with each other in a wide variety of ways. Both B and Event B machines interact in a distinct, well-defined manner and this interaction can be formally modelled by a testing semantics. Thus following the process calculus approach will give some confidence that a definition of refinement can be constructed that is appropriate for Event B machines.

Let Act be a finite set of observable operations.

Definition 1 *LTS—labelled transition systems. Let $Alp \subseteq Act$ be the alphabet, N_{A} be a finite set of nodes and s_{A} the start node. LTS $\mathsf{A} \triangleq (N_{\mathsf{A}}, s_{\mathsf{A}}, e_{\mathsf{A}}, T_{\mathsf{A}}, Alp_{\mathsf{A}})$ where $s_{\mathsf{A}} \in N_{\mathsf{A}}$, $e_{\mathsf{A}} \in N_{\mathsf{A}}$, and $T_{\mathsf{A}} \subseteq \{(n, a, m) | n, m \in N_{\mathsf{A}} \wedge a \in Alp \wedge n \neq e_{\mathsf{A}}\}$.* •

Alp_{A} is the alphabet of A.

A *path* is a sequence of states and actions and the set of paths generated by the LTS A is: $Path_{\mathsf{A}} \triangleq \{s_{\mathsf{A}}, \rho_1^\alpha, n_2, \rho_2^\alpha, \ldots | (n_1, \rho_1^\alpha, n_2), (n_2, \rho_2^\alpha, n_3), \ldots \in T_{\mathsf{A}}\}$.

We write $|\rho|$ for the number of actions in (i.e. length of) a path and ρ^α for the sequence of actions $\rho_1^\alpha, \rho_2^\alpha \ldots$ in path $\rho = s_{\mathsf{A}}, \rho_1^\alpha, n_2, \rho_2^\alpha \ldots$. For finite paths $\rho = s_{\mathsf{A}}, \rho_1^\alpha, n_2, \rho_2^\alpha, \ldots n_i$ define $last(\rho) \triangleq n_i$. We will write ϵ for the empty sequence of actions, hence $s_{\mathsf{A}}^\alpha = \epsilon$. Where A is obvious from context we write $x \xrightarrow{\mathsf{a}} y$ for $(x, \mathsf{a}, y) \in T_{\mathsf{A}}$ $n \xrightarrow{\mathsf{a}}$ for $\exists m.(n, \mathsf{a}, m) \in T_{\mathsf{A}}$, $s_{\mathsf{A}} \xrightarrow{\rho^\alpha}$ when $\rho \in Path_{\mathsf{A}}$, $s_{\mathsf{A}} \xrightarrow{\rho^\alpha} n$ when $\rho \in Path_{\mathsf{A}} \wedge last(\rho) = n$, and finally $\pi(s) \triangleq \{\mathsf{a} | s \xrightarrow{\mathsf{a}}\}$.

We define the traces of A to be: $Tr(\mathsf{A}) \triangleq \{\rho^\alpha | s_{\mathsf{A}} \xrightarrow{\rho^\alpha}\}$.
The complete traces of A are:

$Tr^c(\mathsf{A}) \triangleq \{\rho^\alpha | (s_{\mathsf{A}} \xrightarrow{\rho^\alpha} n \wedge \pi(n) = \emptyset) \vee (s_{\mathsf{A}} \xrightarrow{\rho^\alpha} \wedge |\rho| = \infty)\} \cup \{\rho^\alpha \surd | (s_{\mathsf{A}} \xrightarrow{\rho^\alpha} e_{\mathsf{A}})\}$
Trace refinement is: $\mathsf{A} \sqsubseteq_{Tr} \mathsf{C} \triangleq Tr(\mathsf{C}) \subseteq Tr(\mathsf{A})$ and complete trace refinement is: $\mathsf{A} \sqsubseteq_{Tr^c} \mathsf{C} \triangleq Tr^c(\mathsf{C}) \subseteq Tr^c(\mathsf{A})$
We define refusals: $Ref(n, \mathsf{A}) \triangleq$

$$\{\{\mathsf{a} | n \not\xrightarrow{\mathsf{a}}\} \cup X | \mathsf{a} \in Alp \wedge n \in N_{\mathsf{A}} \wedge \text{ if } n = e_{\mathsf{A}} \text{ then } X = \emptyset \text{ else } X = \{\surd\}\}$$

We define failures: $Fail(\mathsf{A}) \triangleq \{(\rho, X) | s_{\mathsf{C}} \xrightarrow{\rho} n \wedge X \in Ref(n, \mathsf{A})\}$ and failure trace: $FT(\mathsf{A}) \triangleq \{Ref(s_{\mathsf{A}}, \mathsf{A}) \rho_1^\alpha Ref(n_2, \mathsf{A}) \rho_2^\alpha Ref(n_3, \mathsf{A}) \ldots | s_{\mathsf{C}} \xrightarrow{\rho}\}$.
Failure refinement [13, 14]: $\mathsf{A} \sqsubseteq_F \mathsf{C} \triangleq \forall \rho. Fail(\mathsf{C}) \subseteq Fail(\mathsf{A})$ and failure trace refinement [14]: $\mathsf{A} \sqsubseteq_{FT} \mathsf{C} \triangleq \forall \rho. FT(\mathsf{C}) \subseteq FT(\mathsf{A})$.

We only use failure trace refinement in Lemma 5 and all we need is the well-known result $\mathsf{A} \sqsubseteq_{FT} \mathsf{C} \Rightarrow \mathsf{A} \sqsubseteq_F \mathsf{C}$ [14].

Definition 2 *Parallel composition of LTS* A *and* B *is defined only if* $Alp_{\mathsf{A}} \cap Alp_{\mathsf{B}} = \emptyset$.

$N_{\mathsf{A}\|\mathsf{B}} \stackrel{\text{def}}{=} N_{\mathsf{A}} \times N_{\mathsf{B}}$, $e_{\mathsf{A}\|\mathsf{B}} = e_{\mathsf{A}} \times e_{\mathsf{B}}$, $s_{\mathsf{A}\|\mathsf{B}} = (s_{\mathsf{A}}, s_{\mathsf{B}})$ *and* $T_{\mathsf{A}\|\mathsf{B}}$ *is defined:*

$$\frac{n \xrightarrow{\mathsf{a}}_{\mathsf{A}} l, \overline{\mathsf{a}} \notin \alpha(\mathsf{B})}{(n, m) \xrightarrow{\mathsf{a}}_{\mathsf{A}\|\mathsf{B}} (l, m)} \qquad \frac{n \xrightarrow{\mathsf{a}}_{\mathsf{B}} l, \overline{\mathsf{a}} \notin \alpha(\mathsf{A})}{(m, n) \xrightarrow{\mathsf{a}}_{\mathsf{A}\|\mathsf{B}} (m, l)} \qquad \frac{n \xrightarrow{\mathsf{a}}_{\mathsf{A}} l, m \xrightarrow{\overline{\mathsf{a}}}_{\mathsf{B}} k}{(n, m) \xrightarrow{\mathsf{a}}_{\mathsf{A}\|\mathsf{B}} (l, k)}$$

let $\overline{X} \triangleq \{\overline{\mathsf{a}} | \mathsf{a} \in X\}$ *then* $Alp_{\mathsf{A}\|\mathsf{B}} \stackrel{\text{def}}{=} (Alp_{\mathsf{A}} - \overline{Alp_{\mathsf{B}}}) \cup (Alp_{\mathsf{B}} - \overline{Alp_{\mathsf{A}}})$. •

Let LTS be the set of all LTS. Our generic definition of refinement from [20] assumes that processes interact via the parallel composition given in Definition 2. But this generic definition of refinement is parameterised by:

1. the set of contexts that a process can interact with, $\Xi \subseteq \{(\_ \parallel x) | x \in \mathsf{LTS}\}$
2. $Obs : \mathsf{LTS} \rightarrow 2^{Ob}$ a function from LTS to a set of observations. Ob is the set of all possible observations.

Definition 3 $\mathsf{A} \sqsubseteq_{(\Xi, Obs)} \mathsf{C} \triangleq \forall [\_]_x \in \Xi. Obs([\mathsf{C}]_x) \subseteq Obs([\mathsf{A}]_x)$ •

It should be noted that processes can be given a relational semantics where the relation is between contexts and observations, $Rel(\mathsf{A}) \subseteq \Xi \times Ob$. Using this relational semantics the definition of refinement in Definition 3 is simply subset on the relational semantics:

Lemma 1 $\mathsf{A} \sqsubseteq_{(\Xi,Obs)} \mathsf{C} \iff Rel(\mathsf{C}) \subseteq Rel(\mathsf{A})$

It is important to note that this event-based definition of a relational semantics is not simply a relation from pre-state to post-state.

The safety semantics of a process A is given by its traces $Tr(\mathsf{A})$. The safety or partial correctness semantics of a process cannot be used to specify that something must happen. Thus if trace $\mathsf{a;b}$ were observable then any sub-trace a or ϵ must also be observable else if ϵ were not a valid observation we would know that a must be performed and if a were not a valid observation we would know that after performing an a then a b must be performed. Consequently using Definition 3 to model safety or partial correctness semantics we let $Obs : \mathsf{LTS} \rightarrow 2^{Tr}$.

Failure semantics is well-known to model liveness properties, that is from the failure semantics of a process, it is known when an operations **must** occur. This in state-based terminology is the total correctness semantics. It was shown in [20] that when $\Xi \subseteq \mathsf{LTS}$ and $Obs : \mathsf{LTS} \rightarrow 2^{Tr^c}$ then $\sqsubseteq_{(\Xi,Tr^c)}$ is failure refinement i.e. a total correctness semantics.

It is well-known that ADTs can be placed only in contexts that are programs, Prog, i.e. sequences of operations, whereas processes in general can be placed in branching contexts [4]. The refinement of ADTs (or objects) has been defined in the literature [5, 11] in the style of Definition 3 where $\Xi = \mathsf{Prog}$ and $Ob = Tr^c$.

3.1 Relating operation semantics

There is an obvious bijection between LTS semantics and the relational semantics of a machine. Note a machine consists of a set of named operations hence the relational semantics of a machine contains a set of named relations.

The operational semantics of Event B machine A is the set of named partial relations $Npr(\mathsf{A})$ or the operations in A as defined in Section 2.1. This can be used to define a LTS.

Definition 4 $lts(\mathsf{A}) \triangleq (N_{lts\mathsf{A}}, s_{lts\mathsf{A}}, T_{lts\mathsf{A}}, Alp)$ *where* $N_{lts\mathsf{A}}$ *is the set of evaluations of the variables of* A*,* $s_{lts\mathsf{A}} \triangleq init_{\mathsf{A}}$,

$$T_{lts\mathsf{A}} \triangleq \{(x, n, y) | (n, R_n) \in Npr(\mathsf{A}) \wedge (x, y) \in R_n\},$$

$$Alp \triangleq \textit{the set of operations of } \mathsf{A}.$$ •

Example Event B machine The partial relational semantics of Event B machine A is given by the solid arrows in W and U in Fig. 3. (The particular lifting and totalising that appears in Fig. 3 will be explained in the next section on Safe B semantics.)

The observation of lifted total relations. The *lifting* of the semantics adds a new element $\perp$ to the domain of the relations (see Fig. 3). In this paper $\perp$ will always be used to represent the *non-termination* of an operation. An operation is observed only when it terminates, hence when an operation goes to the post-state $\perp$ it does not terminate and consequently is not observed.

The solid arrows in Fig. 3 end at states other than $\perp$. They represent an operation that has terminated and consequently the operation can be **observed**, whereas the dotted arrows end at $\perp$ and represent an operation that has not terminated and consequently **cannot be observed**.

MACHINE A
SETS $Nodes = \{a, b\}$
VARIABLES st
INVARIANT $st \in Nodes$
INITIALISATION $st := a$
OPERATIONS
W $\triangleq$ SELECT $st = a$
THEN $st := a [] st := b$ END;
U $\triangleq$ SELECT $st = a$
THEN $st := a$ END;
END

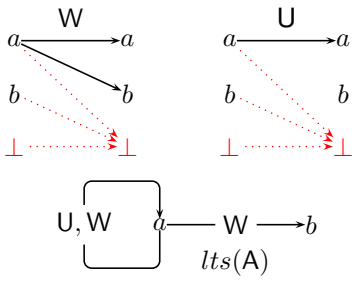

Figure 3: Machine A with its relational and LTS semantics

4 Safe B semantics - $\widehat{\mathsf{A}}$

The Safe B semantics of machines are the *lifted* total relations $\widehat{\mathsf{rel}} \subseteq S_\perp \times S_\perp$ where $S_\perp \triangleq \{v | I(v)\} \cup \{\perp\}$ and

$$\begin{aligned} \widehat{\mathsf{a}} &\triangleq \{v \mapsto v' | I(v) \wedge G(v) \wedge Pre(v) \wedge (R(v, v') \vee v' = \perp)\} \\ \widehat{\mathsf{m}} &\triangleq \{v \mapsto \perp | I(v) \wedge \neg G(v) \wedge Pre(v)\} \\ \widehat{\mathsf{u}} &\triangleq \{v \mapsto x | I(v) \wedge \neg Pre(v) \wedge x \in S_\perp\} \end{aligned}$$

The relation $\widehat{\mathsf{rel}}$ is defined as the union the *active*, *blocked* and *undefined* parts

$$\widehat{\mathsf{rel}} \triangleq \widehat{\mathsf{a}} \cup \widehat{\mathsf{m}} \cup \widehat{\mathsf{u}}.$$

The machine A in Fig. 3 is given a partial relation semantics by Event B, shown by the solid arrows in W and U. The Safe B semantics for A is $\widehat{\mathsf{A}}$, shown in Fig. 3 by both the solid and dotted arrows.

As we are adopting a partial correctness approach in this section the semantics only shows the behaviour of the operation if the operation terminates. As partial correctness semantics cannot guarantee termination any operation must always be able to not terminate. The totalising captures this intention by mapping all pre-states to $\perp$.

4.1 Relational semantics

From the state-based view it is natural to construct the relational semantics of a sequence of two operations by the relational composition of the relational semantics of each operation. But from the event-based view we must consider what can be observed of the sequence of operations.

If we look at the partial relation W; U in Fig. 4 we can see

$$\widehat{\mathsf{W}}; \widehat{\mathsf{U}} = \widehat{\mathsf{W}; \mathsf{U}}$$

We leave it to the interested reader to check that this is true in general.

It is easy to see that the lifted and totalised $\widehat{\mathsf{W}; \mathsf{U}}$ relation tells us the possible change in state or even that it fails to terminate. But let us consider A placed in the context of

program $\overline{\mathsf{W}};\overline{\mathsf{U}}$, then clearly operation W can be be performed and may end in state b (see Fig. 3) after which no further operations can be performed and thus it is possible for W alone to be observed. This observation cannot be inferred from the lifted and totalised relational semantics of $\widehat{\mathsf{W};\mathsf{U}}$ (see Fig. 4) as arrow $a \longrightarrow \bot$ implies either that nothing is observed or that $\mathsf{W};\mathsf{U}$ is not observed.

As pointed out in Section 3 a process can be given a relational semantics where the relation is between its contexts and observations. This is easy to compute from the relational semantics of the operations and the interpretation given in Section 3.1 (details can be found in [4]). A small part of the relation $Rel(\mathsf{A})$ is shown in Fig. 4 (right-hand side).

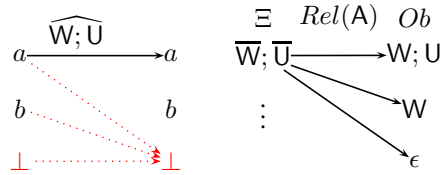

Figure 4: $Rel(\mathsf{A})$ of A Fig. 3

4.2 Operation Refinement

Here operations have a total relation semantics $\widehat{\mathsf{rel}_\mathsf{A}}$ and refinement is defined as the subset of the total relations:

$$\mathsf{A} \sqsubseteq_S \mathsf{C} \triangleq \widehat{\mathsf{rel}_\mathsf{C}} \subseteq \widehat{\mathsf{rel}_\mathsf{A}}.$$

It is easy to see that this refinement is the same as Event B refinement (see Section 2.3) on the partial relation semantics, i.e. $\mathsf{rel}_\mathsf{C} \subseteq \mathsf{rel}_\mathsf{A} \iff \widehat{\mathsf{rel}_\mathsf{C}} \subseteq \widehat{\mathsf{rel}_\mathsf{A}}$ and

$$\mathsf{A} \sqsubseteq_S \mathsf{C} \iff \mathsf{A} \sqsubseteq_E \mathsf{C}.$$

4.3 Machine Refinement

We are going to apply the event-based definition of process refinement (see Section 3) that is based on placing a process in a context and observing the execution of both the process and context.

In Section 2.3 we defined the contexts that Event B machines can be placed in. In Section 3.1 we discussed the observation of operations with lifted total relations. When we apply this to the lifted total relations that model partial correctness semantics we find that traces are observable. It is easy to see that when traces are observable then refinement given in Definition 3 is the standard trace refinement. This is exactly what we expect as it is already known in the process literature that for safety-only properties (partial correctness) trace semantics is all that is needed.

Definition 5 *Safe B refinement* $\sqsubseteq_{SB}$*:*

$$\mathsf{A} \sqsubseteq_{SB} \mathsf{C} \triangleq \forall x \in \mathbb{B}.Tr(lts(\llbracket \mathsf{C} \rrbracket_\mathsf{x})) \subseteq Tr(lts(\llbracket \mathsf{A} \rrbracket_\mathsf{x})) \quad \bullet$$

It is easy to establish that, for machines A in Fig. 3 and C in Fig. 5, $\mathsf{A} \sqsubseteq_{SB} \mathsf{C}$.

We can infer B machine refinement from the well-known corresponding process-style refinements given in Section 3.

Lemma 2 $lts(\mathsf{A}) \sqsubseteq_{Tr} lts(\mathsf{C}) \Rightarrow \mathsf{A} \sqsubseteq_{SB} \mathsf{C}$

MACHINE C
SETS $Nodes = \{a, b\}$
VARIABLES st
INVARIANT $st \in Nodes$
INITIALISATION $st := a$
OPERATIONS
W $\triangleq$ SELECT $st = a$
THEN $st := b$ END;
U $\triangleq$ SELECT $st = a$
THEN $st := a$ END;
END

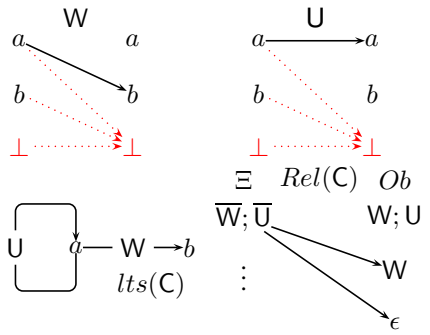

Figure 5: Machine C and safety only semantics

Proof (sketch). It is well-known [15] that Prog are all the tests needed to establish the safe (may) refinement of a process which is trace refinement and hence:

$lts(\mathsf{A}) \sqsubseteq_{(\mathsf{Prog},Tr)} lts(\mathsf{C}) \Leftrightarrow lts(\mathsf{A}) \sqsubseteq_{(LTS,Tr)} lts(\mathsf{C}) \Leftrightarrow lts(\mathsf{A}) \sqsubseteq_{Tr} lts(\mathsf{C})$.

From definitions $lts([\![\mathsf{A}]\!]_{\mathsf{x}}) =_{Tr^c} [lts(\mathsf{A})]_{lts(\mathsf{X})}$. Finally as $\mathsf{Prog} \subset lts(\mathbb{B}) \subset LTS$, we have $lts(\mathsf{A}) \sqsubseteq_{(LTS,Tr)} lts(\mathsf{C}) \Rightarrow \mathsf{A} \sqsubseteq_{SB} \mathsf{C}$. •

As we would expect, and it is easy to check, for machines A in Fig. 3 and C in Fig. 5, $lts(\mathsf{A}) \sqsubseteq_{Tr} lts(\mathsf{C})$.

4.4 Proof obligations and refinement

Quantifying over all Event B machines is not computationally feasible and the standard solution is to construct a set of sufficient conditions quantified over all operations.

The Event B proof obligations are crafted so as to establish the well-known forward simulation as illustrated in Fig. 6. Forward simulation is sufficient to guarantee refinement. The lack of a "final" operation in Fig. 6 may, from the state-based perspective, seem worrying. But let us recall that many useful event-based programs do not terminate and event-based definitions of refinement and simulation [21] do not need any final operation.

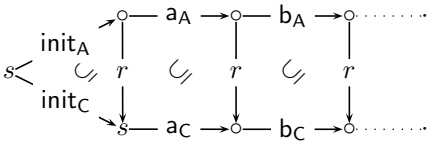

Figure 6: Forward simulation

The proof obligations given here are no more that a slight rephrasing of those found in [8, 3].

$\mathsf{init_C}(w) \Rightarrow \exists v.\mathsf{init_A}(v) \wedge I_C(v, w)$	INIT_REF

The FIS_REF proof obligation ensures refinement builds a total relation on the *active* states of the concrete operation.

The GRD_REF proof obligation guarantees that refinement preserves *blocked* states of the relation, i.e. those v for which $(Pre_A(v) \wedge \neg G_A(v))$ holds.

The INV_REF proof obligation guarantees that the refinement of *active* states satisfies the subset in Fig. 6 . The only constraint on the the *undefined* states of the relation, i.e. those v for which $(\neg Pre_A(v))$ holds, is that it must be in one of $\widehat{u}$, $\widehat{m}$ or $\widehat{a}$. This is automatically guaranteed from the interpretation of the three predicates.

$I_A(v) \wedge I_C(v,w) \wedge Pre_A(v) \wedge G_A(v) \Rightarrow \quad \exists w'.R_C(w,w')$	FIS_REF
$I_A(v) \wedge I_C(v,w) \wedge Pre_A(v) \wedge \neg G_A(v) \Rightarrow \quad Pre_C(w) \wedge \neg G_C(w)$	GRD_REF
$I_A(v) \wedge I_C(v,w) \wedge Pre_C(w) \wedge G_C(w) \wedge R_C(w,w') \Rightarrow$ $G_A(v) \wedge Pre_A(v) \wedge \exists v'.(R_A(v,v') \wedge I_C(v',w'))$	INV_REF

Let Safe be the set of proof obligations {INIT_REF, FIS_REF, GRD_REF, INV_REF} It is well-known and easy to see that these proof obligations establish the individual subset relations in Fig. 6. Hence the proof obligations establish a refinement.

The mapping lts between the state- and event-based semantics can be used to show what we might expect: that the Safe proof obligations of Event B refinement are sufficient to establish trace refinement.

Lemma 3 *The* Safe *proof obligations imply* $lts(\mathsf{A}) \sqsubseteq_{Tr} lts(\mathsf{C})$ *and* $\mathsf{A} \sqsubseteq_{SB} \mathsf{C}$

Proof (sketch). The Safe proof obligations imply the subset relations in Fig. 6. It is clear from inspection of Fig. 6 that $lts(\mathsf{A}) \sqsubseteq_{(\mathsf{Prog},Tr)} lts(\mathsf{C})$. The result follows from Lemma 2. •

This section has no new results. It is interesting, though, as it makes use of the close connection between the state-based relational semantics and the event-based LTS semantics to apply a testing-style definition of refinement to Event B machines.

5 Live B semantics - $\widetilde{\mathsf{A}}$

In order to model what can be observed of an execution of an Event B machine we need to define the effect of executing a sequence of operations. But as already discussed in Section 1.1, with total correctness semantics modelling sequential composition of operations as the relational composition of partial relations is problematic.

We will write $\mathsf{W_C}$ for the W operation of machine C (Fig. 5) and $\mathsf{W_A}$ for the W operation of machine A (Fig. 3).

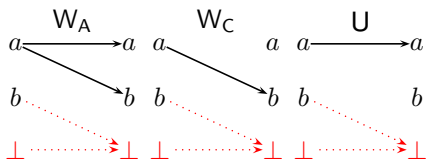

Figure 7: Operations from A and C

Using the partial relation semantics, the solid arrows in Fig. 7, we can see that $\mathsf{W_A; U}$ must terminate! As we want, and would expect, $\mathsf{W_A} \sqsubseteq \mathsf{W_C}$, but using the natural notion of refinement given in Section 2.4, $\mathsf{A} \not\sqsubseteq \mathsf{C}$. This can easily be seen by considering what can be observed, based on partial relation semantics:

$$Obs([\![\mathsf{A}]\!]_{\overline{\mathsf{W}};\overline{\mathsf{U}}}) = \{\mathsf{W;U}\} \text{ but } Obs([\![\mathsf{C}]\!]_{\overline{\mathsf{W}};\overline{\mathsf{U}}}) = \{\mathsf{W}\}.$$

The clairvoyance that partial relations would introduce means that $\mathsf{W} \notin Obs([\![\mathsf{A}]\!]_{\overline{\mathsf{W}};\overline{\mathsf{U}}})$. This we regard as counter-intuitive. It means that refinement, defined in a natural way

(Section 2.4), would give unexpected and undesirable results. Because of this a lifted, totalised semantics is, we believe, to be preferred.

5.1 Lifting and totalising

The Live B semantics of machines are the *lifted*, total relation $\widetilde{\mathsf{rel}} \subseteq S_\perp \times S_\perp$ where $S_\perp \triangleq \{v | I(v)\} \cup \{\perp\}$ and

$$\begin{array}{rcl} \mathsf{a} & \triangleq & \{v \mapsto v' | I(v) \wedge G(v) \wedge Pre(v) \wedge R(v,v')\} \\ \widehat{\mathsf{m}} & \triangleq & \{v \mapsto \perp | I(v) \wedge \neg G(v) \wedge Pre(v)\} \\ \widehat{\mathsf{u}} & \triangleq & \{v \mapsto x | I(v) \wedge \neg Pre(v) \wedge x \in S_\perp\} \\ \widetilde{\mathsf{rel}} & \triangleq & \mathsf{a} \cup \widehat{\mathsf{m}} \cup \widehat{\mathsf{u}} \end{array}$$

The difference between $\widetilde{\mathsf{rel}}$ and $\widehat{\mathsf{rel}}$ is that in $\widetilde{\mathsf{rel}}$ the active part of the relation, a, is guaranteed to terminate (see Fig. 7) whereas in the partial correctness semantics $\widehat{\mathsf{rel}}$ the active part is $\widehat{\mathsf{a}}$ where no such guarantee exists.

Definition 6 *Live B refinement* $\sqsubseteq_{LB}$ *:*

$$\mathsf{A} \sqsubseteq_{LB} \mathsf{C} \triangleq \forall x \in \mathbb{B}.Tr^c(lts([\![\mathsf{C}]\!]_\mathsf{x})) \subseteq Tr^c(lts([\![\mathsf{A}]\!]_\mathsf{x}))$$ •

5.2 Proof obligations and refinement

The ACT_REF proof obligation guarantees that refinement preserves *active* states of the relation, those states x where $(Pre(x) \wedge G(x))$.

$I_A(v) \wedge I_C(v,w) \wedge Pre_A(v) \wedge G_A(v) \Rightarrow \quad Pre_C(w) \wedge G_C(w)$	ACT_REF

Let Live be the proof obligations Safe $\cup \{$ACT_REF$\}$. The Live proof obligations of Event B refinement are sufficient to establish failure refinement.

It should be noted that this proof obligation is little more than a rephrasing of the proof obligation $Fwd.CT$ in Dunne and Conroy's Proposition 2 [3, p58].

Lemma 4 $lts(\mathsf{A}) \sqsubseteq_F lts(\mathsf{C}) \Rightarrow \mathsf{A} \sqsubseteq_{LB} \mathsf{C}$

Proof (sketch). From [20] $lts(\mathsf{A}) \sqsubseteq_F lts(\mathsf{C}) \Leftrightarrow lts(\mathsf{A}) \sqsubseteq_{(LTS,Tr^c)} lts(\mathsf{C})$. From definitions $lts([\![\mathsf{A}]\!]_\mathsf{x}) =_{Tr^c} [lts(\mathsf{A})]_{lts(\mathsf{X})}$. Finally as $lts(\mathbb{B}) \subset LTS$ we have

$$lts(\mathsf{A}) \sqsubseteq_{(LTS,Tr^c)} lts(\mathsf{C}) \Rightarrow \mathsf{A} \sqsubseteq_{LB} \mathsf{C}.$$ •

Lemma 5 *The* Live *proof obligations imply* $lts(\mathsf{A}) \sqsubseteq_F lts(\mathsf{C})$ *and* $\mathsf{A} \sqsubseteq_{LB} \mathsf{C}$

Proof (sketch). From Lemma 3 we know that the Live proof obligations imply $lts(\mathsf{A}) \sqsubseteq_{Tr} lts(\mathsf{C})$. But ACT_REF implies that the active states are preserved by refinement thus $\pi(n_\mathsf{A}) \subseteq \pi(n_\mathsf{C})$ and hence $Ref(n_\mathsf{C}) \subseteq Ref(n_\mathsf{A})$. Thus $lts(\mathsf{A}) \sqsubseteq_{FT} lts(\mathsf{C})$. From the literature it is well-known that this implies $lts(\mathsf{A}) \sqsubseteq_F lts(\mathsf{C})$.

From Lemma 4 we have $\mathsf{A} \sqsubseteq_{LB} \mathsf{C}$. •

5.2.1 Comparison of Live B with Event B

It is clear that Event B has guarded (and hence blocked) states, but not undefined states. In order to compare Live B with Event B we first remove the undefined states from Live B. This we can do by assuming:

$$Pre_A(v) = Pre_C(w) = true$$

and simplifying our proof obligations.

Our INV_REF and GRD_REF proof obligations will be the same as Event B proof obligations by the same name, see [8]. But, our ACT_REF proof obligation, though an obvious proof obligation, is not found in [8] because they construct partial correctness semantics. ACT_REF prevents the active part of machine operations from being refined into "do nothing".

6 Conclusion

This paper introduces little that, taken in individual parts, is new. Most of the work consists in pulling together material, not all of which is widely known, from both the state-based and event-based literature. What we found of interest is that unlifted relational semantics are ambiguous, being open to both partial correctness and total correctness interpretations (Section 1.1). If we add to this the fact that using partial relations to model total correctness semantics introduces clairvoyant behaviour (Section 1.1), we see good reason to use a semantics based on lifted, total relations.

We have introduced two lifted, total relation semantics for Event B operations. The Safe B semantics $\widehat{A}$ is a partial correctness semantics and the Live B semantics $\widetilde{A}$ is a total correctness semantics. Applying a natural definition of refinement to the partial correctness semantics $\widehat{A}$ results in the same pre-order as Event B's usual partial relations. Applying this same natural definition of refinement, but to the total correctness semantics $\widetilde{A}$, gives a new definition of refinement. Proof obligations, essentially taken from [3], are sufficient to imply this new definition of refinement while avoiding clairvoyant behaviour.

Because partial correctness and total correctness semantics are modelled by the lifting and totalising of individual operations there is no reason why we should not have operations with partial correctness semantics and operations with total correctness semantics in the same machine. This could be of use if some operations represented calls to a remote process that might not terminate, and hence need safety-only semantics, and other operations are under local control and we can guarantee that they will terminate, hence they can be given live semantics.

6.1 Comparison with literature

Dunne and Conroy's Proposition 3 [3, p59] claims to give local conditions that are necessary and sufficient to establish singleton failure refinement for Event B machines. In [3, Section 4.1] a trace of a B machine is defined in terms of a sequence of the operations. As Event B operations have a partial relation semantics, this definition introduces clairvoyant behaviour as seen in Fig. 1. As a consequence of using partial

relations our example machines A (Fig. 3) and C (Fig. 5) are refinements but are not singleton failure refinements.

References

[1] Spivey, J.M.: The Z notation: A reference manual. 2nd. edn. Prentice Hall (1992)

[2] Nelson, G.: A generalization of Dijkstra's calculus. ACM Transactions on Programming Languages and Systems **11** (1989) 517–561

[3] Dunne, S., Conroy, S.: Process refinement in B. In Treharne, H., King, S., Henson, M.C., Schneider, S., eds.: ZB 2005: Formal Specification and Development in Z and B, 4th International Conference of B and Z Users. Volume 3455 of Lecture Notes in Computer Science., Springer (2005) 45–64

[4] Bolton, C., Davies, J.: A singleton failures semantics for Communicating Sequential Processes. Research Report PRG-RR-01-11, Oxford University Computing Laboratory (2001)

[5] Woodcock, J., Davies, J.: Using Z: Specification, Refinement and Proof. Prentice Hall (1996)

[6] Deutsch, M., Henson, M.C., Reeves, S.: An analysis of total correctness refinement models for partial relation semantics: part 1. The Logic Journal of the IGPL **11** (2003) 285–316

[7] de Roever, W.P., Engelhardt, K.: Data Refinement: Model oriented proof methods and their comparison. Cambridge Tracts in theoretical computer science 47 (1998)

[8] Metayer, C., Abrial, J.R., Voisin, L.: Event-B language. RODIN Project Deliverable D7 (2005)

[9] Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (1996)

[10] Abrial, J.R.: Extending B without changing it (for developing distributed systems). In Habrias, H., ed.: Proceedings of 1st Conference on the B method. Putting into Practice methods and tools for information system design, 3 rue du Maréchal Joffre, BP 34103, 44041 Nantes Cedex 1, B1996, IRIN Institut de recherche en informatique de Nantes (1996) 169–191

[11] Derrick, J., Boiten, E.: Refinement in Z and Object-Z: Foundations and Advanced Applications. Formal Approaches to Computing and Information Technology. Springer (2001)

[12] Derrick, J., Boiten, E.: Relational concurrent refinement. Formal Aspects of Computing **15** (2003) 182–214

[13] Roscoe, A.: The Theory and Practice of Concurrency. Prentice Hall International Series in Computer Science (1997)

[14] Baeten, J.C.M., Weijland, W.P.: Process Algebra. Cambridge Tracts in Theoretical Computer Science 18 (1990)

[15] Hennessy, M.: Algebraic Theory of Processes. The MIT Press (1988)

[16] Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall International Series in Computer Science (1985)

[17] de Nicola, R., Hennessy, M.: Testing equivalences for processes. Theoretical Computer Science **34** (84)

[18] van Glabbeek, R.J.: Linear Time-Branching Time Spectrum I. In: CONCUR '90 Theories of Concurrency: Unification and Extension. LNCS 458, Springer-Verlag (1990) 278–297

[19] van Glabbeek, R.J.: The Linear Time - Branching Time Spectrum II. In: International Conference on Concurrency Theory. (1993) 66–81

[20] Reeves, S., Streader, D.: Comparison of Data and Process Refinement. In Dong, J.S., Woodcock, J.C.P., eds.: ICFEM 2003. LNCS 2885. Springer-Verlag (2003) 266–285

[21] Lynch, N., Vaandrager, F.: Forward and backward simulations, part i: Untimed systems. Information and Computation 121(2) (1995) 214–233