Specification and Proof of Liveness Properties in B Event Systems

Olfa Mosbahi, Jacques Jaray

SPECIFICATION AND PROOF OF LIVENESS PROPERTIES IN B EVENT SYSTEMS

Olfa MOSBAHI, Jacques JARAY
LORIA, INRIA Lorraine, Nancy University, France
mosbahi@loria.fr, jaray@loria.fr

Keywords: Automated systems, Event B method, Liveness properties, Language TLA$^+$, Verification.

Abstract: In this paper, we give a framework for defining an extension to the event B method. The event B method allows us to state only invariance properties, but in some applications such as automated or distributed systems, fairness and eventuality properties must also be considered. We first extend the expressiveness of the event B method to deal with the specification of these properties. Then, we give a semantics of this extended syntax over traces, in the same spirit as the temporal logic of actions TLA does. Finally, we give verification rules of these properties. We denote by temporal B model, the B model extended with liveness properties. We illustrate our method on a case study related to automated system.

1 INTRODUCTION

The paper deals with liveness properties of automated systems. In such systems, we distinguish a software part : the *controller* and an *operative part* formed by a physical device and its environment.

The event B method provides us with techniques and tools for specifying, refining, verifying invariant properties and implementing systems. B is not well suited to deal with liveness properties. We define an extension of B in order to capture liveness properties. We describe the syntax of the extension and define the semantics in terms of traces in the same spirit of the language TLA$^+$. We also give the verification rules of these properties.

Several related works concern B extensions for capturing and proving liveness temporal properties. J-R. Abrial and L. Mussat in (Abrial and Mussat, 1998) proposed an extension consisting in a dynamic invariant clause containing linear temporal logic formulae (LTL). In order to allow verification by theorem proving, the user has to provide the model with decreasing functions, a variant and a loop invariant. Such items are necessary for the prover but are indeed not part of the specification. Furthermore, finding variant and loop invariant is not an easy task. D.Bert and R.Barradas (Barradas and Bert, 2002) have proposed a method for the specification and proof of liveness properties in B event systems under fairness assumptions. They give proof obligations in order to prove basic progress properties in B event systems under two types of assumptions : minimal progress and weak fairness. They define proof obligations in terms of *weakest preconditions*, which allow us to prove basic liveness properties as usual B proof obligations. They suggest the use of UNITY "Leadsto" operator to specify more general liveness properties. The semantics of these properties is defined in terms of *weakest preconditions* but in our work, we give a semantics in terms of *traces*.

The paper is organized as follows : section 2 presents an overview of the event B method, section 3 presents an overview of the language TLA$^+$, section 4 gives a description of our proposal using a case study : we give the syntax, the semantics of liveness properties and then the verification rules necessary to prove these properties under fairness assumptions. Finally, section 5 ends with a conclusion and future work.

2 Overview of the event B method

The event B method (Abrial, 2003) is based on the B notation (Abrial, 1996). It extends the methodological scope of basic concepts such as set-theoretical notations and generalized substitutions in order to take into account the idea of *formal models*. Roughly speaking, a formal model is characterized by a (finite) list x of *state variables* possibly modified by a (finite) list of *events*; an invariant $I(x)$ states some properties that must always be satisfied by the variables x and maintained by the activation of the events. Generalized substitutions provide a way to express the transformations of the values of the state variables of a formal model. An event consists of two parts : a *guard* (denoted *grd*) and an action. A guard is a predicate built from the state variables, and an *action* is a generalized substitution

(denoted *GS*).

An event can take one of the forms shown in the table 1. Let *BA(x,x')* be the before-after predicate associated with each event shape. This predicate describes the event as a logical predicate expressing the relationship linking the values of the state variables just before (x) and just after (x') the event "execution". In the table below, x denotes a vector built on the set of state variables of the model. In the general substitution $x : p(x_0, x)$, x denotes the *new value* of the vector, whereas x_0 denotes its *old value* and t represents a vector of distinct local variables.

Table 1: Event forms.

Event	*Before-after Predicate BA(x, x')*	*Guard*
$BEGIN$ $x : P(x_0, x)$ END;	$P(x, x')$	TRUE
$SELECT\ G(x)$ $THEN\ x : Q(x_0, x)$ END;	$G(x) \wedge Q(x, x')$	$G(x)$
$ANY\ t$ $WHERE\ G(t, x)$ $THEN\ x : R(x_0, x, t)$ END;	$\exists t.(G(t, x)$ $\wedge R(x, x', t))$	$\exists t.G(t, x)$

Proof obligations are associated to events and state that the invariant condition $I(x)$ is preserved. We next give the general rule to be proved. It follows immediately from the very definition of the before-after predicate, $BA(x, x')$ of each event :

$$\boxed{I(x) \wedge BA(x, x') \Rightarrow I(x')}$$

The B model has the following form :

MODEL ⟨*name*⟩
SETS ⟨*sets*⟩
CONSTANTS ⟨*constants*⟩
PROPERTIES ⟨*properties of sets and constants*⟩
VARIABLES ⟨*variables x*⟩
INVARIANT ⟨*invariants* $I(x)$⟩
ASSERTIONS ⟨$A(x)$⟩
INITIALISATION ⟨*initialization of variables*⟩
EVENTS ⟨*events*⟩
END

An abstract B model has a name; the clause **SETS** contains definitions of sets; the clause **CONSTANTS** allows us to introduce information related to the mathematical structure. The clause **PROPERTIES** contains the effective definitions of constants. The clause **ASSERTIONS** contains the list of theorems to be discharged by the proof engine. The clause **VARIABLES** contains a (finite) list of state variables possibly modified by a (finite) list of events; the clause **INVARIANT** states some properties that must always be satisfied by the variables and maintained by the activation of the events. The clause **EVENTS** contains all the system events which preserve the set of invariants.

2.1 Refinement

Construction by refinement (Back and v. Wright, 1998; Back and K-Sere, 1989) is a technique suitable for the development of complex systems. The refinement of a formal model allows us to enrich a model in a step by step approach. It is used to transform an abstract model into a more concrete version by modifying the state description (Spivey, 1988). This is essentially done by extending the list of state variables, refining each abstract event into a corresponding concrete version, and adding new events.

The essence of the refinement relationship is that it preserves already proved system properties. The invariant of an abstract model plays a central role for deriving safety properties and our method focuses on the incremental discovery of the invariant; the goal is to obtain a formal statement of properties through the final invariant of the last refined abstract model. Atelier B (ClearSy, 2002), the toolkit supporting the B method, generates the proof obligations associated with a model or a refinement. It also provides automatic and iterative proof procedures to discharge these proof obligations.

2.2 Example : A parcel sorting device

In this section, we present an example of reactive system : a parcel sorting device (Jaray and A.Mahjoub, 1996) which will be taken to illustrate our proposed approach. We just give the abstract model of the system and not the refinement steps. The problem is to sort parcels into baskets according to an address written on the parcel. In order to achieve such a sorting function we are provided with a device made of a feeder connected to the root of a binary tree made of switches and pipes as shown in the figure 1. The switches are the nodes of the tree, pipes are the edges and baskets are the leaves. A parcel, thanks to gravity, can slide down through switches and pipes to reach a basket.

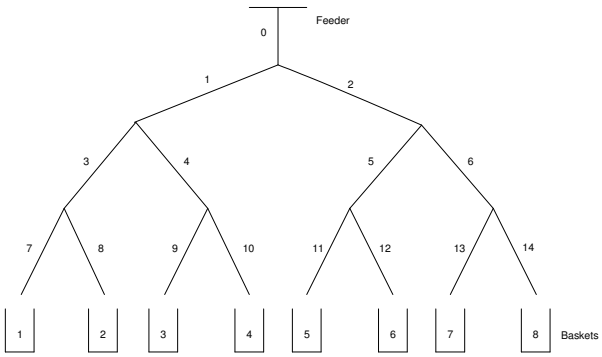

Figure 1: Router

A switch is connected to an entry pipe and two exit pipes, a parcel crossing the switch is directed to an exit pipe depending on the switch position. The feeder releases one parcel at a time in the router, the feeder contains a device to read the address of the parcel to be released. When released, a parcel enters a first switch (the root of the binary tree) and

slides down the router to reach a basket. The controller can activate the feeder and change the switches position. For safety reasons, it is required that switch change should not occur when a parcel is crossing it. In order to check this condition, sensors are placed at the entry and the exits of each switch.

We consider a simplified version of the system with only safety properties to illustrate a specification with the event B method and we will deal in the following with liveness properties (eventuality and fairness) to explain our approach.

Abstract model of the system

the abstract model of the system is given in the figure 2.

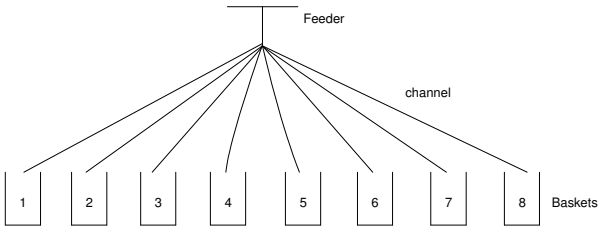

Figure 2: Router

The sorting device. The sorting device consists of a feeder and a sorting layout. The feeder has two functions: selection of the next parcel to introduce into the sorting layout and opening the gate (releasing a parcel in the sorting layout). We introduce the events *select* and *release* to capture the two functions. In order to produce the abstract model of the sorting layout, we have to notice that a given state of the switches forms a *channel* linking the entrance to a unique sorting basket. A basket is an element of a set named *Baskets*. Channels and sorting baskets are in a one to one correspondence. Therefore, the abstract model of the sorting device can be reduced to a single variable *channel* taking the value of the sorting basket it leads to, namely a value in the set *Baskets*. The *channel* value is changed by the event *set_channel*. It is worth noticing that the abstraction forces a "sequential functioning" of the sorting device, i.e. the value of the channel remains unchanged as long as the parcel released in the sorting device has not reached a sorting basket.
Parcels. Parcels, as part of the environment, are represented as elements of a set we name *PARCELS*. We use a total function (*adr*) from *PARCELS* to the interval *Baskets* to refer to the parcels address. We give the status "*arrived*" to the parcel which has reached a sorting basket. The variable (*arrived*) is a function from *PARCELS* to *Baskets*. The goal of the sorting system is to decrease the set of the parcels to sort. The variable *sorted* represents the set of sorted parcels. The remaining parcels are defined by the expression *PARCELS* - *sorted* named *UNSORTED*. As *pe* is undefined when the sorting device is empty, we have introduced a set *PPARCELS* of which *PARCELS* is a proper subset; *pe* is an element of *PPARCELS* and assignment of any value in *PPARCELS* - *PARCELS* stands for "undefined". The expression *PPARCELS* - *PARCELS* will be referred as *NOPARCELS*. The selection of a parcel is an event which may be activated once the device is free and the variable *pe* is undefined, which means it does not exist a parcel being sorted.
Moving parcels. In our abstraction a parcel takes no time to travel from the feeder to a basket. A parcel arrives in the basket to which the channel leads up. When the event *cross_parcel* occurs, the current parcel sorting is finished and then, of course, the current parcel becomes undefined.
The Controller. The controller has to ensure right parcel routing. Two events are added for the controller : *Set_channel* and *Release*. The event *Set_channel* assigns to channel the value of *adr(pe)*. The event *Release* changes the state of the sorting device from *free* to *busy*. The model of the automated system is presented in Figure 3.
Simulation of the B model with ProB. We have used ProB (Leuschel and Butler, 2003), witch is an simulator and model checker for the (B/Event-B) Method. It allows fully automatic animation of many B specifications, and can be used to systematically check a specification for errors. ProB's animation facilities allow users to gain confidence in their specifications, and unlike the animator provided by the B-Toolkit, the user does not have to guess the right values for the operation arguments or choice variables. ProB contains a model checker and a constraint-based checker, both of which can be used to detect various errors in B specifications. ProB enables users to uncover errors that are not easily discovered by existing tools. Figure 4, shows the simulation of the abstract model of the system.
Verification of the B model . All generated proof obligations are verified with the B click_n_Prove tool.
Requirement of liveness properties. In our example, we need to consider the dynamics of the system. Our model must take into account the following properties

1. Every parcel introduced in the entry eventually reaches one of the baskets, this property is described with : $\forall p.(p \in UNSORTED \Rightarrow \Diamond arrived(p) \in Baskets)$
2. Every parcel introduced in the entry must reach the basket corresponding to its destination address, this property is described with : $\forall p.(p \in UNSORTED) \rightsquigarrow arrived(p) = adr(p))$
3. Weak fairness conditions on the events is assumed : $WF(select\_parcel) \wedge WF(cross\_parcel) \wedge WF(set\_channel) \wedge WF(release)$

These properties can not be specified in the clause INVARIANT. We need to extend the expressivity of event B to take into account such properties.

3 Overview of the language TLA$^+$

TLA$^+$ is a language intended for the high level specification of reactive, distributed, and in particular asynchronous systems. It combines the linear-time temporal logic of actions TLA (Lamport, 1994), and mathematical set theory. The language has a mechanism for structuring in the form of modules, either by extension, or by instance. The semantics of TLA is based on behaviors of state variables. It can be viewed as a logic built in an incremental way in three stages :

1. predicates whose semantics is based on states.

MODEL **Parcel_Sorting**
SETS *PPARCELS* ; *SortingState* = {*free* , *busy* }
CONSTANTS *PARCELS, adr, Baskets*
PROPERTIES
$PARCELS \subset PPARCELS \wedge PARCELS \neq \varnothing \wedge$
$Baskets \neq \varnothing \wedge adr \in PARCELS \rightarrow Baskets$
VARIABLES
arrived, channel, sorting, pe, sorted, ready_to_sort
INVARIANT
$arrived \in PARCELS \nrightarrow Baskets \wedge channel \in Baskets \wedge$
$pe \in PPARCELS \wedge sorting \in SortingState \wedge$
$ready\_to\_sort \in BOOL \wedge sorted \subseteq PARCELS \wedge$
$(sorting = busy \Rightarrow channel = adr(pe)) \wedge$
$(sorting = busy \Rightarrow \neg\ ready\_to\_sort) \wedge$
$(ready\_to\_sort \Rightarrow channel = adr(pe)) \wedge$
$(ready\_to\_sort \Rightarrow pe \in PARCELS) \wedge$
$\forall p.(p \in PARCELS \wedge p \in dom(arrived) \Rightarrow arrived(p) = adr(p))$
DEFINITIONS
UNSORTED == PARCELS - sorted ;
NOPARCELS == PPARCELS - PARCELS
INITIALISATION
arrived := {} || *channel* :∈ *Baskets* || *sorting* := *free* ||
pe :∈ *NOPARCELS* || *sorted* := {} ||
ready_to_sort := *FALSE*
EVENTS
select_parcel = **ANY** *p* **Where** $p \in UNSORTED \wedge$
$pe \in NOPARCELS \wedge sorting = free$
THEN *pe* := *p*
END;

set_channel = **SELECT** $sorting = free \wedge pe \in PARCELS$
$\wedge \neg\ ready\_to\_sort$
THEN *channel* := *adr(pe)* ||
ready_to_sort := *TRUE*
END;

release = **SELECT** $sorting = free \wedge pe \in PARCELS \wedge$
ready_to_sort
THEN *sorting* := *busy* ||
ready_to_sort := *FALSE*
END;

cross_parcel = **SELECT** *sorting* = *busy*
THEN *arrived(pe)* := *channel* ||
sorted := *sorted* ∪ { *pe*} ||
pe :∈ *NOPARCELS* || *sorting* := *free*
END
END

Figure 3: Abstract model of the sorting device

2. actions whose semantics is based on pairs of states.
3. temporal formulas of actions whose semantics is based on state behaviors of variables.

A TLA specification of a system denoted by $Spec(S)$ looks like : $Init \wedge \Box[Next]_x \wedge L$ where :

1. *Init* is the predicate which specifies initial states,

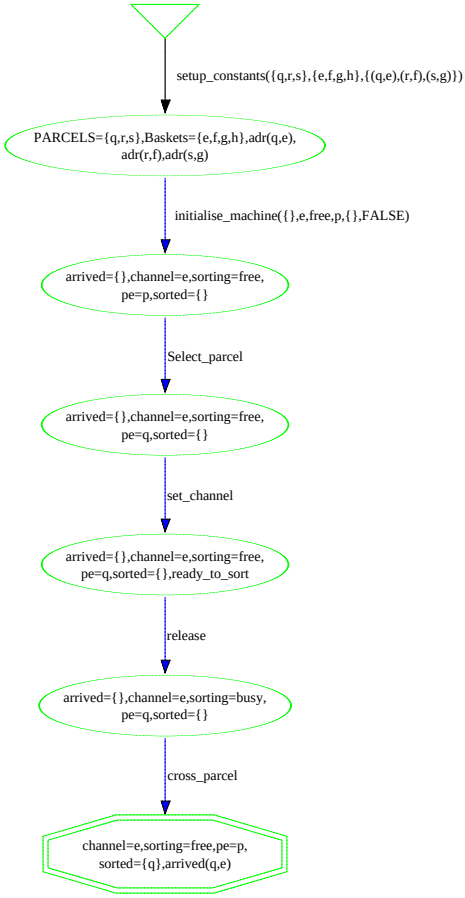

Figure 4: Model checking of the router abstract model

2. x is the list of all state variables and $\Box[Next]_x$ means that either two consecutive states are equal on $x, x' = x$ (stuttering), or *Next* is an action (a relation) that describes the next-state relation, usually written as a disjunction of more elementary actions,

3. L is a fairness assumption (strong or weak) on actions. $WF_{unprimed\_var(S)}(S)$ defines the condition of weak fairness over the system S and $SF_{unprimed\_var(S)}(S)$ defines the condition of strong fairness over the system S, where $primed\_var(S)$ are primed occurrences of the system variables x and as is conventional, a primed occurrence v' of a state variable v denotes the value of v in the state following the transition described by *Next*. $unprimed\_var(S)$ are unprimed occurrences of the system variables x and an unprimed occurrence denotes the value of a variable v in the state before the transition.

In the sequel we will focus on the extension of the event B method with liveness properties, their syntax, their semantics and verification rules.

4 Assigning temporal meaning to B models

This section defines an extension to event B in order to deal with liveness properties. The most important construction we need is the *"leads to"* eventuality operator as in TLA and Unity which expresses requirements on behaviors, i.e. sequence of states. In order to assess eventuality properties we must state assumptions on the fair occurrence of events. Such assumptions are stated using the TLA operators WF and SF. $WF(e)$ assumes that the event e is weakly fair, i.e. the event e occurs infinitely often provided that it is eventually always enabled. $SF(e)$ assumes that the event e is strongly fair, i.e. the event e occurs infinitely often provided that it is infinitely often enabled.

We indeed integrate some pieces of the language TLA$^+$ into the event B models and we deal with proof obligations of "temporal" B models.

In the following, we start with the syntax of the extension, then we give a semantics and verification rules of liveness properties over traces as it is done in TLA$^+$. We suggest the use of TLA$^+$ operators because the two methods are very close with respect to their foundations.

4.1 Syntax of the extension

In order to establish liveness properties we must assume some progress conditions on the system. As long as we have to verify that an event system satisfies safety properties, it is sufficient to refer to a pair of states (before and after states of a triggering event). But in order to prove temporal properties we need to introduce *behaviors* (sequences of states) starting from the initial state and where two consecutive states s_i and s_{i+1} are such that some event enabled in s_i and leads to the state s_{i+1}.

Before defining the syntax of formulae which extends B expressivity, we start with some definitions.

State and rigid variables. The state of a system is composed of a denumerable set of flexible or state variables (V). Let (X) be a denumerable set of rigid variables. These variables are not modified by program transitions and hence keep the initially chosen value during a program run (logical constant). A state is a valuation of flexible variables.

Terms and States. A term t is defined recursively as follows :

$t ::= c \mid x \mid f(t_1,...,t_n)$ where c is a constant, x is a variable ($x \in [V \cup X]$) , $t_1,...,t_n$ are terms and f is a function symbol with arity n.

Atomic propositions. An atomic proposition ap is a formula of the form :

$ap ::= p(t_1,...,t_n)$ where p is a predicate symbol with arity n and $t_1,...,t_n$ are terms.

State predicates. A state predicate sp is a formula defined by the following grammar

$sp ::= ap \mid \neg sp \mid sp \vee sp \mid sp \wedge sp \mid sp \Rightarrow sp \mid sp \Leftrightarrow sp \mid \exists x\ sp \mid \forall x\ sp$.

In our extension, we introduce transition and liveness formulae.

Transition formulae. A transition formula describes state transitions. A transition formula ac is a formula of the form :

$ac ::= GS(e) \mid [e]sp \mid \langle e \rangle sp$ where e is an event, $GS(e)$ is its generalized substitution and sp is a state predicate.

Safety properties. Safety properties are formulae of the form

$F ::= \Box sp \mid \Box(sp \Rightarrow \Box sp)$, where sp is a state predicate.

Liveness properties. Liveness properties (fairness and eventuality) are formulae defined as follows:

- Eventuality properties are expressed with formulae of the form :

  $F \rightsquigarrow G$ (F *leads to* G) defined as $\Box(F \Rightarrow \Diamond G)$ and means that every F will be followed by G, where F and G are formulae of the form : $F ::= sp \mid \Diamond F \mid \Box F \mid WF(e) \mid SF(e)$.

  Where sp is a state predicate, $WF(e)$ and $SF(e)$ are respectively the weak and strong fairness of the event e.

  These properties are added in the clause EVENTUALITY.

- Fairness properties are expressed with formulae of the form :

  - $WF(e)$ defined as $\Diamond\Box grd(e) \Rightarrow \Box\Diamond GS(e)$. It is the weak fairness condition of an event e and it means that the event e occurs infinitely often provided that it is eventually always enabled,
  - $SF(e)$ defined as $\Box\Diamond grd(e) \Rightarrow \Box\Diamond GS(e)$. It is the strong fairness condition of an event e and it means that the event e occurs infinitely often provided that it is infinitely often enabled,

  Where :

  - e is a B event,
  - $grd(e)$ is the guard of this event e (state predicate),
  - $GS(e)$ is the generalized substitution of the event e. It is a transition formula containing both primed and unprimed occurrences of states variables, such as a before-after predicate.

  These properties are added in the clause FAIRNESS.

4.2 Semantics of the extension

In our extension, we deal with properties over state sequences (fairness and eventuality properties). This is why we need a semantics over sequence of states and have to explain how we can view events as a relation over primed and unprimed variables and we will use this point to find the extension of the event B method. A system S is modelled as a set of possible events triggering actions, when guards are true. An event e as it was shown in the table 1, is defined by a guard denoted $grd(e)$ (condition for triggering or enabledness condition) and by a relation over a set of flexible variables (V) denoted $GS(e)$ (relation stating the transformation of variables). According a TLA$^+$ module, we consider three kinds of properties :

- *State properties* which denote properties on states of the system S and are interpreted over states. These properties are state predicates,
- *Relational properties* which denote relations on S between pairs of states, which we call transition formulae,

- *Temporal Properties* state properties over traces and use state properties, relational properties and temporal operators ($\Box$, $\Diamond$, $\rightsquigarrow$, ...), which we call liveness properties.

Properties are interpreted over traces (sequences of states). We introduce notations for characterizing systems :

- V is the set of state variables of the system S, v is a state variable; x is the current value of v and x' is the next value of v. $Primed\_Var(S) = \{x' | v \in V\}$ and $Unprimed\_Var(S) = \{x | v \in V\}$.
- $Init(S)$ specifies the initial values of state variables of the system S.
- $Events(S)$ specifies the set of possible events of S; it means that we list the possible events defined in the figure 1. An event e is defined as follows :

$$e \triangleq grd(e) \text{ then } GS(e)$$

- $Next(S)$ is a formula over primed and unprimed variables of S corresponding to the relation over $States(S)$, namely $\rightarrow$, where $States(S)$ is the set of states of the system S. *Next* has the following form :

  $Next(S) \triangleq R(e_1)(x,x') \vee .... \vee R(e_n)(x,x')$
  where $R(e_i)(x,x')$ is a relation corresponding to one of the event forms presented in the table 1.

  $R(e_i)(x,x') \triangleq P(x,x') \vee (G(x) \wedge P(x,x')) \vee (\exists t.(G(t,x) \wedge P(x,x',t)))$
- $\rightarrow$ is a relation over States(S) simulating the execution of the system S.
- $Invariants(S)$ is a set of properties over States(S) invariant for S. φ is in $Invariants(S)$, if
  1. $Init(S) \Rightarrow \varphi$
  2. $\forall s_0, s_i \in States(S) \ : \ s_0, \xi \models Init(S) \wedge (s_0 \rightarrow^* s_i) \Rightarrow s_i, \xi \models \varphi$
- $Traces(S)$ is the set of traces (state sequences) generated from Init(S) using $\rightarrow$. A trace is denoted by $\sigma = s_0 s_1 .....s_i, ...$ and satisfying the following constraints :
  1. $s_0, \xi \models Init(S)$ (the initial state s_0 satisfies the initial condition),
  2. $\forall i \in \mathbb{N} : (s_i \rightarrow s_{i+1}) \vee (s_i = s_{i+1})$ any two successive states (s_i, s_{i+1}) either satisfy the before-after predicate $BA_e(x,x')$ for some event e and some variables x, or agree on the values of all system variables (called stuttering steps)

Let $\sigma \in Traces(S)$, A property φ over states sequence of the system S is a state property, a relational property or a temporal property; the semantics over traces unified semantics over states and pairs of states as follows :

1. a state property φ is a trace property as follows :
   $\sigma, \xi \models \varphi$, if $s_0, \xi \models \varphi$.
2. a relational property φ is also a trace property by extending the semantics over pairs of states into a semantics over traces as follows : $\sigma, \xi \models \varphi$, if $(s_0, s_1), \xi \models \varphi$.

Temporal properties contains state properties, relational properties and temporal combination of these properties. Our extension is the same one than TLA$^+$ and a system S is specified by the following temporal expression :

$$\begin{aligned} \text{Specification}(S) \triangleq \wedge\ & \text{Init}(S) \\ & \wedge \Box[Next(S)]_{<unprimed\_var(S)>} \\ & \wedge WF_{unprimed\_var(S)}(S) \\ & \wedge SF_{unprimed\_var(S)}(S) \end{aligned}$$

Where:
Init(S) states initial conditions,
$\Box[Next(S)]_{<unprimed\_var(S)>}$ states how traces are built,
$WF_{unprimed\_var(S)}(S)$ defines the condition of weak fairness over the system S and
$SF_{unprimed\_var(S)}(S)$ defines the condition of strong fairness over the system S.

$WF_{unprimed\_var(S)}(S)$ and $SF_{unprimed_var(S)}(S)$are defined as follows :

$WF_{unprimed\_var(S)}(S) \triangleq \bigwedge_{E \in WF\_Events(S)} WF_{unprimed\_var(S)}(E)$
and
$SF_{unprimed\_var(S)}(S) \triangleq \bigwedge_{E \in SF\_Events(S)} SF_{unprimed\_var(S)}(E)$

Where $WF\_Events(S)$ is the set of weakly fair events and $SF\_Events(S)$ is the set of strongly fair events. $WF_{unprimed\_var(S)}(E)$ is the weak fairness associated to the event E and $SF_{unprimed\_var(S)}(E)$ is the strong fairness associated to the event E. Each event is associated with a fairness condition which will be a weak or strong or undefined.

In the event B, $BA_e(x,x')$ is the before-after predicate for an event; this is a first-order formula built from the constants declared for the system specification, as well as primed and unprimed occurrences of the system variables V. The before-after predicate $BA_e(x,x')$ in B method is interpreted by the formula *Next* in TLA$^+$. In TLA$^+$, or an action e, the enabled condition $Enabled(e)$ is defined by existentially quantifying over the primed occurrences of the state variables; thus, the state predicate $Enabled(e)$ is true of those states that have a successor state related by an occurrence of the event e.
$Enabled(e) \triangleq \exists x' : BA_e(x,x')$.
The guard $grd(e)$ in B is interpreted by the condition $Enabled(e)$ in TLA$^+$.

We can summarize the semantics of temporal B notations over traces by the following equivalences in TLA$^+$:

$$\begin{array}{ccc} grd(e) & \triangleq & Enabled(e) \\ BA_e & \triangleq & Next \end{array}$$

Interpretation of formulae

Let $\sigma = s_0 s_1 ...$ be a behavior, i.e. a sequence of states and ξ a valuation of the rigid variables of S. Let $[|x|]^{\xi}_{s_i}$ be the value of the variable x in the state s_i, $[|f(t_1,...,t_n)|]^{\xi}_{s_i}$ gives the semantics of the term $f(t_1,...,t_n)$ in the state s_i.

$$[|x|]^{\xi}_{s_i} = \begin{cases} \xi(x) & where \ \ x \in X; \\ s_i(x) & where \ \ x \in V. \end{cases}$$

$$[|f(t_1,...,t_n)|]^{\xi}_{s_i} = [|f|]([|t_1|]^{\xi}_{s_i}, ..., [|t_n|]^{\xi}_{s_i})$$

In the following, we denote by $s_i, \xi \models sp$ the satisfaction of the state predicate sp in the state s_i of a transition

system and by $\sigma,\xi \models F$ the satisfaction of the temporal formula F over a trace $\sigma \in Traces(S)$.

Proposition formulae

$s_i,\xi \models ap$ iff ap holds in the state s_i

State formulae

$s_i,\xi \models sp$ iff sp holds in the state s_i

Boolean formulae

$s_i,\xi \models \neg sp$ iff $s_i,\xi \models sp$ is false
$s_i,\xi \models sp_1 \wedge sp_2$ iff $s_i,\xi \models sp_1$ and $s_i,\xi \models sp_2$
$s_i,\xi \models sp_1 \vee sp_2$ iff $s_i,\xi \models sp_1$ or $s_i,\xi \models sp_2$
$s_i,\xi \models sp_1 \Rightarrow sp_2$ iff $s_i,\xi \models \neg sp_1$ or $(s_i,\xi \models sp_1$ and $s_i,\xi \models sp_2)$
$s_i,\xi \models sp_1 \Leftrightarrow sp_2$ iff $s_i,\xi \models sp_1 \Rightarrow sp_2$ and $s_i,\xi \models sp_2 \Rightarrow sp_1$
$s_i,\xi \models (\exists x)\ sp$ iff $(\exists x) \in V :\ s_i,\xi \models sp$
$s_i,\xi \models (\forall x)\ sp$ iff $(\forall x) \in V :\ s_i,\xi \models sp$

Transition formulae

$(s,s'),\xi \models GS(e)$ iff $s \xrightarrow{e} s'$

$s,\xi \models [e]sp'$ iff for every execution of the event e, if $s \xrightarrow{e} s'$ then the state $s',\xi \models sp'$

This formula is satisfied by a state which evolves to a state s' satisfying sp' for every execution of the event e.

$s_i,\xi \models \langle e\rangle sp'$ iff it exists an execution of the event e, such that if $s \xrightarrow{e} s'$ then the state $s',\xi \models sp'$

This formula is satisfied by a state which can evolve to a state satisfying sp' by the execution of the event e.

Temporal formulae

We interpret a temporal formula on behaviors. In the definitions below, $\sigma|_i,\xi \models F$ means that formula F holds of the suffix of σ from point i onwards.

$\sigma,\xi \models \Box F$ iff $\sigma|_i,\xi \models F$ for all $i \in \mathbb{N}$

The formula $\Box F$ asserts that F is true at all times during the behavior σ.

Leads-to property $F \rightsquigarrow G$

This formula asserts that every suffix satisfying the temporal property F is followed by some suffix satisfying the temporal property G.

$\sigma,\xi \models F \rightsquigarrow G$ iff for all $i \in \mathbb{N}$, if $\sigma|_i,\xi \models F$ then $\sigma|_j,\xi \models G$ for some $j \geq i$

$F \rightsquigarrow G \equiv \Box(F \Rightarrow \Diamond G)$ where $\Diamond G = \neg\Box\neg G$

Weak fairness property

A behavior is weakly fair for some event e iff e occurs infinitely often provided that it is eventually always enabled $(WF(e) \equiv \Diamond\Box\ grd(e) \Rightarrow \Box\Diamond GS(e))$.

$\sigma,\xi \models WF(e)$ iff it exists $j \in \mathbb{N}$ such that for all $i \geq j$, $\sigma|_i,\xi \models grd(e)$ then for all $n \in \mathbb{N}$, it exists $m \in \mathbb{N}$ such that for all $k \geq n+m$, $(s_i,s_k),\xi \models GS(e)$

Strong fairness property

A behavior is strongly fair for some event e iff e occurs infinitely often provided that it is infinitely often enabled $(SF(e) \equiv \Box\Diamond\ grd(e) \Rightarrow \Box\Diamond GS(e))$.

$\sigma,\xi \models SF(e)$ iff for all $i \in \mathbb{N}$, it exists $j \in \mathbb{N}$ such that for all $l \geq i+j$, $\sigma|_l,\xi \models grd(e)$ then for all $n \in \mathbb{N}$, it exists $m \in \mathbb{N}$ such that for all $k \geq n+m$, $(s_i,s_k),\xi \models GS(e)$

4.3 Verification Rules of liveness properties

In this section, we give verification rules (*WF*, *SF* and *LATTICE*) to prove liveness properties under fairness assumptions.

Under weak fairness

Let S be an extended B event system and $WF_{EVENTS}(S)$ is the set of events of the system S satisfying the weak fair assumption. Let $[e]P$ be the weakest pre-condition which ensures that P is true after the execution of the event e. Let $\langle e\rangle P(\neg[e]\neg P)$ be the conjugate weakest pre-condition, i.e. the state from which it is possible for an event e to ensure P. The following rule is used to prove a leads-to formula under a weak fairness assumption.

$$\text{WF.}\ \frac{\begin{array}{l} I \wedge P \wedge \neg Q \Rightarrow [e](P \vee Q) \quad \textit{for all event } e \textit{ of } S \\ \textit{it exists an event } e \textit{ of } S \textit{ where}: \\ \quad I \wedge P \wedge \neg Q \Rightarrow \langle e\rangle\ true \wedge [e]Q \\ \quad e \in WF_{EVENTS}(S) \end{array}}{S \models P \rightsquigarrow Q}$$

In this rule, P and Q are state predicates, I is the invariant of the B event system S. By the first premise, any successor of a state satisfying P has to satisfy P or Q, so P must hold for as long as Q has not been true. By the second premise, it exists a successor of a state satisfying P must satisfy Q and ensures that in every state, the event e is enabled ($\langle e\rangle\ true$ means the feasibility condition of the event e), and so the assumption of weak fairness ensures that e eventually occurs, unless Q has become true before. Finally, the third premise ensures that e is an event for which weak fairness is assumed.

Proof of a liveness property under weak fairness.

To see why the rule is correct, assume that $\sigma = s_0,s_1,...s_i,...$ is a behavior satisfying $\Box I \wedge WF_{EVENTS}(S)$, and that P holds in s_i. We have to show that Q holds of some states s_j with j $\geq$ i. Let s_0 be the initial state and s_i satisfies P. Suppose that no next state satisfies Q, so all next states must satisfy P. By the second premise, it exists a successor of a state satisfying P in which an event e is enabled and always its execution carry out in a state satisfying Q (contradiction). So, from a state s_i satisfying P, we can reach a state $s_j (j \geq i)$ satisfying Q with the execution of an event e under weak fairness.

Under strong fairness

Let S be a B event system and $SF_{EVENTS}(S)$ is the set of strong fair events. As similar to the previous rule, the following rule is used to prove a leads-to formula from a strong fairness assumption.

$$\text{SF.}\ \frac{\begin{array}{l} I \wedge P \wedge \neg Q \Rightarrow [e](P \vee Q) \quad \textit{for all event } e \textit{ of } S \\ \textit{it exists an event } e \textit{ of } S \textit{ where :} \\ \quad I \wedge P \wedge \neg Q \Rightarrow [e]Q \\ \quad S \models \Box(I \wedge P \wedge \neg Q) \Rightarrow \Diamond grd\ (e) \\ \quad e \in SF_{EVENTS}(S) \end{array}}{S \models P \rightsquigarrow Q}$$

In this rule, P and Q are state predicates, I is again an invariant, e is an event for which strong fairness is assumed. We assume that σ is a behavior satisfying $\Box I \wedge SF(e)$ and that P holds of a state s_i. We have to show that Q holds of some s_j with $j \geq i$. By the first premise, any successor of a state satisfying P has to satisfy P or Q. By the second premise, it exists an event $e \in S$ where its execution from a state satisfying p evolves the system to a state satisfying Q. The third premise ensures that in all of these states, the event e is enabled, and so the assumption of strong fairness ensures that eventually e occurs, unless Q has become true before, in which case we are done. Finally, the last premise ensures that e is an event for which strong fairness is assumed.

Using LATTICE rule
The Lattice rule is used to verify complex liveness properties using well-founded relations. $(S, \prec)$ is a binary relation such that there does not exist an infinite descending chain $x_1 \prec x_2, \ldots$ of elements $x_i \in S$. $F$and G are temporal formulae.

$$\text{LATTICE.}\ \frac{\begin{array}{l} (S, \prec)\ \textit{is a well} - \textit{founded relation over}\ S \\ \forall x \in S : F(x) \rightsquigarrow G \vee (\exists y \in S : (y \prec x) \wedge F(y)) \end{array}}{(\exists x \in S : F(x)) \rightsquigarrow G \quad (x\ \textit{not free in}\ G)}$$

In this rule, x and y are rigid variables such that x does not occur in G and y does not occur in F. The second hypothesis of the rule is itself a temporal formula that requires that every occurrence of F, for any value $x \in S$, be followed either by an occurrence of G, or again by some F, for some smaller value y. Because the first hypothesis ensures that there cannot be an infinite descending chain of values in S, eventually G must become true. This rule allows us to derive liveness properties by induction over some well-founded ordering.

Other verification rules

$$\frac{P \rightsquigarrow Q \quad Q \rightsquigarrow R}{P \rightsquigarrow R}\ (trans) \qquad \frac{P \rightsquigarrow Q \quad R \rightsquigarrow Q}{P \vee R \rightsquigarrow Q}\ (disj)$$

$$\frac{P \Rightarrow Q}{P \rightsquigarrow Q}\ (dedu) \qquad \frac{P \rightsquigarrow Q}{(\exists x : P(x)) \rightsquigarrow (\exists x : Q(x))}\ (exists)$$

These rules can be used to prove complex *Leadsto* formulas.

Application to the example.
We try now to prove the following property :
$\forall p.(p \in UNSORTED) \rightsquigarrow arrived(p) = adr(p))$.

Let $P \equiv \forall p.(p \in UNSORTED)$, $Q \equiv arrived(p) = adr(p)$ and I be the invariant of the system S. Let e be an event of S. We have : $I \wedge P \wedge \neg Q \Rightarrow [e](P \vee Q)$ $for\ all\ event\ e\ of\ S$. With the event *cross_parcel*, we have : $I \wedge P \wedge \neg Q \Rightarrow \langle cross\_parcel \rangle\ true \wedge [cross\_parcel]Q$ and $e \in WF_{EVENTS}(S)$. So by applying the rule WF, we have $S \models P \rightsquigarrow Q$.

5 Conclusion

In this paper, we have built an extension of the event B method to deal with fairness and eventuality properties. We have proposed a semantics of the extension over traces, in the same spirit as TLA$^+$ does and we have given verification rules in the axiomatic of the event B method.

In future work, we plan to define new required proof obligations. Moreover, the B prover may not be enough powerful for proving new proof obligations. Future work will explore also, the question of the refinement and the properties of refinement, within the extended language.

ACKNOWLEDGEMENTS Thanks to Stephan Merz for comments on an earlier draft of this paper.

REFERENCES

Abrial, J.-R. (1996). Extending B without changing it (for developing distributed systems). In Habrias, H., editor, *Proceedings of the 1st Conference on the B method*, pages 169–191.

Abrial, J.-R. (2003). B# : Toward a synthesis between Z and B. In Bert, D., Bowen, J. P., King, S., and Waldén, M., editors, *ZB'2003 – Formal Specification and Development in Z and B*, volume 2651 of *Lecture Notes in Computer Science (Springer-Verlag)*, pages 168–177, Turku, Finland. Springer.

Abrial, J.-R. and Mussat, L. (1998). Introducing dynamic constraints in B. In Bert, D., editor, *B'98 : The 2nd International B Conference*, volume 1393 of *Lecture Notes in Computer Science (Springer-Verlag)*, pages 83–128, Montpellier. Springer Verlag.

Back, R.-J. and K-Sere (1989). Stepwise refinement of action systems. In *Mathematics of Program Construction.*, pages 115–138, Berlin - Heidelberg - New York. Springer.

Back, R.-J. and v. Wright, J. (1998). *Refinement Calculus: A Systematic Introduction*. Graduate Texts in Computer Science. Springer-Verlag.

Barradas, H. R. and Bert, D. (2002). Specification and proof of liveness properties under fairness assumptions in B event systems. In *IFM*, pages 360–379.

ClearSy (2002). Atelier b. Technical Note Version 3.6, Aix-en-Provence(F).

Jaray, J. and A.Mahjoub (1996). Une mthode itrative de construction d'un modle de systme ractif . *TSI*, 15. .

Lamport, L. (1994). The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923.

Leuschel, M. and Butler, M. (2003). ProB: A model checker for B. In Araki, K., Gnesi, S., and Mandrioli, D., editors, *FME 2003: Formal Methods*, LNCS 2805, pages 855–874. Springer-Verlag.

Spivey, J.-M. (1988). Understanding Z, A Specification Language and its Formal Semantics. *Tracts in Theoretical Computer Science*, 3. Cambridge University Press.