Working Paper Series
ISSN 1170-487X

Liberalising Event B
Without changing it

Steve Reeves and David Streader

Working Paper: 07/2006
July 10, 2006

©Steve Reeves and David Streader
Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Liberalising Event B (Without changing it)

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

July 10, 2006

Abstract

We transfer a process algebraic notion of refinement to the B method by using the well-known bridge between the relational semantics underlying the B machines and the labelled transition system semantics of processes. Thus we define *delta refinement* on Event B systems. We then apply this new refinement to a problem from the literature that previously could only be solved by retrenchment.

Keywords: process refinement, automatic verification, frame refinement, Event B

1 Introduction

In this paper we will clearly state a well-known formal relation between the set theoretic semantics underlying B machines and the labelled transition system (LTS) semantics of process algebra. However since such operational semantics only define part of the semantics, we also need to define a refinement relation on the operational semantics.

The semantics (meaning) of a LTS is commonly given by defining a refinement relation on the LTS, indeed see Glabeck [1] for an interesting survey of refinements and testing semantics. The informal intuition is that the meaning of a specification is given by the set of implementations that *satisfy* it and each distinct refinement relation defines a distinct set of implementations that the specification can be refined into. Hence each refinement may define a distinct meaning for a single LTS

First we will review simple B machines (Section 2) and their operational semantics. We state the proof obligations from Event B and give a definition of refinement of Event B machines. It is easy to show that the proof obligations imply the refinement. Next we define the LTS semantics of processes (Section 3) and give the definition of trace refinement. Using this we define the relation between the two operational semantics and show (Section 3.1) that proof obligations imply trace refinement.

Then to illustrate the operational semantics of B machines and motivate our discussion we give an example problem (Section 4) taken from a paper on retrenchment [2].

In this paper we are interested in *frame refinements* that allow events (actions) that do not appear in the abstract specification to appear in the more concrete specification.

We introduce two known definitions of process refinement and apply them to Event B machines (Section 5) to give a new and more liberal notion of refinement.

Finally we use this to refine our example specification so as to satisfy the specification (Section 6).

2 B Machine operational semantics

Both B and Event B define machines with a set of private variables, an invariant condition and a set of public operations. We will refer to the set of operations as the **frame** of the machine. We do this because this set of of operations forms a frame of reference and the machine will tell us nothing about operations that appear outside of the frame of reference.

To be valid the machine must satisfy some "proof obligations". What make B and Event B machines so useful is the first order proof obligations that can be automatically generated and which are *sufficient* to infer refinement.

In B [3] operations are given a partial relational semantics. The domain is partitioned into three disjoint sets. The states that do not satisfy a precondition Pre are *undefined*. The states that satisfy Pre can be either pre-states not related to any post-state (referred to as *magic*) or pre-states that are related to a post-state (referred to as *active*).

Event B was introduced in *Extending B without changing it* [4] to model operations that could be guarded in the process algebraic sense. In Event B operations (now called events) do not use the precondition Pre and cannot be undefined on part of their domain. Here we revert back to B like syntax for events. Thus we permit both guards $G(v)$ and preconditions $Pre(v)$ to be applied to generalised substitutions $R(v, v')$:

$$\boxed{E \triangleq \textbf{PRE}\ Pre(v)\ \textbf{SELECT}\ G(v)\ \textbf{THEN}\ R(v, v')\ \textbf{END}}$$

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

$$\boxed{stop \triangleq \textbf{PRE}\ FALSE\ \textbf{THEN}\ skip\ \textbf{END}}$$

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

2.1 Refinement in B and Event B

B generates proof obligations that are *sufficient* but not *necessary* to establish that a concrete machine is a refinement of an abstract machine. In Event B refinement is defined as a property quantified over all operations whereas in B refinement is defined as a property quantified over all programs.

The effect of a program consists of the initialisation of a machine init followed by a sequence of the machine's operations $\mathsf{p_x} \triangleq \mathsf{init_x}; \mathsf{o_x^*}$, where the machine can be abstract $(\mathsf{x} = \mathsf{a})$ or concrete $(\mathsf{x} = \mathsf{c})$. Let P be the set of all programs and r the relation between the abstract and concrete state that is defined in the $INVARIANT$ clause of the B $REFINEMENT$.

The definition of refinement we use is:

$$\mathsf{A} \sqsubseteq_B \mathsf{C} \triangleq \forall \mathsf{p} \in P.\mathsf{p}_\mathsf{C} \subseteq \mathsf{p}_\mathsf{A}; r$$

It is easy to see, and well-known [5], that Event B proof obligations:

$I_A(v) \wedge J(v,w) \wedge Pre_A(v) \wedge G_A(v) \Rightarrow \exists w'.S(w,w')$	FIS_REF
$\mathsf{init}_\mathsf{C}(w) \Rightarrow \exists v.\mathsf{init}_\mathsf{A}(v) \wedge I_C(v,w)$	INIT_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

directly imply the subset relations in Fig. 1.

$$\begin{tikzcd}
 & & \circ \arrow[rr, "\mathsf{a}_\mathsf{A}"] \arrow[dd, "r"] & & \circ \arrow[rr, "\mathsf{b}_\mathsf{A}"] \arrow[dd, "r"] & & \circ \arrow[r, dotted] \arrow[dd, "r"] & {} \\
s \arrow[urr, "\mathsf{init}_\mathsf{A}"] \arrow[drr, "\mathsf{init}_\mathsf{C}"'] & \subseteq & & \subseteq & & \subseteq & & \\
 & & \circ \arrow[rr, "\mathsf{a}_\mathsf{C}"] & & \circ \arrow[rr, "\mathsf{b}_\mathsf{C}"] & & \circ \arrow[r, dotted] & {}
\end{tikzcd}$$

Figure 1: Forwards refinement

Lemma 1 *The subset relations in Fig. 1 imply* $\mathsf{A} \sqsubseteq_B \mathsf{C}$

This is clear from inspection of Fig. 1. •

3 Operational semantics of processes

Let Act be a finite set of observable operations and τ and δ be two special unobservable 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* $\mathsf{A}_{Alp} \triangleq (N_\mathsf{A}, s_\mathsf{A}, e_\mathsf{A}, T_\mathsf{A}, Alp)$ *where* $s_\mathsf{A} \in N_\mathsf{A}$, $e_\mathsf{A} \in N_\mathsf{A}$, *and* $T_\mathsf{A} \subseteq \{(n,a,m) | n,m \in N_\mathsf{A} \wedge a \in Alp^\tau \wedge n \neq e_\mathsf{A}\}$. •

To reduce notational clutter A_{Alp} will be frequently written as A where its alphabet Alp is clear from context and will write $\alpha(\mathsf{A})$ for Alp the alphabet of A.

A *path* is a sequence of states and actions and the set of paths generated by the LTS A is: $Path_\mathsf{A} \triangleq \{s_\mathsf{A}, \rho^\alpha_1, n_2, \rho^\alpha_2, \ldots | (n_1, \rho^\alpha_1, n_2), (n_2, \rho^\alpha_2, n_3), \ldots \in T_\mathsf{A}\}$.

We write $|\rho|$ for the number of actions in (i.e. length of) a path and ρ^α for the sequence of actions $\rho^\alpha_1, \rho^\alpha_2 \ldots$ in path $\rho = s_\mathsf{A}, \rho^\alpha_1, n_2, \rho^\alpha_2 \ldots$. For finite paths $\rho = s_\mathsf{A}, \rho^\alpha_1, n_2, \rho^\alpha_2, \ldots n_i$ define $last(\rho) \triangleq n_i$. We will write ϵ for the empty sequence of actions, hence $s^\alpha_\mathsf{A} = \epsilon$. 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 $\rho \in Path_\mathsf{A}$ and finally $s_\mathsf{A} \xrightarrow{\rho^\alpha} n$ when $\rho \in Path_\mathsf{A} \wedge last(\rho) = n$.

Definition 1 takes no account of τ actions being unobservable, and we call $\rightarrow$ a **strong semantics** and define the traces of A to be: $Tr(\mathsf{A}) \triangleq \{\rho^\alpha | s_\mathsf{A} \xrightarrow{\rho^\alpha}\}$.

and trace refinement to be: $\mathsf{A} \sqsubseteq_{Tr} \mathsf{C} \triangleq Tr(\mathsf{C}) \subseteq Tr(\mathsf{A})$.

3.1 Relating operational 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:

Let the transitions of LTS A be the set of triples T_{A}; this defines the following set of named partial relations:

$$\{(n, R_n) | (x, n, y) \in T_{\mathsf{A}}\} \text{ where } R_n \triangleq \{(x, y) | (x, n, y) \in T_{\mathsf{A}}\}.$$

Similarly the set of named partial relations Npr defines a set of transitions:

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

The alphabet of an Event B machine is the set of names of its operations. As an operation $\mathsf{o} \triangleq stop$ will generate no transitions the LTS needs an explicit definition of its alphabet.

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

Lemma 2 $\mathsf{A} \sqsubseteq_B \mathsf{C}$ *implies* $\mathsf{A} \sqsubseteq_{Tr} \mathsf{C}$

Proof: We will prove $\mathsf{A} \not\sqsubseteq_{Tr} \mathsf{C} \Rightarrow \mathsf{A} \not\sqsubseteq_B \mathsf{C}$ and from this, using propositional logic, we can infer $\mathsf{A} \sqsubseteq_B \mathsf{C} \Rightarrow \mathsf{A} \sqsubseteq_{Tr} \mathsf{C}$.

Assume $\mathsf{A} \not\sqsubseteq_{Tr} \mathsf{C}$ and hence $\exists \mathsf{t} \in Tr(\mathsf{C}) \wedge \mathsf{t} \notin Tr(\mathsf{A})$

$$\begin{tikzcd}
 & \circ \arrow[r, "\mathsf{a_A}"] \arrow[dd, "r"] \arrow[ddr, phantom, "\circlearrowleft"] & \circ \arrow[r, "\mathsf{b_A}"] \arrow[dd, "r"] \arrow[ddr, phantom, "\circlearrowleft"] & \circ \arrow[dd, "r"] \arrow[ddr, phantom, "\not\circlearrowleft"] & \\
s \arrow[ur, "\mathsf{init_A}"] \arrow[dr, "\mathsf{init_C}"'] \arrow[r, phantom, "\circlearrowleft"] & & & & \\
 & \circ \arrow[r, "\mathsf{a_C}"'] & \circ \arrow[r, "\mathsf{b_C}"'] & \circ \arrow[r, "\mathsf{c_C}"'] & \cdots
\end{tikzcd}$$

Figure 2: Contradiction

With out loss of generality let $\mathsf{t} \triangleq \mathsf{a}; \mathsf{b}; \mathsf{c} \ldots$ and let the greatest prefix in $Tr(\mathsf{A})$ be $\mathsf{a}; \mathsf{b}$. Hence clearly from Fig. 2 $\mathsf{A} \not\sqsubseteq_B \mathsf{C}$. •

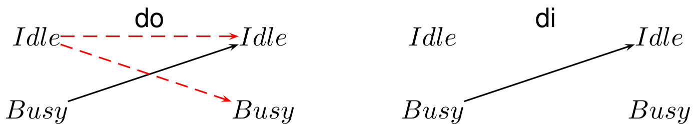
4 Example - Mobile Radio

For our example we use the specification of a Mobile Radio from [2] where the difficulties refining the low-level mobile radio $LLMR$ are used to justify the use of retrenchment. We make one small change to the specification: their call outgoing operation $\mathsf{co(x)}$ is amended to $\mathsf{x} \leftarrow \mathsf{co}$ with an outgoing parameter.

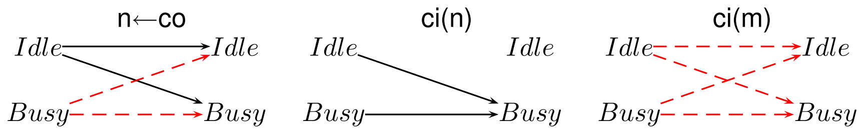
MACHINE	$HLMR$
SETS	$CALLS = \{Idle, Busy\}$
VARIABLES	$callState, currChan$
INVARIANT	$callState \in CALLS \wedge currChan \in CHANNELS$
INITIALISATION	$callState := Idle \parallel currChan :\in CHANNELS$
OPERATIONS	

```
    do  ≜ PRE callState = Busy THEN callState := Idle END;
    di  ≜ SELECT callState = Busy THEN callState := Idle END;
  x←co  ≜ PRE $callState = Idle \wedge x \in CHANNELS$ THEN
                CHOICE $callState := Busy \parallel x := currChan$ OR $skip$
                END
          END;
   ci(x) ≜ PRE $x \in CHANNELS$ THEN
                SELECT $callState = Idle$ THEN
                              callState := Busy || currChan:= $x$ ELSE $skip$
                END
          END;
END
```

In the relational semantics we use solid arrows for the *active* part of the operation and dashed arrows for the *undefined* part of the operation.



The radio is either in an $Idle$ or a $Busy$ state and the channel number it is always set to an element of $CHANNEL$. The radio is initially in $Idle$. The call outgoing action $\mathsf{n} \leftarrow \mathsf{co}$ and call incoming action $\mathsf{ci}(\mathsf{x})$ both take a channel number as parameter and this parameter is either $\mathsf{n} \in CHANNEL$ or $\mathsf{m} \notin CHANNEL$.



In the right-hand LTS of Fig. 3 we have included only the **active** and **guarded** operations of the relational semantics and have omitted the **undefined** operations. Note the dotted arrow to $\perp$ represents that di is blocked from state $Idle$. The undefined operations appear in the left-hand LTS. We could have added the undefined operations to the right-hand LTS but this would have made it very hard to read.

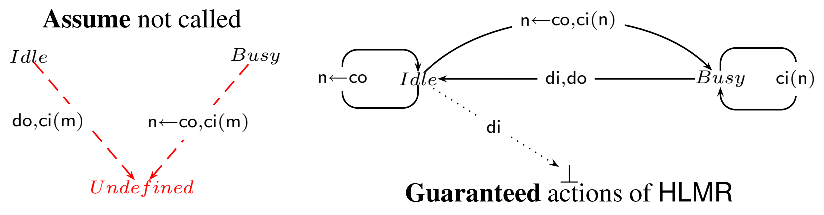


Figure 3: Assume Guarantee Specification of High Level Mobile Radio

The specification makes no guarantee as to its behaviour if one of the undefined operations is called. Rephrasing the specification assumes that undefined operations of the left LTS are not called and only guarantees its behaviour (the right-hand LTS) when the assumption is satisfied.

The high-level Mobile Radio has been partially specified. How it behaves with operation $\mathsf{ci(m)}$ is completely undefined, as is operation do from state $Idle$ and operation $\mathsf{n} \leftarrow \mathsf{co}$ from state $Busy$.

5 Frame Refinement

The state of our machines is private. The only part of a machine that can be seen by, or interact with, a context is the set of operations (events). Consequently the frame of a machine is the set of its operations, and frame refinement introduces new operations.

The treatment of frame refinement is different in the state-based and event-based approaches. The first point to note is that in the event-based world there are two forms of *unobservable* event: the τ and δ events, see [6] for details. Consequently there are two ways to remove events from a concrete process: one, *hiding* or *abstraction*; and the other, *restriction*. Each of these methods of removing events can be reversed to give two definitions of *frame refinement*, whereas the state-based literature has focused on just one of these.

Definition 2 *Operation on LTS* A
$\mathsf{A}\delta_D \triangleq (N_\mathsf{A}, s_\mathsf{A}, e_\mathsf{A}, T_{\mathsf{A}\delta_D})$ *where* $D \subseteq Act$ *and* $T_{\mathsf{A}\delta_D}$ *defined by:*

$$\frac{n \xrightarrow{\mathsf{a}}_\mathsf{A} l,\ \mathsf{a} \notin D \cup \{\delta\}}{n \xrightarrow{\mathsf{a}}_{\mathsf{A}\delta_D} l} \qquad \alpha(\mathsf{A}\delta_D) \triangleq \alpha(\mathsf{A}) - D$$

$\mathsf{A}\tau_S \triangleq (N_\mathsf{A}, s_\mathsf{A}, e_\mathsf{A}, T_{\mathsf{A}\tau_T})$ *where* $T \subseteq Act$ *and* $T_{\mathsf{A}\tau_S}$ *defined by:*

$$\frac{n \xrightarrow{\mathsf{a}}_\mathsf{A} l,\ \mathsf{a} \notin T}{n \xrightarrow{\mathsf{a}}_{\mathsf{A}\tau_T} l} \frac{n \xrightarrow{\mathsf{a}}_\mathsf{A} l,\ \mathsf{a} \in T}{n \xrightarrow{\tau}_{\mathsf{A}\tau_T} l} \qquad \alpha(\mathsf{A}\tau_T) \triangleq \alpha(\mathsf{A}) - T \qquad \bullet$$

Both unobservable operations τ and δ can be removed from the LTS without changing its observational semantics. The removal of δ actions always simplifies the LTS and is so simple it has been built into Definition 2. All δ operations can simply be removed.

The removal of the τ operations is more problematic and in Definition 2 operations are simply renamed as τ to be removed later by abstraction.

5.1 Abstraction

To model τ events as unobservable we show how to abstract them. This results in a LTS that consists only of observable events.

Definition 3 *Observational semantics* $\Longrightarrow$*:*

$$s \overset{\tau}{\Longrightarrow} t \triangleq s \xrightarrow{\tau} s_1, s_1 \xrightarrow{\tau} s_2, \ldots s_{n-1} \xrightarrow{\tau} t$$
$$n \overset{\mathsf{a}}{\Longrightarrow} m \triangleq n \overset{\tau}{\Longrightarrow} n', n' \xrightarrow{\mathsf{a}} m', m' \overset{\tau}{\Longrightarrow} m \wedge \mathsf{a} \in Act$$
$$Abs(\mathsf{A}) \triangleq (N_\mathsf{A}, s_\mathsf{A}, e_\mathsf{A}, \{n \xrightarrow{x} m \mid n \overset{x}{\Longrightarrow} m\}). \qquad \bullet$$

Our observational semantics is not the same as in CCS [7] where not all τ operations can be removed. We, like CSP, are able to remove all τ actions (see Fig. 4).

Essentially the same definition as Definition 3 has appeared in [8, 9, 10], and see [10] for further comparison with the literature.

From the definition of an **observational semantics** ($\Rightarrow$) we define the traces of A, $Tr(\mathsf{A}) \triangleq \{\rho^\alpha | s_{\mathsf{A}} \overset{\rho^\alpha}{\Longrightarrow}\}$.

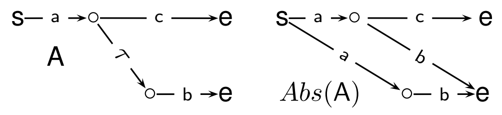


Figure 4: Action abstraction

The frame of reference of a LTS is its alphabet Alp. Usually in the process literature the alphabet Alp is fixed for all LTS and hence its role is not made explicit and the refinement rules so far mentioned do not permit a change in Alp.

But in the process literature there are known refinements that do permit a change in the frame of the LTS, however they are extensions of failure refinements. In [11] two refinements are considered where some actions of a concrete process are renamed. In one form of refinement they are renamed to be τ actions and in the other they are renamed to be δ actions.

Here all we do that is different is apply the same extension to trace refinement:

Definition 4 *tau and delta refinement.*

$$(\mathsf{A}_{Alp} \sqsubseteq_{Trd} \mathsf{C}_{Alp \cup New}) \triangleq (\mathsf{A}_{Alp} \sqsubseteq_{Tr} \mathsf{C}_{Alp \cup New} \delta_{New})$$
$$(\mathsf{A}_{Alp} \sqsubseteq_{Trt} \mathsf{C}_{Alp \cup New}) \triangleq (\mathsf{A}_{Alp} \sqsubseteq_{Tr} \mathsf{C}_{Alp \cup New} \tau_{New}) \quad \bullet$$

5.2 Event B frame refinement

If a new event is a refinement of a *skip* statement then the concrete machine will be *observationally equivalent* to the abstract machine.

$I_A(v) \wedge I_C(v,w) \wedge Pre_C(w) \wedge G_C(w) \wedge R_C(w,w') \Rightarrow \quad I_C(v,w')$	TAU_REF

The proof obligation TAU_REF is sufficient to establish $\sqsubseteq_{Trt}$

If a new event is a refinement of a *blocked* statement then the concrete machine will be a $\sqsubseteq_{Trd}$ refinement of the abstract machine. This delta refinement requires **no proof obligation**. At first glance this seems so liberal as to render refinement meaningless. Although "do nothing" is always a refinement of any specification and we can now add new events anywhere, it looks like we can refine any specification into any implementation. But we cannot.

The particular "do nothing" that is a refinement of A_{Alp} must have the same frame as A, i.e. $stop_{Alp}$. Delta refining $stop_{Alp}$ cannot add any action in Alp. Hence the specification A_{Alp} defines the safe behaviour of events in Alp and states nothing about events not in Alp.

$TAUOPERATIONS$ and $DELTAOPERATIONS$ are defined by extending B syntax. We will require that the sets are disjoint and that all operations in the refinement but not in the original machine are in one of these sets. B machine refinement will only require operations in $TAUOPERATIONS$ to satisfy the proof obligation TAU_REF.

Lemma 3 *Let* $\alpha(\mathsf{C}) = \alpha(\mathsf{A}) \cup N$ *and* N *is defined to be a set of TAUOPERATIONS.*

$\mathsf{A} \sqsubseteq_B \mathsf{C}$ *implies* $\mathsf{A} \sqsubseteq_{Trt} \mathsf{C}$

Proof: We will prove $\mathsf{A} \not\sqsubseteq_{Trt} \mathsf{C} \Rightarrow \mathsf{A} \not\sqsubseteq_{B} \mathsf{C}$ and using propositional logic we can infer $\mathsf{A} \sqsubseteq_{B} \mathsf{C} \Rightarrow \mathsf{A} \sqsubseteq_{Trt} \mathsf{C}$.

Assume $\mathsf{A} \not\sqsubseteq_{Trt} \mathsf{C}$ and hence $\exists \mathsf{t} \in Tr(\mathsf{C}\tau_{New}) \wedge \mathsf{t} \notin Tr(\mathsf{A})$

$$\begin{tikzcd}
& \circ \arrow[r, "\mathsf{a_A}"] \arrow[dd, "r"] & x \arrow[r, "skip"] \arrow[dd, "r"] & x \arrow[dd, "r"] & \\
s \arrow[ur, "\mathsf{init_A}"] \arrow[dr, "\mathsf{init_C}"'] & \not\hookrightarrow & \not\hookrightarrow & \not\hookrightarrow & \not\hookleftarrow \\
& \circ \arrow[r, "\mathsf{a_C}"] \arrow[rr, bend right, "\mathsf{a_C}"'] & \circ \arrow[r, "n_C"] & \circ \arrow[r, "\mathsf{c_C}"] & \cdots\cdots
\end{tikzcd}$$

Figure 5: $\mathsf{n_C} \in New$

Without loss of generality let $\mathsf{t} \triangleq \mathsf{a}; \mathsf{n}; \mathsf{c} \ldots$, let $\mathsf{n_C} \in New$ and let the greatest prefix of $\mathsf{a}; \mathsf{c} \ldots$ in $Tr(\mathsf{A})$ be a. Hence clearly from Fig. 5 $\mathsf{A} \not\sqsubseteq_{B} \mathsf{C}$. •

Lemma 4 *Let $\alpha(\mathsf{C}) = \alpha(\mathsf{A}) \cup N$ and N is defined to be a set of DELTAOPERATIONS.*
$\mathsf{A} \sqsubseteq_{B} \mathsf{C}$ *implies* $\mathsf{A} \sqsubseteq_{Trd} \mathsf{C}$.

The proof is the same as for Lemma 2 except we need to consider the special case of the δ operations. With B refinement $\sqsubseteq_B$ these are ignored and on the $\sqsubseteq_{Trd}$ they are also ignored. •

6 Continuing the Mobile Radio example from Section 4

The less abstract, lower-level view of the mobile radio $LLMR$ takes into account new features, in particular three new operations that do not appear anywhere in the high-level specification $HLMR$:

1. When the radio is *Busy* it may fade and when a fade occurs the radio is *Jammed*;
2. When the radio is *Jammed* it must be reset to the *Idle* state;

   This specification is very weak, it assumes that reset will only be called when $callState = Jam$;
3. Before the radio will work the user must select a suitable wave band.

Let us assume that the $LLMR$ description is a more accurate depiction of the actual radio. The high-level view $HLMR$ is meaningful if we assume that the essential operations sel and reset are always performed when needed and are never observed. But to make the high-level view $HLMR$ meaningful we must assume that fade is ignored or simply never occurs. We can view the refinement from $LLMR$ to $HLMR$ as adding this new feature and how to deal with it, while providing the service guaranteed in $HLMR$.

This difference in the interpretation of the new operations is reflected in the details of how the operations are removed from the high-level view $HLMR$:

```
REFINEMENT          $HLMR$
REFINES             $LLMR$
TAUOPERATIONS       sel,reset                                              $*$
DELTAOPERATIONS     fade                                                   $*$
SETS                $JCALLSTATES = CALLSTATES \cup \{Jam\}$
VARIABLES           $jcallState, jcurrChan, bandSelected$
INVARIANT           $jcallState \in JCALLSTATES \wedge bandSelected \in Bool$
                    $\wedge jcurrChan \in CHANNELS$
  $\wedge (callState, jcallState) \in \{(Idle, Idle), (Idle, Jam), (Busy, Busy)\}$   $Glue*$
  $\wedge currChan = jcurrChan$                                                  $Glue*$
  $\wedge bandSelected = FALSE \Rightarrow jcallState = Idle$                     $Reach*$
INITIALISATION      $jcallState = Idle \parallel bandSelected = FALSE \parallel$
                    $jcurrChan :\in CHANNELS$
OPERATIONS
   sel   $\triangleq$ PRE $band\_selected = FALSE$ THEN $band\_selected := TRUE$ END;
  reset  $\triangleq$ PRE $callState = Jam$ THEN $callState := Idle$ END;
  fade   $\triangleq$ SELECT $callState = Busy \wedge band\_selected = TRUE$ THEN
               $callState := Jam$
         END;
    do   $\triangleq$ PRE $jcallState = Busy \wedge bandSelected = TRUE$
               THEN $jcallState := Idle$ END;
    di   $\triangleq$ SELECT $jcallState = Busy \wedge bandSelected = TRUE$
               THEN $jcallState := Idle$ END;
 x←co    $\triangleq$ PRE $jcallState = Idle \wedge bandSelected = TRUE \wedge x \in CHANNELS$
               THEN
               CHOICE $jcallState := Busy \parallel jcurrChan := x$ OR
                     $skip$ OR $jcallState := Jam$
               END
         END;
  ci(x)  $\triangleq$ PRE $x \in CHANNELS$ THEN
               SELECT $jcallState = Idle \wedge bandSelected = TRUE$ THEN
                     $jcallState := Busy \parallel jcurrChan := x$ ELSE $skip$
               END
         END;
END
```

The $LLMR$ machine is taken from [2]. The only amendments are the addition of lines marked with a $*$ on the right-hand side. The first two additional lines state how to abstract the new operations and hence which proof obligations should be applied. The third and fourth new lines define the *gluing invariant* between $LLMR$ and $HLMR$. The final additional line defines the reachable set of nodes. Without this the proof obligations for the refinement of operation co(x) from $HLMR$ to $LLMR$ could not be satisfied.

In order to verify $LLMR \sqsubseteq HLMR$ we could either:

1. apply $\_\delta_{\{\mathsf{fade}\}}\tau_{\{\mathsf{sel,reset}\}}$ to the semantic model of HLMR and using automatic tools test for trace refinement; or

2. adopt the B approach.

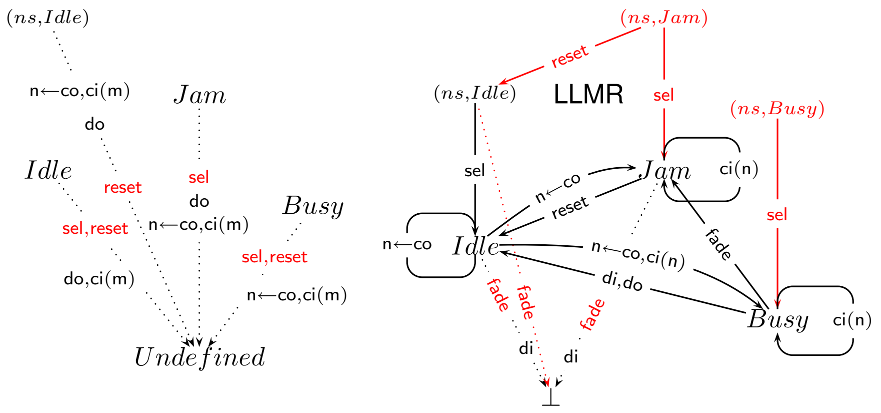


Figure 6: Mobile Radio Low Level

States that are unreachable by guaranteed behaviour may be deleted as, by refinement, the nondeterministic actions of the undefined behaviour that do reach those states can be deleted and consequently the states may become unreachable by any behaviour. Consequently states not selected and Jammed (ns, Jam) and not selected and Busy $(ns, Busy)$ may be deleted.

With the state-based approach in [2] they comment "One might say very loosely that one had refined the $HLMR$ model to the $LLMR$ model, but one could not attach any mathematical weight to such a statement." With the simple extension to Event B refinement we can see that we can give a formal definition of refinement so that $LLMR \sqsubseteq HLMR$.

7 Conclusion

Using the obvious relation between the relational semantics of B machines and the LTS semantics of processes we have a simple link between these two separate models. Using this link we apply the well-known delta refinement, defined on processes, to B machines. This gives two definitions of frame refinement of B machines: the usual tau refinement and the new delta refinement.

By applying a combination of tau and delta refinement to a problem taken from the literature we demonstrate both the usefulness of the two styles of frame refinement and their conceptually distinct roles. This enabled us to construct a formal refinement between machines that previously were only related by retrenchment [2].

References

[1] van Glabbeek, R.L.: The linear time - branching time spectrum I. the semantics of concrete sequential processes. In Bergstra, J., Ponse, A., Smolka, S., eds.: Handbook of Process Algebra. Elsevier Science, Amsterdam, The Netherlands (2001) 3–99

[2] Banach, R., Poppleton, M.: Retrenchment, refinement and simulation. In: Proc. ZB-00. Volume 1878 of LNCS. (2000) 304–323

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

[4] 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

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

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

[7] Milner, R.: Communication and Concurrency. Prentice-Hall International (1989)

[8] Brinksma, E., Rensink, A., Vogler, W.: Applications of fair testing. FORTE **69** (1996) IFIP Conference Proceedings.

[9] Valmari, A., Tienari, M.: Compositional Failure-based Semantics Models for Basic LOTOS. Formal Aspects of Computing **7** (1995) 440–468

[10] Reeves, S., Streader, D.: Atomic Components. In: ICTAC 2004. LNCS 3407. Springer-Verlag (2004) 128–139

[11] Fischer, C., Wehrheim, H.: Behavioural subtyping relations for object-oriented formalisms. LNCS **1816** (2000) 469–483