



Edition	1
Revision	0
Number of pages	43
Status	Review

EVENT-B:

Syntax and Proof Obligations in Atelier B

	Redaction	Verification	Approval
Nom	H. Ruyz Barradas	D. Déharbe	
Date	31/08/2020	21/10/2020	

Revision Table

Version	Date	Contributors	Contribution
1.0	31/08/2020	H. Ruiz Barradas (Clearsy)	Initial version

Contents

1	Introduction	5
2	Syntax	7
2.1	Specifications	7
2.2	Refinements	9
3	Proof Obligations	13
3.1	Refinement Proof Obligations	13
3.2	Feasibility Proof Obligation	14
3.3	Deadlock Freeness Proof Obligation	16
3.4	Non Divergence	18
3.5	Exclusiveness	20
3.5.1	Exclusiveness in Specifications	20
3.5.2	Exclusiveness in Refinements	21
3.6	Coverage	22
A	Resources	27
B	Proof Obligations	29
B.1	Definitions	29
B.2	Proof Obligations for Specifications	31
B.2.1	Feasibility (FIS) Proof Obligation	31
B.2.2	Deadlock Freedom (DLF) Proof Obligation	32
B.2.3	Exclusiveness (EXC) Proof Obligation	33
B.3	Proof Obligations for Refinements	34
B.3.1	Merge Refinement Correctness	34
B.3.2	Feasibility (FIS) Proof Obligation	35
B.3.3	Relative Deadlock Freedom (DLF) Proof Obligation	36
B.3.4	Skip Refinement Correctness Proof Obligation	37
B.3.5	Numeric Variant Proof Obligation	38

B.3.6	Non Divergence Proof Obligation	39
B.3.7	Exclusiveness (EXC) Proof Obligation	40
B.3.8	Coverage Proof Obligation	43

Chapter 1

Introduction

System modelisation using the Event B approach is supported by the Atelier B tool. To use this kind of modelling, the project type *System* must be specified when a new project is created. A System project type is made up of three kinds of components:

1. system specification components (files with a `*.sys` extension),
2. refinement components (`*.ref` extension),
3. closing implementation components (`*.imp` extension.): their purpose is the generation of proof obligations (PO) allowing for feasibility proof of constants and variables.

The syntax is very similar to that of the B language for the development of software component. In this document, we will refer to this latter language as *B Software* and its reference manual [6].

The proof obligations to be generated for a system project can be selected through the graphical interface of Atelier B in the settings of the project. These PO include the classical invariant preservation proofs, which are always generated, the well definedness proofs and optional proof obligations specific to Event B.

The next sections detail the syntactical particularities of the System components and the specific Event B proof obligations.

Chapter 2

Syntax

The syntax of specifications and refinements in a System project is mainly a constrained version of the syntax of specification and refinements in a Software project of the Atelier B tool.

The following sections detail the syntax of specifications and refinements signalling the constraints to be respected with respect to the syntax of B Software.

2.1 Specifications

A system specification must be declared in a component having an `*.sys` extension.

The syntax of this kind of component is given by the syntactical category 1 which is a constrained version of the syntax of the *Machine* components in B Software¹.

Syntactical Category 1 (*System Specification*)

```
System_Specification ::=  
  SYSTEM Identifier  
  System_Specification_Clause  
END
```

where

```
System_Specification_clause ::=  
  | Sees_clause  
  | Sets_clause  
  | Definitions_clause
```

¹The syntactical categories specified in this document are given in the style of [6]

| *Concrete_Constants_clause*
 | *Abstract_Constants_clause*
 | *Properties_clause*
 | *Concrete_Variables_clause*
 | *Abstract_Variables_clause*
 | *Invariant_clause*
 | *Assertions_clause*
 | *Initialisation_clause*
 | *Operations_clause*

The current version of Atelier B also allows references to included (with promoted operations), or extended System specifications, but there is no semantic meaning for these type of references in Event B models. For this reason a System specification does not consider included or extended specifications.

The *Operations_clause* in the list of *System_Specification_clause* is the only clause where the syntax is a restriction of the corresponding clause in software projects. All the other clauses in the list follow the syntax of B Software.

The syntax of the *Operations_clause* is given by the Syntactical Category 2.

Syntactical Category 2 (*System Operation Clause*)

Operations_clause ::= (OPERATIONS | EVENTS) *Event*^{+", "}

Event ::= *Identifier* = *Event_body*

Event_body ::= *Block_substitution* | *Identity_substitution* | *Any_substitution*
 | *Simple_substitution* | *Multiple_substitution* | *Select_substitution*

The *Block*, *Identity* and *Any* substitutions follow the same syntactical rules as the corresponding substitutions in B Software projects:

BEGIN		ANY <i>Identifier</i> ^{+", "}
<i>Substitution</i>	skip	WHERE <i>Predicate</i>
END		THEN <i>Substitution</i>
		END

where *Substitution* is any kind of substitution allowed in the Machine components in B Software. *Simple_substitution* and *Multiple_substitution* also have the same syntactical rules as the corresponding substitutions in B Software.

The *Select_substitution* is a restriction of the corresponding substitution in B Software. The syntax of the *Select_substitution* is given in the syntactical category 3.

Syntactical Category 3 (*Select Substitution*)

```
Select_substitution ::=  
  SELECT Predicate THEN Substitution  
  (WHEN Predicate THEN Substitution)*  
END
```

A *Select_substitution* occurring as the *Event_body* of an event does not allow the ELSE clause of a *Select* substitution in B Software. However, inside the *Select* or *When* part of the *Select_substitution*, the *Substitution* can be any kind of Machine substitution allowed in B Software.

2.2 Refinements

A system specification can be refined through a sequence of refinements declared in components having the extension `*.ref`. The syntax of this kind of components is given by the syntactical category 4 which is mainly a constrained version of the syntax of *Refinement* components in B Software.

Syntactical Category 4 (*Refinement Specification*)

```
Refinement_Specification ::=  
  REFINEMENT Identifier  
  REFINES Identifier  
    Refinement_Clause  
  END
```

where

```
Refinement_clause ::=  
  | Sees_clause
```

- | *Sets_clause*
- | *Definitions_clause*
- | *Concrete_Constants_clause*
- | *Abstract_Constants_clause*
- | *Properties_clause*
- | *Concrete_Variables_clause*
- | *Abstract_Variables_clause*
- | *Invariant_clause*
- | *Variant_clause*
- | *Assertions_clause*
- | *Initialisation_clause*
- | *Operations_clause*

The *Variant_clause* in a refinement component is a new clause which does not exist in B Software refinements. The syntax of this clause is given by the syntactical category 5:

Syntactical Category 5 (*Variant Clause*)

Variant_clause ::= VARIANT *Expression*

where *Expression* in this *Variant_clause* is an expression of NATURAL type, similar to the *Expression* in the *Variant* clause of a *While* substitution in B Software. The *Non-Divergence* proof obligation, stated in the next section, uses this *Expression* to prove that new events in the current refinement only introduce bounded stuttering steps.

The *Operations_clause* is the only refinement clause whose syntax differs from B Software.

The refinement *Operations_clause* is almost the same as the system *Operations_clause* stated by the syntactical category 2. The only one difference is the non terminal symbol *Event* which is defined by the following syntactical category:

Syntactical Category 6 (*Refined Events*)

Event ::= Identifier [ref Identifier⁺,'] = *Event_body*

The keyword `ref` can be used in three different ways:

1. Rename events

Abstract Event

Event_a = *Substitution*

Concrete Event

Event_c ref Event_a = *Substitution*

The abstract event Event_a is refined by the concrete event Event_c with the same or a different *Substitution*.

2. Split events

Abstract Event

$\text{Event_a} = \text{Substitution}$

Concrete Event

$\text{Event_c1} \text{ ref } \text{Event_a} = \text{Substitution}_1$

$\text{Event_c2} \text{ ref } \text{Event_a} = \text{Substitution}_2$

The abstract event Event_a is refined (split) by two concrete events Event_c1 and Event_c2 .

3. Merge events

Abstract Events

$\text{Event_a1} = \text{Substitution}_1$

$\text{Event_a2} = \text{Substitution}_2$

Concrete Event

$\text{Event_c} \text{ ref } \text{Event_a1}, \text{Event_a2} = \text{Substitution}$

The abstract events Event_a1 and Event_a2 are refined (merged) into the concrete event Event_c .

It must be noted that the merge of events in the Atelier B implementation of Event B is more general than the merge of events as presented by the creator of Event-B [2]. In Atelier B, the bodies of the abstract events are not necessarily the same.

Events in the *Operation_clause* of a refinement are partitioned into two sets: implicit events (*IE*) and explicit events (*EE*). The set *EE* contains explicitly refined events as well as new events appearing in the current component. The set *IE* is equal to the set difference $RE - EE$, where *RE* denotes all the events (explicit and implicit) of the component referenced by the *REFINES* clause, if any, or all the events in a system specification otherwise.

When an abstract event, named *a* is refined through the *ref* keyword, by a concrete event, named *c*, the *Operation_clause* of the refinement contains, among others, the abstract event *a* (implicit event) and the refined one *c* (explicit event). The Proof Obligation Generator treats the implicit and explicit events in the same way, generating Invariant Preservation PO for both kind of events. To avoid the proof obligation generation for an implicit refined event, it must be *closed* by a miraculous substitution as stated by the Definition 1.

Definition 1 (*Closed Event*)

An abstract event *Abstract* is closed by declaring it as miraculous substitution as follows:

Abstract = SELECT 0=1 THEN skip END

The last syntactical category introduced in the refinement of Event B models implemented by the Atelier B is a new substitution named *Witness*. The rationale for this substitution comes from the Event B reference in [2] where refined events contain the *with* clause to denote a witness in the refined event for each disappearing parameter of the abstract event and for each disappearing abstract variable assigned in a non deterministic way in the abstract event.

The implementation of the *Witness* substitution in Atelier B is a weak form of the *with* clause implemented in the Rodin platform [9]. In Atelier B, the witness is only used for each disappearing parameter of a refined ANY substitution. Moreover, only deterministic witnesses are allowed in the Atelier B through equality predicates. The *Witness* substitution is not mandatory.

The syntax of the *Witness* substitution is given in the syntactical category 7.

Syntactical Category 7 (*Witness*)

Witness_substitution ::=
WITNESS (*Identifier* = *Expression*)⁺ "&"
THEN *Substitution*
END

where *Identifier* is a variable declared in the abstract event not appearing in the refined one and *Expression* is an expression of the same type as the removed variable.

A *Witness* substitution should appear in the body of refined events, for example :

```
BEGIN
  WITNESS xx = ee THEN Substitution END
END

or

SELECT Expression THEN
  WITNESS xx = ee THEN Substitution END
END

or

ANY yy WHERE Predicate THEN
  WITNESS xx = ee THEN Substitution END
END
```

Chapter 3

Proof Obligations

This section presents Event B Proof Obligations (POs) implemented in Atelier B. First, the mandatory POs for splitting or merging events are commented. Then five subsections describe the following optional proof obligations:

- Feasibility PO.
- Deadlock freedom PO.
- Non divergence PO.
- Exclusiveness PO.
- Coverage PO.

The feasibility, deadlock freedom and exclusiveness POs apply to system specifications and refinements. Non Divergence and Coverage POs apply only to system refinements. The POs are presented together with an explanation of their usefulness in the context of a system specification or refinement. In the appendices B.2 and B.3, these POs are also given in terms of XPATH expressions of IBXML documents [7].

3.1 Refinement Proof Obligations

The syntactical category 6 concerning the refined events introduces the `ref` clause. As indicated, this keyword can be used in three different ways to rename, split or merge abstract events.

1. Renaming events is not different from the classical refinement in the B Method, for this reason it is not commented here.

2. The split of an abstract event a into a set of refined ones $ref(a)$ leads to the generation of refinement POs to demonstrate that a is refined by each e' in $ref(a)$; as it is also a classical refinement, it is not further commented in this section.
3. The PO for merging abstract events is mandatory and is always generated for the Atelier B.

A set of abstract events can be merged into a concrete event. In this case, it must be proved that the choice of abstract events is currently refined by the concrete one as stated by the proof obligation 1.

Proof Obligation 1 (*Merge Refinement Correctness*)

Let c be an event in a system refinement with invariant J refining through its `ref` clause a set of abstract events a_1, \dots, a_n in an system specification or refinement with invariant I . The Merge Refinement correctness proof obligation is:

$$ctx \wedge I \wedge J \vdash [c] \neg [\text{CHOICE } a_1 \text{ OR } \dots \text{ OR } a_n \text{ END}] \neg J$$

where ctx is the proof context.

It must be noted that in the case of guarded commands (`SELECT` substitutions without `WHEN-THEN` clauses) having the same body, PO 1 becomes equivalent to *MRG* Proof Obligation in [2].

The mandatory proof obligation to prove the correctness of new events introduced in a refinement is given in section 3.4 page 18.

3.2 Feasibility Proof Obligation

In Event B [2], the *guard* concept denotes the predicate, built on the sets, constants and variables of a system or refinement, representing the *necessary* conditions for an event to occur. The constraints in the syntax of events in [2] exhibit the guard of events in an explicit way.

However, the syntax of events in the Event B implementation of Atelier B, according to the syntactical category 2, allows embedded arbitrary substitutions in the body of the *Block*, *Any* and *Select* substitutions. Therefore, in Atelier B, two guards concepts are introduced : *explicit* and *implicit* guards.

The explicit guard of an event is a syntactic guard, defined according to the substitution defining the event. For the *Block* and *Identity* substitutions, the explicit guard is defined by `true`. For *Select* substitutions with only one guarded command, that is, no `WHEN` clause at

all, the explicit guard is the guard of the guarded command. If the *Select* substitution contains *WHEN* clauses, its explicit guard is **true**. Finally, the explicit guard of the *Any* command, is the predicate of its *WHERE* clause, existentially quantified over the *Any* variables. The definition of the explicit guard is given in definition 2.

Definition 2 (*Explicit Guard*)

For any *Substitution* s its explicit guard $\text{grd}(s)$ is defined as follows:

$$\text{grd}(s) = \begin{cases} P & \text{if } s = \text{SELECT } P \text{ THEN } S \text{ END} \\ \exists x \cdot (P) & \text{if } s = \text{ANY } x \text{ WHERE } P \text{ THEN } S \text{ END} \\ \text{true} & \text{for any other type of } \textit{Substitution } s \end{cases}$$

It should be noted that the explicit guard $\text{grd}(s)$ of an event e , defined by the substitution s , does not represent the necessary conditions for e to occur, since to do so, it needs to have enabled all the guards of the embedded guarded commands in s . The necessary condition for event e to occur is its implicit guard, defined in definition 3 [1].

Definition 3 (*Implicit Guard*)

For any *Substitution* s its implicit guard $\text{fis}(s)$ is defined as follows:

$$\text{fis}(s) = \neg [s] \text{ false}$$

The explicit and implicit guards are related through the Property 1.

Property 1

For any *Substitution* s , the implicit guard is stronger than or equivalent to the explicit guard:

$$\text{fis}(S) \Rightarrow \text{grd}(S)$$

Property 1 is proved by structural induction in the structure of S . If S is the *Select* substitution defined in definition 2, its implicit guard is $P \wedge \neg [S] \text{ false}$ which implies the explicit guard. If S is the *Any* substitution defined in definition 2, its implicit guard is $\exists x \cdot (P \wedge \neg [S] \text{ false})$ which implies its explicit guard. In any other case of the structure of S its explicit guard is defined as **true**, which is implied by any predicate and in particular by $\text{fis}(S)$.

As an example of the instantiation of property 1, if $\text{fis}(S)$ is equal to **true**, where S is the substitution in definition 2, the explicit and implicit guards of the *Select* and *Any* substitutions of that definition are equivalent. However, having a substitution S as follows:

$$S = \text{SELECT } P_1 \text{ THEN } S_1 \text{ WHEN } P_2 \text{ THEN } S_2 \text{ END}$$

where $\text{fis}(S_1) = \text{fis}(S_2) = \text{true}$, we have $\text{fis}(S) = P_1 \vee P_2$ and $\text{grd}(S) = \text{true}$; in this case we have $\text{fis}(S) \Rightarrow \text{grd}(S)$. If the converse of this implication is proved, that is $\text{grd}(S) \Rightarrow \text{fis}(S)$, it means that the explicit and implicit guards are equivalent.

Apart from allowing the equivalence proof between the explicit and implicit guards, the converse of the implication $\text{fis}(S) \Rightarrow \text{grd}(S)$ defines the *Feasibility* PO of events in an Event B model. This proof obligation states, from a logical point of view, that an event is not able to establish any post-condition, that is, it is not miraculous, when it occurs in a state satisfying its external guard. From an operational point of view, it states the existence of a state reached after the event occurs in an state where its explicit guard holds. The Feasibility PO is therefore defined as follows:

Proof Obligation 2 (*Feasibility Proof Obligation –FIS–*)

For any event E of a System Specification or Refinement, the Feasibility Proof Obligation is:

$$ctx \vdash (\text{grd}(E) \Rightarrow \text{fis}(E))$$

where ctx is the proof context of the Specification or Refinement.

The PO 2 with its proof context ctx is formally defined in section B.2.1 of the appendix for system specifications and section B.3.2 for system refinements. It should be noted that PO 2 is equivalent to the proof obligation *avt/act/FIS* defined in [2].

The resource to be set in the Atelier B for generation of the Feasibility PO is the following:

Resource 1 (*Feasibility Proof Obligation*)

$$\text{ATB*POG*Generate_EventB_Feasibility_PO}$$

3.3 Deadlock Freeness Proof Obligation

In the case of a system specification reaching a state where no event can occur, that state is known as a *deadlock* state.

In order to ensure that a system specification will never reach a deadlock state, the Deadlock Freeness Proof Obligation is defined to guarantee that at least one event of the system can occur at any reached stated:

Proof Obligation 3 (*Deadlock Freeness Proof Obligation –DLF–*)

Let E be the set of events in a system specification, I the invariant of the system and ctx the proof context. The Deadlock Freedom PO is:

$$ctx \wedge I \vdash \exists e \cdot (e \in E \wedge \mathbf{grd}(e))$$

where $\mathbf{grd}(e)$ denotes the explicit guard of e given in definition 2.

To be meaningful, this Proof Obligation needs the FIS PO to be proved.

If FIS POs are proved, it is guaranteed that the explicit guard of any event in the system is equivalent to its implicit guards, and therefore the DLF PO effectively guarantees that the system does not reach a deadlock state. This PO and its ctx proof context are formally defined in section B.2.2 of the appendix.

The refinement of an abstract event by a concrete one has a consequence that the implicit guard of the refined event becomes stronger than the implicit guard of the abstract one. Therefore, if the DLF PO has been proved in a system, it is not guaranteed that its refinement does not reach a deadlock state. In order to guarantee that the refined system does not deadlock more often than the abstract one, it must be proved that the implicit guard of any abstract event guarantees, at least, the implicit guard of a refined or new event. The PO to guarantee that a refinement does not deadlock more often than the abstract one is known as Relative Deadlock Freedom PO:

Proof Obligation 4 (*Relative Deadlock Freeness PO –DLF–*)

Let C be the set of events in a system refinement with invariant J and A the set of events contained in the component referenced by the `REFINES` clause of the refinement and I its invariant. The Relative Deadlock Freedom PO is:

$$ctx \wedge I \wedge J \vdash \forall e \cdot (e \in A \wedge \mathbf{grd}(e) \Rightarrow \exists e' \cdot (e' \in C \wedge \mathbf{grd}(e')))$$

where $\mathbf{grd}(e)$ and $\mathbf{grd}(e')$ denote the explicit guard of events given in definition 2 and ctx the proof context.

To be meaningful, this Proof Obligation needs the FIS PO to be proved.

This PO and its *ctx* proof context are formally defined in section B.3.3 of the appendix.

The Deadlock Freeness PO and the Relative Deadlock Freeness PO are equivalent to the DLF proof obligation defined in [2].

The Atelier B resource to enable the generation of the Deadlock Freeness POs is:

Resource 2 (*Deadlock Freeness Proof Obligation*)

ATB*TC*EventB_DeadlockFreeness

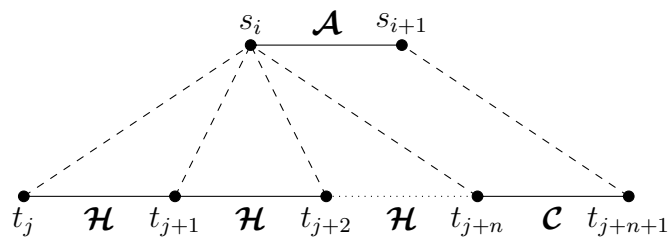
3.4 Non Divergence

Apart from *Data Refinement*, which is the classical refinement in the B method [1], Event B allows *Atomicity Refinement* through the introduction of *new events* [2]. These new events must satisfy two conditions:

1. They are not allowed to modify the abstract state.
2. They are only allowed to introduce *bounded* stuttering steps.

The first condition constrains the effect of new events to the components where they are introduced and their refinements. The second guarantees that any trace of the refinements cannot contain infinite sequences of new events.

A classical use of atomicity refinement is the decomposition of the execution of an abstract event \mathcal{A} into a series of refined steps $\mathcal{H}, \mathcal{H}, \mathcal{H}, \dots, \mathcal{C}$, as depicted in the following scheme:



In this scheme, the abstract event \mathcal{A} is refined by the concrete one \mathcal{C} while a new event \mathcal{H} is introduced. It is supposed that \mathcal{A} transforms the abstract state s_i into s_{i+1} whereas \mathcal{C} transforms the refined state t_{j+n} into t_{j+n+1} . The departure state of the abstract event is linked by the gluing invariant to the departure state of the refined one and the arrival state of \mathcal{A} is linked to the arrival state of \mathcal{C} through the same invariant; it is denoted by the dashed lines in the scheme.

The execution of new events modifies the refined state through a sequence of transformations t_j, \dots, t_{j+n} . All these transformations are linked to the abstract state s_i by the gluing invariant, meaning that no state transformation is perceived in the abstract state. Moreover, the sequence of transformations is finite, taking at most n steps, starting in a refined state t_j and leading to a final state t_{j+n} , where the refined event \mathcal{C} can start its execution.

Event B defines two kinds of POs to ensure that new events satisfy the two conditions previously stated. For the first condition, in order to avoid abstract state modifications, each new event \mathcal{H} must refine the *identity* substitution *skip*:

Proof Obligation 5 (*Skip Refinement Correctness*)

Let H be a new event in a System Refinement with invariant J refining an abstract system with invariant I . The Skip Refinement Correctness PO is:

$$ctx \wedge I \wedge J \vdash [H] \neg [\text{skip}] \neg J$$

where ctx is the proof context.

In Atelier B, PO 5 is mandatory and is always generated when a new event is introduced in a refinement. The PO and its ctx proof context are formally defined in section B.3.4.

For the second condition, in order to ensure that new events introduce bounded stuttering steps, two POs are defined. These POs are stated to guarantee that the variant expression V , defined in the Variant clause of refinements introducing new events (syntactical category 5), is indeed a natural type expression decremented by any new event:

Proof Obligation 6 (*Numeric Variant –NAT–*)

Let V be the expression of the *Variant* clause in a refinement with invariant J refining an abstract system with invariant I . The Numeric Variant PO is:

$$ctx \wedge I \wedge J \vdash V \in \mathbb{N}$$

where ctx is the proof context.

PO 6 and its ctx proof context are formally defined in section B.3.4. It must be noted that PO 6 is slightly stronger than the *evt/NAT* PO defined in [2] as the guard of new events are not taken into account in the antecedents of PO 6.

In order to guarantee that new events only introduce bounded stuttering steps, the expression in the *Variant* clause must be decremented by any new event:

Proof Obligation 7 (Non Divergence –NDI–)

Let V be the expression of the *Variant* clause in a refinement with invariant J refining an abstract system with invariant I . For any new event H defined in the refinement, the Non Divergence PO is:

$$ctx \wedge I \wedge J \vdash [v := V] [H] V < v$$

where ctx is the proof context and v is a fresh variable such that $v \setminus (J, I, ctx)$.

The PO 7 and its ctx proof context are formally defined in section B.3.6. This PO 6 is the *evt/VAR* PO defined in [2].

The Atelier B resources to be set to enable Non Divergence POs are:

Resource 3 (Non Divergence Proof Obligation)

ATB*POG*Generate_EventB_Non_Divergence_PO

ATB*BCOMP*Generate_EventB_Non_Divergence_PO

3.5 Exclusiveness

Event B makes no *fairness* assumptions on the occurrence of events. When two or more events have enabled their guards, one of these events occurs, but it is not known which. The choice of the event to occur is non-deterministic. However, if the guards of the events are exclusive, the non deterministic occurrence of events becomes deterministic.

The Atelier B resource to be enable the Exclusiveness POs is:

Resource 4 (Exclusiveness Proof Obligation)

ATB*POG*Generate_EventB_Exclusivity_PO

3.5.1 Exclusiveness in Specifications

Exclusiveness may be checked with the following proof obligation:

Proof Obligation 8 (Exclusiveness in System Specifications –EXC–)

Let E be the set of events in a system specification, (E, \prec) a strict total order, I the invariant

of the system and ctx the proof context. The Exclusiveness PO is:

$$ctx \wedge I \vdash \forall(e, f) \cdot (e \mapsto f \in E \times E \wedge e \prec f \Rightarrow \neg(\text{grd}(e) \wedge \text{grd}(f)))$$

where $\text{grd}(e)$ and $\text{grd}(f)$ denote the explicit guard of e and f given in definition 2.

To be meaningful, this PO needs the FIS PO to be proved.

The PO 8 and its proof context ctx are formally defined in section B.2.3 of the annex.

3.5.2 Exclusiveness in Refinements

If the exclusiveness of $\text{grd}(e)$ and $\text{grd}(f)$ has been proved, for two abstract events e and f , then the exclusiveness of $\text{grd}(e')$ and $\text{grd}(f')$ is guaranteed for their respective refinements e' and f' .

If an abstract event e is refined by two or more events e_1, \dots, e_n , it must be proved that $\text{grd}(e_1), \dots, \text{grd}(e_n)$ are exclusive. Moreover, if new events are introduced, the exclusiveness of the explicit guards of the new events with respect to the refined ones and among themselves must be proved to preserve determinacy:

Proof Obligation 9 (*Exclusiveness in Refinements –EXC–*)

Case Split of Events

For any abstract event e in a system specification or refinement with invariant I , let S_e be the set of events refining e with $\text{card}(S_e) > 1$ in a refinement with invariant J and (S_e, \prec) a strict total order. The Exclusiveness PO:

$$ctx \wedge I \wedge J \vdash \forall(e', f') \cdot (e' \mapsto f' \in S_e \times S_e \wedge e' \prec f' \Rightarrow \neg(\text{grd}(e') \wedge \text{grd}(f')))$$

Case New Events

Let $\{R, H\}$ be a partition of the set of events in a system refinement with invariant J , where any event e' in R is a refinement of an abstract event e in a system specification or refinement with invariant I or an implicit event and any h in H , is a new event defined in the refinement. Moreover, let (H, \prec) be a strict total order.

The Exclusiveness PO for new events vs refined ones is:

$$ctx \wedge I \wedge J \vdash \forall(h, e) \cdot (h \in H \wedge e \in R \Rightarrow \neg(\text{grd}(h) \wedge \text{grd}(e)))$$

The Exclusiveness PO for new events is:

$$ctx \wedge I \wedge J \vdash \forall(e, f) \cdot (e \mapsto f \in H \times H \wedge e \prec f \Rightarrow \neg(\text{grd}(e) \wedge \text{grd}(f)))$$

To be meaningful, this Proof Obligation needs the FIS PO to be proved.

As other POs using the explicit guard, PO 8 and PO 9 state that FIS PO must be proved in order to ensure that the external and internal guards are equivalent.

The Exclusiveness POs are formally defined in section B.3.7.

3.6 Coverage

The non deterministic choice in the occurrence of events in a system specification can be considered as an *external non determinism* [8]; it models external choices made by the environment. However, this external non determinism can be lost when the system is being refined because the guards of events are strengthened by refinement.

A motivating example, in the framework of Action Systems, showing the lost of external non determinism is given in [4]. That example is coded in Event B in figure 3.1. In this example, system *SS* models a vending machine dispatching *tea* or *coffee*. These goods are modeled by events having the same names in system *SS*. At any time, events *tea* or *coffee* can occur as their implicit guards are *true*. The choice between these events is made by the external environment synchronously engaging in events *tea* or *coffee* as demanded by the user of the vending machine ¹. Figure 3.1 presents a possible refinement *TT* of system *SS*. In this refinement, events *tea* and *coffee* become guarded commands. Their external guards, which are equivalent to the internal ones, since these events are feasible, depend on the value of the *gg* variable. In this refinement, both events refine *skip* and the guards of the concrete events are stronger than the abstract ones. The choice between events does not depend in the environment any more, it is determined internally by the vending machine.

In order to avoid the loss of external non determinism in refinements, Atelier B implements the Coverage PO:

Proof Obligation 10 (Coverage –COV–)

Let *H* be the set of new events in a System Refinement with invariant *J* and *A* the set of events contained in the component referenced by the *REFINES* clause of the refinement and *I* its invariant. For any event *e* in *A* let *S_e* be the set of events refining *e*. The Coverage PO for abstract event *e* is

$$ctx \wedge I \wedge J \vdash \text{grd}(e) \Rightarrow \exists e' \cdot (e' \in S_e \cup H \wedge \text{grd}(e'))$$

¹The composing approach of Action Systems through a synchronous parallel operator among actions, described in [4], as been also proposed in the Event B framework [5]. In this context, the *external* and *internal* non determinism terms, imported from CSP [8], can be used in Event B.

<pre> SYSTEM SS EVENTS tea = BEGIN skip END; coffee = BEGIN skip END END </pre>	<pre> REFINEMENT TT REFINES SS VARIABLES gg INVARIANT gg : BOOL INITIALISATION gg :: BOOL EVENTS tea = SELECT gg = TRUE THEN gg :: BOOL END; coffee = SELECT gg = FALSE THEN gg :: BOOL END END </pre>
---	--

Figure 3.1: Loss of External Non Determinism

where $\text{grd}(e)$ and $\text{grd}(e')$ denote the explicit guard of events given in definition 2 and ctx the proof context.

To be meaningful, this Proof Obligation needs the FIS PO to be proved.

PO 10 is formally defined in section B.3.8. It must be noted that the Coverage PO is stronger than the Relative Deadlock Freedom PO 4. It means that the proof of absence of relative deadlock in a refinement does not need the DLF PO when the Coverage PO is proved.

Moreover, in [3] the Coverage PO is given as a sufficient condition to guarantee the preservation of a form of liveness properties in a refinement. In that proposal, liveness properties (i.e. dynamic constraints) take the form of a *leads to* property, where it can be specified and proved that the execution of a system in a state where a certain predicate P holds eventually leads to another state where a predicate Q holds. Refinement does not preserve this kind of properties, for this reason the *progress condition* [4] is given as a sufficient condition to preserve these dynamic constraints. When all the events in a system specification or refinement are involved in the *leads to* property, the PO to guarantee the preservation of that dynamic constraint take the form of the Coverage PO, referenced as the “progress condition” in [3].

The Atelier B resource enabling the Coverage PO is:

Resource 5 (*Coverage Proof Obligation*)

ATB*POG*Generate_EventB_Coverage_PO

Bibliography

- [1] J.-R. Abrial, *The B-Book, Assigning Programs to Meanings*, Cambridge University Press, 1996.
- [2] J.-R. Abrial, *Modelling in Event-B, System and Software Engineering*, Cambridge University Press, 2010.
- [3] J.-R. Abrial, L. Mussat, *Introducing dynamic constraints in B*, In B'98 Recent Advances in the Development and Use of the B Method, LNCS 1393, Springer-Verlag, 1998.
- [4] M.J. Butler, *Stepwise Refinement of Communicating Systems*, Science of Computer Programming, 1996.
- [5] M.J. Butler, *Decomposition Structures for Event-B*, In 7th International Conference of Integrated Formal Methods IFM 2009, LNCS 5423, Springer-Verlag, 2009.
- [6] *Manuel de Référence du Langage B*, Clearsy, Version 1.8.10.
- [7] *Proof Obligation Generator - Parametrization*, Version 0.1.
- [8] C.A.R. Hoare, *Communicating Sequential Process*, Prentice Hall International, 1985.
- [9] J.-R. Abrial, *A System Development Process with Event-B and the Rodin Platform*. In ICFEM 2007. Lecture Notes in Computer Science, vol 4789. 2007.

Appendix A

Resources

PO	Resource
Coverage	ATB*POG*Generate_EventB_Coverage_PO
Deadlock freeness	ATB*TC*EventB_DeadlockFreeness
Exclusiveness	ATB*POG*Generate_EventB_Exclusivity_PO
Feasibility	ATB*POG*Generate_EventB_Feasibility_PO
Non divergence	ATB*POG*Generate_EventB_Non_Divergence_PO
	ATB*BCOMP*Generate_EventB_Non_Divergence_PO

Appendix B

Proof Obligations

B.1 Definitions

In the following two appendices, the Atelier B POs for Event B are described as they are coded in the parametrization file `paramGOPSsystem.xsl` of the PO Generator. The POs in these annexes are not given in the XSL syntax. However, some XPATH expressions are given to reference elements in IBXML documents [7]. In this annex some definitions used in the POs are given.

- **Sequence conjunction**

Let l be a sequence of predicates. $and_join(l)$ is defined as follows:

$$and_join(l) = \begin{cases} head(l) \wedge and_join(tail(l)) & \text{if } l \neq [] \\ true & \text{if } l = [] \end{cases}$$

- **Abstract Sets**

Let $Sets/*$ be a XPath in a `*.ibxml` document. The sequence of abstract sets as predicates is defined by:

$$abstract_sets(Sets/*) = [s \in FIN_1(INTEGER) \mid s \text{ in } Sets/* \wedge is_abstract_set(s)]$$

where $is_abstract_set(s)$ is equal to *true* if s is not defined as enumerated set $s = \{e_1, \dots\}$ in the XPath expression.

The sequence of identifiers of abstract sets is defined by:

$$abstract_set_ids(Sets/*) = [s \mid s \text{ in } Sets/* \wedge is_abstract_set(s)]$$

- **Enumerated Sets**

Let $Sets/*$ be a XPath in a $*.ibxml$ document. The sequence of enumerated sets as predicates is defined by:

$$enumerated_sets(Sets/*) = [s = t \mid s = t \text{ in } Sets/*]$$

where the equality $s = t$ in the sequence denotes the enumerated set s in the XPath expression defined by the enumeration t .

- **All Sets**

Let $Sets/*$ be a XPath in a $*.ibxml$ document. The sequence of all sets as predicates is defined by:

$$all_sets(Sets/*) = abstract_sets(Sets/*) \hat{\ } enumerated_sets(Sets/*)$$

- **B_definitions**

$$\begin{aligned} NAT &= 0 \dots MAXINT \wedge \\ INT &= MININT \dots MAXINT \end{aligned}$$

B.2 Proof Obligations for Specifications

B.2.1 Feasibility (FIS) Proof Obligation

For any event e of a system specification in `/Machine/Operations/*`, let $\text{grd}(e)$ and $\text{fis}(e)$ be the explicit and implicit guards given in definitions 2 and 3. The Feasibility PO to prove for event e is the following:

$$\begin{aligned} & B_definitions \wedge \\ & / * \text{Sets and Properties from Seen components:} * / \\ & \text{and_join}(\text{all_sets}(/Machine/Sees/Machine/Sets/*)) \wedge \\ & \text{and_join}(/Machine/Sees/Machine/Properties/*) \wedge \\ & / * \text{Sets and Properties of the System} * / \\ & \text{and_join}(\text{all_sets}(/Machine/Sets/*)) \wedge \\ & /Machine/Properties/* \wedge \\ & / * \text{Invariant and Assertions from Seen components:} * / \\ & \text{and_join}(/Machine/Sees/Machine/Invariant/*) \wedge \\ & \text{and_join}(/Machine/Sees/Machine/Assertions/*) \wedge \\ & / * \text{Invariant and Assertions of the System} * / \\ & /Machine/Invariant/* \\ & /Machine/Assertions/* \\ & \Rightarrow \\ & (\text{grd}(e) \Rightarrow \text{fis}(e)) \end{aligned}$$

B.2.2 Deadlock Freedom (DLF) Proof Obligation

Let E be the set of events of a system specification in $/Machine/Operations$. For any event e of E , let $\mathbf{grd}(e)$ be the explicit guard of e given in Definitions 2. The Deadlock Freedom PO to prove is the following:

$$\begin{aligned}
 & B_definitions \wedge \\
 & / * \text{Sets and Properties from Seen components:} * / \\
 & \text{and_join}(\text{all_sets}(/Machine/Sees/Machine/Sets/*)) \wedge \\
 & \text{and_join}(/Machine/Sees/Machine/Properties/*) \wedge \\
 & / * \text{Sets and Properties of the System} * / \\
 & \text{and_join}(\text{all_sets}(/Machine/Sets/*)) \wedge \\
 & /Machine/Properties/* \wedge \\
 & / * \text{Invariant and Assertions from Seen components:} * / \\
 & \text{and_join}(/Machine/Sees/Machine/Invariant/*) \wedge \\
 & \text{and_join}(/Machine/Sees/Machine/Assertions/*) \wedge \\
 & / * \text{Invariant and Assertions of the System} * / \\
 & /Machine/Invariant/* \\
 & /Machine/Assertions/* \\
 & \Rightarrow \\
 & \exists e \cdot (e \in E \wedge \mathbf{grd}(e))
 \end{aligned}$$

To be meaningful, the DLF PO needs the FIS PO to be proved.

B.2.3 Exclusiveness (EXC) Proof Obligation

Let E be an injective sequence of events of a system specification whose range is the set of events in $/Machine/Operations$. For any pair of events (e, f) where $e \in \text{ran}(E)$ and $f \in E[E^{-1}(e) + 1..size(E)]$, let $\text{grd}(e)$ and $\text{grd}(f)$ be the explicit guard of events e and f given in definitions 2. The Exclusiveness PO to prove is the following:

$$\begin{aligned}
& B_definitions \wedge \\
& / * \text{Sets and Properties from Seen components:} * / \\
& \text{and_join}(\text{all_sets}(/Machine/Sees/Machine/Sets/*)) \wedge \\
& \text{and_join}(/Machine/Sees/Machine/Properties/*) \wedge \\
& / * \text{Sets and Properties of the System} * / \\
& \text{and_join}(\text{all_sets}(/Machine/Sets/*)) \wedge \\
& /Machine/Properties/* \wedge \\
& / * \text{Invariant and Assertions from Seen components:} * / \\
& \text{and_join}(/Machine/Sees/Machine/Invariant/*) \wedge \\
& \text{and_join}(/Machine/Sees/Machine/Assertions/*) \wedge \\
& / * \text{Invariant and Assertions of the System} * / \\
& /Machine/Invariant/* \\
& /Machine/Assertions/* \\
& \Rightarrow \\
& \neg(\text{grd}(e) \wedge \text{grd}(f))
\end{aligned}$$

To be meaningful, the DLF PO needs the FIS PO to be proved.

B.3 Proof Obligations for Refinements

B.3.1 Merge Refinement Correctness

For any event e of a system refinement in $/\text{Machine}/\text{Operations}/*$, let a_1, \dots, a_n be the set of events in $/\text{Machine}/\text{Abstraction}/\text{Machine}[1]/\text{Operations}$ where each a_i is referenced in the set $e/\text{Refines}/*$ and e denotes the context of event e . The Merge Refinement Correctness PO for each event e is:

$$\begin{aligned}
 & B_definitions \wedge \\
 & / * \text{Sets and Properties from Seen components:} * / \\
 & \text{and_join}(\text{all_sets}(/ \text{Machine}/\text{Sees}/\text{Machine}/\text{Sets}/*)) \wedge \\
 & \text{and_join}(/ \text{Machine}/\text{Sees}/\text{Machine}/\text{Properties}/*) \wedge \\
 & / * \text{Sets and Properties clause from all refinement components:} * / \\
 & \text{and_join}(\text{all_sets}(/ \text{Machine}/\text{Abstraction}/\text{Machine}/\text{Sets}/*)) \wedge \\
 & \text{and_join}(/ \text{Machine}/\text{Abstraction}/\text{Machine}/\text{Properties}/*) \wedge \\
 & / * \text{Sets and Properties of the System} * / \\
 & \text{and_join}(\text{all_sets}(/ \text{Machine}/\text{Sets}/*)) \wedge \\
 & / \text{Machine}/\text{Properties}/* \wedge \\
 & / * \text{Invariant and Assertions from Seen components:} * / \\
 & \text{and_join}(/ \text{Machine}/\text{Sees}/\text{Machine}/\text{Invariant}/*) \wedge \\
 & \text{and_join}(/ \text{Machine}/\text{Sees}/\text{Machine}/\text{Assertions}/*) \wedge \\
 & / * \text{Invariant and Assertion clauses from all refinement components:} * / \\
 & \text{and_join}(\text{all_sets}(/ \text{Machine}/\text{Abstraction}/\text{Machine}/\text{Invariant}/*)) \wedge \\
 & \text{and_join}(/ \text{Machine}/\text{Abstraction}/\text{Machine}/\text{Assertions}/*) \wedge \\
 & / * \text{Invariant and Assertions of the Refinement} * / \\
 & / \text{Machine}/\text{Invariant}/* \\
 & / \text{Machine}/\text{Assertions}/* \\
 & \Rightarrow \\
 & [e] \neg [\Box i \cdot (i \in 1..n \mid a_i)] \neg J
 \end{aligned}$$

where $\Box i \cdot (i \in 1..n \mid a_i)$ is a shorthand for

CHOICE a_1 OR \dots a_n END

B.3.2 Feasibility (FIS) Proof Obligation

For any event e of a system refinement in $/Machine/Operations/*$, let $grd(e)$ and $fis(e)$ be the explicit and implicit guards given in definitions 2 and 3. The Feasibility PO to prove for event e is the following:

```
B_definitions ∧  
/* Sets and Properties from Seen components: */  
and_join(all_sets(/Machine/Sees/Machine/Sets/*)) ∧  
and_join(/Machine/Sees/Machine/Properties/*) ∧  
/* Sets and Properties clause from all refinement components: */  
and_join(all_sets(/Machine/Abstraction/Machine/Sets/*)) ∧  
and_join(/Machine/Abstraction/Machine/Properties/*) ∧  
/* Sets and Properties of the System */  
and_join(all_sets(/Machine/Sets/*)) ∧  
/Machine/Properties/* ∧  
/* Invariant and Assertions from Seen components: */  
and_join(/Machine/Sees/Machine/Invariant/*) ∧  
and_join(/Machine/Sees/Machine/Assertions/*) ∧  
/* Invariant and Assertion clauses from all refinement components: */  
and_join(all_sets(/Machine/Abstraction/Machine/Invariant/*)) ∧  
and_join(/Machine/Abstraction/Machine/Assertions/*) ∧  
/* Invariant and Assertions of the Refinement */  
/Machine/Invariant/*  
/Machine/Assertions/*  
⇒  
(grd( $e$ ) ⇒ fis( $e$ ))
```

B.3.3 Relative Deadlock Freedom (DLF) Proof Obligation

Let A be the set of abstract events in $/Machine/Abstraction/Machine[1]/Operations/*$ and C be the set of refined events in $/Machine/Operations/*$.

The Relative Deadlock Freedom PO is the following:

$$\begin{aligned}
& B_definitions \wedge \\
& / * \text{Sets and Properties from Seen components:} * / \\
& and_join(all_sets(/Machine/Sees/Machine/Sets/*)) \wedge \\
& and_join(/Machine/Sees/Machine/Properties/*) \wedge \\
& / * \text{Sets and Properties clause from all refinement components:} * / \\
& and_join(all_sets(/Machine/Abstraction/Machine/Sets/*)) \wedge \\
& and_join(/Machine/Abstraction/Machine/Properties/*) \wedge \\
& / * \text{Sets and Properties of the System} * / \\
& and_join(all_sets(/Machine/Sets/*)) \wedge \\
& /Machine/Properties/* \wedge \\
& / * \text{Invariant and Assertions from Seen components:} * / \\
& and_join(/Machine/Sees/Machine/Invariant/*) \wedge \\
& and_join(/Machine/Sees/Machine/Assertions/*) \wedge \\
& / * \text{Invariant and Assertion clauses from all refinement components:} * / \\
& and_join(all_sets(/Machine/Abstraction/Machine/Invariant/*)) \wedge \\
& and_join(/Machine/Abstraction/Machine/Assertions/*) \wedge \\
& / * \text{Invariant and Assertions of the Refinement} * / \\
& /Machine/Invariant/* \\
& /Machine/Assertions/* \\
& \Rightarrow \\
& \forall e \cdot (e \in A \wedge \text{grd}(e) \Rightarrow \exists e' \cdot (e' \in C \wedge \text{grd}(e')))
\end{aligned}$$

To be meaningful, the DLF PO needs the FIS PO to be proved.

B.3.4 Skip Refinement Correctness Proof Obligation

For any new event h in $/Machine/Operations/*$, The Skip Refinement Correctness PO is the following:

$$\begin{aligned}
& B_definitions \wedge \\
& / * \text{Sets and Properties from Seen components:} * / \\
& and_join(all_sets(/Machine/Sees/Machine/Sets/*)) \wedge \\
& and_join(/Machine/Sees/Machine/Properties/*) \wedge \\
& / * \text{Sets and Properties clause from all refinement components:} * / \\
& and_join(all_sets(/Machine/Abstraction/Machine/Sets/*)) \wedge \\
& and_join(/Machine/Abstraction/Machine/Properties/*) \wedge \\
& / * \text{Sets and Properties of the System} * / \\
& and_join(all_sets(/Machine/Sets/*)) \wedge \\
& /Machine/Properties/* \wedge \\
& / * \text{Invariant and Assertions from Seen components:} * / \\
& and_join(/Machine/Sees/Machine/Invariant/*) \wedge \\
& and_join(/Machine/Sees/Machine/Assertions/*) \wedge \\
& / * \text{Invariant and Assertion clauses from all refinement components:} * / \\
& and_join(all_sets(/Machine/Abstraction/Machine/Invariant/*)) \wedge \\
& and_join(/Machine/Abstraction/Machine/Assertions/*) \wedge \\
& / * \text{Invariant and Assertions of the Refinement} * / \\
& /Machine/Invariant/* \\
& /Machine/Assertions/* \\
& \Rightarrow \\
& [h] \neg [skip] \neg J
\end{aligned}$$

B.3.5 Numeric Variant Proof Obligation

Let V be the expression in $/Machine/Variant/*$, the Numeric Variant PO is the following:

```
B_definitions ∧  
/* Sets and Properties from Seen components: */  
and_join(all_sets(/Machine/Sees/Machine/Sets/*)) ∧  
and_join(/Machine/Sees/Machine/Properties/*) ∧  
/* Sets and Properties clause from all refinement components: */  
and_join(all_sets(/Machine/Abstraction/Machine/Sets/*)) ∧  
and_join(/Machine/Abstraction/Machine/Properties/*) ∧  
/* Sets and Properties of the System */  
and_join(all_sets(/Machine/Sets/*)) ∧  
/Machine/Properties/* ∧  
/* Invariant and Assertions from Seen components: */  
and_join(/Machine/Sees/Machine/Invariant/*) ∧  
and_join(/Machine/Sees/Machine/Assertions/*) ∧  
/* Invariant and Assertion clauses from all refinement components: */  
and_join(all_sets(/Machine/Abstraction/Machine/Invariant/*)) ∧  
and_join(/Machine/Abstraction/Machine/Assertions/*) ∧  
/* Invariant and Assertions of the Refinement */  
/Machine/Invariant/*  
/Machine/Assertions/*  
⇒  
 $V \in \mathbb{N}$ 
```

B.3.6 Non Divergence Proof Obligation

Let V be the expression in $/Machine/Variant/*$, v a fresh variable non free in the antecedents of the PO and h a new event in $/Machine/Operations/*$. The Non Divergence PO is the following:

```

B_definitions ∧
/* Sets and Properties from Seen components: */
and_join(all_sets(/Machine/Sees/Machine/Sets/*)) ∧
and_join(/Machine/Sees/Machine/Properties/*) ∧
/* Sets and Properties clause from all refinement components: */
and_join(all_sets(/Machine/Abstraction/Machine/Sets/*)) ∧
and_join(/Machine/Abstraction/Machine/Properties/*) ∧
/* Sets and Properties of the System */
and_join(all_sets(/Machine/Sets/*)) ∧
/Machine/Properties/* ∧
/* Invariant and Assertions from Seen components: */
and_join(/Machine/Sees/Machine/Invariant/*) ∧
and_join(/Machine/Sees/Machine/Assertions/*) ∧
/* Invariant and Assertion clauses from all refinement components: */
and_join(all_sets(/Machine/Abstraction/Machine/Invariant/*)) ∧
and_join(/Machine/Abstraction/Machine/Assertions/*) ∧
/* Invariant and Assertions of the Refinement */
/Machine/Invariant/*
/Machine/Assertions/*
⇒
[v := V] [h] V < v

```

B.3.7 Exclusiveness (EXC) Proof Obligation

Case Split of Events

For any abstract event a in `/Machine/Abstraction/Machine[1]/Operations/*` let $E(a)$ be an injective sequence in the set of events whose range is the following set:

$$\{e \mid e \in \text{/Machine/Operations/*} \wedge e \text{ refines } a\}$$

For any pair of events (e, f) , where $e \in \text{ran}(E(a))$ and $f \in E(a)[E(a)^{-1}(e) + 1..\text{size}(E(a))]$ the Exclusiveness PO for the case of split of events is:

```

B_definitions ∧
/* Sets and Properties from Seen components: */
and_join(all_sets(/Machine/Sees/Machine/Sets/*)) ∧
and_join(/Machine/Sees/Machine/Properties/*) ∧
/* Sets and Properties clause from all refinement components: */
and_join(all_sets(/Machine/Abstraction/Machine/Sets/*)) ∧
and_join(/Machine/Abstraction/Machine/Properties/*) ∧
/* Sets and Properties of the System */
and_join(all_sets(/Machine/Sets/*)) ∧
/Machine/Properties/* ∧
/* Invariant and Assertions from Seen components: */
and_join(/Machine/Sees/Machine/Invariant/*) ∧
and_join(/Machine/Sees/Machine/Assertions/*) ∧
/* Invariant and Assertion clauses from all refinement components: */
and_join(all_sets(/Machine/Abstraction/Machine/Invariant/*)) ∧
and_join(/Machine/Abstraction/Machine/Assertions/*) ∧
/* Invariant and Assertions of the Refinement */
/Machine/Invariant/*
/Machine/Assertions/*
⇒
¬(grd( $e$ ) ∧ grd( $f$ ))

```

Case New Vs Refined Events

Let H be an injective sequence of events whose range is the set of new events:

$$\{e \mid e \in \text{/Machine/Operations/*} \wedge e \text{ is new}\}$$

For any pair of events (h, e) where $h \in \text{ran}(H)$ and e is an event in $\text{/Machine/Operations/*}$ satisfying $e \notin \text{ran}(H)$, the Exclusiveness PO for the case of new vs refined events is:

```

B_definitions ∧
/* Sets and Properties from Seen components: */
and_join(all_sets(/Machine/Sees/Machine/Sets/*)) ∧
and_join(/Machine/Sees/Machine/Properties/*) ∧
/* Sets and Properties clause from all refinement components: */
and_join(all_sets(/Machine/Abstraction/Machine/Sets/*)) ∧
and_join(/Machine/Abstraction/Machine/Properties/*) ∧
/* Sets and Properties of the System */
and_join(all_sets(/Machine/Sets/*)) ∧
/Machine/Properties/* ∧
/* Invariant and Assertions from Seen components: */
and_join(/Machine/Sees/Machine/Invariant/*) ∧
and_join(/Machine/Sees/Machine/Assertions/*) ∧
/* Invariant and Assertion clauses from all refinement components: */
and_join(all_sets(/Machine/Abstraction/Machine/Invariant/*)) ∧
and_join(/Machine/Abstraction/Machine/Assertions/*) ∧
/* Invariant and Assertions of the Refinement */
/Machine/Invariant/*
/Machine/Assertions/*
⇒
¬(grd( $h$ ) ∧ grd( $e$ ))

```

Case New Events

Let H be an injective sequence of events whose range is the set of new events:

$$\{e \mid e \in \text{Machine/Operations}/\star \wedge e \text{ is new}\}$$

and $(\text{ran}(H), \prec)$ a strict total order. For any pair of events (e, f) , where $e \in \text{ran}(H)$ and $f \in H[H^{-1}(e) + 1..\text{size}(H)]$ the Exclusiveness PO for the case of new events is:

```

B_definitions  $\wedge$ 
/* Sets and Properties from Seen components: */
and_join(all_sets(/Machine/Sees/Machine/Sets/*))  $\wedge$ 
and_join(/Machine/Sees/Machine/Properties/*)  $\wedge$ 
/* Sets and Properties clause from all refinement components: */
and_join(all_sets(/Machine/Abstraction/Machine/Sets/*))  $\wedge$ 
and_join(/Machine/Abstraction/Machine/Properties/*)  $\wedge$ 
/* Sets and Properties of the System */
and_join(all_sets(/Machine/Sets/*))  $\wedge$ 
/Machine/Properties/*  $\wedge$ 
/* Invariant and Assertions from Seen components: */
and_join(/Machine/Sees/Machine/Invariant/*)  $\wedge$ 
and_join(/Machine/Sees/Machine/Assertions/*)  $\wedge$ 
/* Invariant and Assertion clauses from all refinement components: */
and_join(all_sets(/Machine/Abstraction/Machine/Invariant/*))  $\wedge$ 
and_join(/Machine/Abstraction/Machine/Assertions/*)  $\wedge$ 
/* Invariant and Assertions of the Refinement */
/Machine/Invariant/*
/Machine/Assertions/*
 $\Rightarrow$ 
 $\neg(\text{grd}(e) \wedge \text{grd}(f))$ 

```

B.3.8 Coverage Proof Obligation

Let H be the set of new events in a refinement:

$$\{e \mid e \in \text{/Machine/Operations/*} \wedge e \text{ is new}\}$$

and A the set of abstract events:

$$\{e \mid e \in \text{/Machine/Abstraction/Machine[1]/Operations/*}\}$$

. Let $R(a)$ be the set of events refining a :

$$\{e \mid e \in \text{/Machine/Operations/*} \wedge e \text{ refines } a\}$$

The Coverage PO is the following:

```

B_definitions  $\wedge$ 
/* Sets and Properties from Seen components: */
and_join(all_sets(/Machine/Sees/Machine/Sets/*))  $\wedge$ 
and_join(/Machine/Sees/Machine/Properties/*)  $\wedge$ 
/* Sets and Properties clause from all refinement components: */
and_join(all_sets(/Machine/Abstraction/Machine/Sets/*))  $\wedge$ 
and_join(/Machine/Abstraction/Machine/Properties/*)  $\wedge$ 
/* Sets and Properties of the System */
and_join(all_sets(/Machine/Sets/*))  $\wedge$ 
/Machine/Properties/*  $\wedge$ 
/* Invariant and Assertions from Seen components: */
and_join(/Machine/Sees/Machine/Invariant/*)  $\wedge$ 
and_join(/Machine/Sees/Machine/Assertions/*)  $\wedge$ 
/* Invariant and Assertion clauses from all refinement components: */
and_join(all_sets(/Machine/Abstraction/Machine/Invariant/*))  $\wedge$ 
and_join(/Machine/Abstraction/Machine/Assertions/*)  $\wedge$ 
/* Invariant and Assertions of the Refinement */
/Machine/Invariant/*
/Machine/Assertions/*
 $\Rightarrow$ 
 $\forall e \cdot (e \in A \wedge \text{grd}(e) \Rightarrow \exists e' \cdot (e' \in R(e) \cup H \wedge \text{grd}(e')))$ 

```