A Formal Approach for the Development of Automated Systems

Olfa Mosbahi, Leila Jemni, Jacques Jaray

A FORMAL APPROACH FOR THE DEVELOPMENT OF AUTOMATED SYSTEMS

Olfa MOSBAHI, Leila JEMNI
LORIA, INRIA Lorraine, Nancy University, France
Faculté des Sciences, El Manar II University, Tunisie
mosbahi@loria.fr, leila.jemni@fsegt.rnu.tn

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

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

Abstract: This paper deals with the use of two verification approaches : theorem proving and model checking. We focus on the event B method by using its associated theorem proving tool (Click_n_Prove), and on the language TLA$^+$ by using its model checker TLC. By considering the limitation of the event B method to invariance properties, we propose to apply the language TLA$^+$ to verify liveness properties on a software behavior.
We extend first of all the expressivity of a B model (called *temporal B model*) to deal with the specification of fairness and eventuality properties. Second, we give transformation rules from a temporal B model into a TLA$^+$ module. We present in particular, our prototype system called B2TLA$^+$, that we have developed to support this transformation. Finally, we verify these properties thanks to the TLC model checker.

1 INTRODUCTION

In the paper, we propose to combine and apply two techniques: theorem proving, when possible, and model checking otherwise, in the construction and verification of safe automated systems. The theorem prover concerned is part of the B toolkit associated to the event B method and the model checker is TLC for TLA$^+$ models. The B event 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 in automated systems. The language, TLA$^+$, provides us with an abstract and powerful framework for modelling, specifying and verifying safety, eventuality, fairness behavioral properties of reactive systems.

We propose to define briefly the syntax and the semantics of the extension of a B model (called *temporal B*) to deal with the specification of fairness and eventuality properties. Then, we give the transformation rules from a temporal B model into a TLA$^+$ module and we present a prototype system called B2TLA$^+$ supporting these transformation rules. Finally, we verify these properties thanks to the TLC model checker. The second part of our contribution describes a development method combining the two methods with their associated tools.

We suppose that the specifier is familiar with the use of the B technology but not with TLA$^+$. He uses the prototype system to translate a temporal B model into a TLA$^+$ module and then he verifies liveness properties with the TLC model checker. To our knowledge, there is no previous work which resolved the problem that we tackle in this paper.

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 the proposed approach using a case study and section 5 ends with a conclusion and perspectives.

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*).

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.

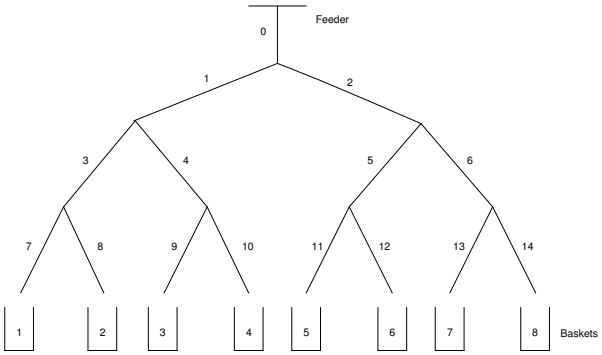

Figure 1: Router

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 figure 2.

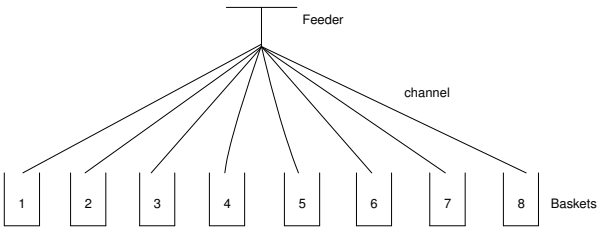

Figure 2: Router

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 be introduced into the sorting layout and gate opening (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*.

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 that no parcel is processed.

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.

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.

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.

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* : $\in$ *Baskets* || *sorting* := *free* ||
pe : $\in$ *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* $\cup$ { *pe*} ||
pe : $\in$ *NOPARCELS* || *sorting* := *free*
END
END

Figure 3: Abstract model of the sorting device

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 semantics of TLA is based on behaviors of state variables.

A TLA specification of a system denoted by $Spec(S)$ looks like : $Init \wedge \Box[Next]_x \wedge L$ where $Init$ is the predicate which specifies initial states ($s_0[|Init|]$), $Next$ is an action (a relation) that describes the next-state relation and L is a fairness assumption (strong or weak) on actions.

TLA$^+$ (Lamport, 2002) is an extension of TLA with ZF set theory and modules for structuring a specification. A module is a text containing a name, a list of definitions (constants, variables, operators, functions, predicates, assumptions, theorems, proofs).

4 Proposed approach

The event B method deals with safety properties, but there are applications, such as automated or distributed systems, where liveness properties must be considered. TLA$^+$ deals with safety, eventuality and fairness properties. We suggest the use of TLA$^+$ because these two methods are very close with respect to their foundations. In this proposed method, we suppose that the specifier uses the event B method for developing automated systems and he is not familiar with TLA$^+$ modelling. He uses the prototype system B2TLA$^+$, witch we have developed to transform a temporal B model into TLA$^+$ module. Then he can verifies liveness properties using the TLC model checker.

The proposed approach as shown in figure 4 uses the event B method and the temporal logic TLA$^+$ as follows:

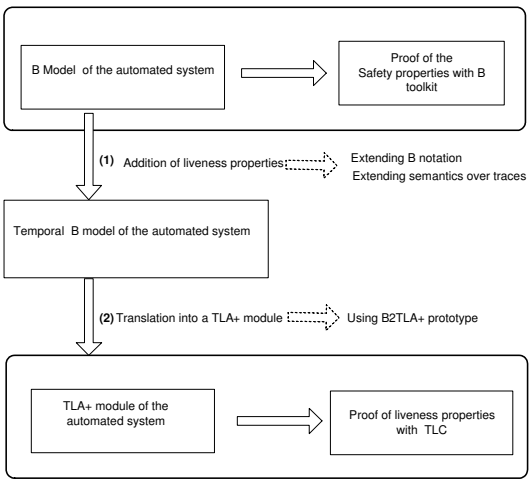

Figure 4: Translation of a B model into a TLA$^+$ module

1. At first, an abstract model of the system is given in the event B method where only invariance properties are considered,
2. Invariants properties are verified with the Click_n_Prove tool,
3. Liveness properties are added to the abstract B model when necessary. The FAIRNESS clause is used for specifying fairness properties and EVENTUALITY clause for the specification of eventuality properties. The obtained model is as follows :

MODEL ⟨*name*⟩
SETS ⟨*list of sets*⟩
CONSTANTS ⟨*constants*⟩
PROPERTIES ⟨*Properties*⟩
VARIABLES ⟨*variables*⟩
INVARIANT ⟨*invariants*⟩
INITIALISATION ⟨*initialization of variables*⟩
EVENTS ⟨*events*⟩
FAIRNESS ⟨*Fairness properties*⟩
EVENTUALITY ⟨*Eventuality properties*⟩
END

4. The new temporal model is then translated into a TLA$^+$ module using the prototype system B2TLA+ which we have developed,
5. Liveness properties are checked with TLC,
6. In each refinement step, we repeat the steps 1, 2, 3 and 4 until satisfying all properties of the system. The refinement is shown in the figure 5.

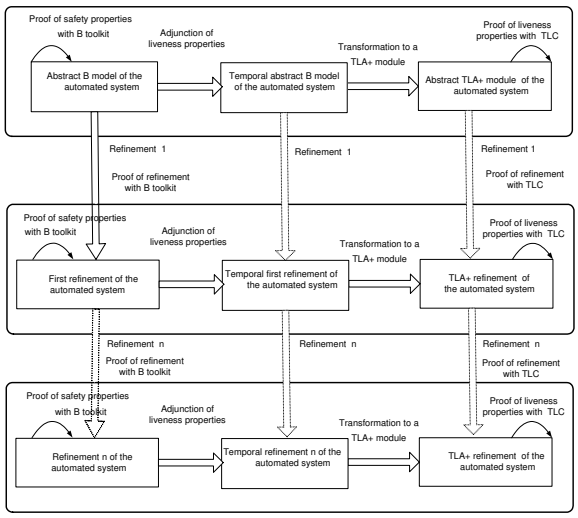

Figure 5: Refinement for automated systems development

In the sequel we will focus on the translation of the extended B model into a TLA$^+$ module and the presentation of the prototype B2TLA$^+$.

Translation rules from a B model into TLA$^+$ module. In this section, we present translation rules from a temporal B model into a TLA$^+$ module. The tables below show the correspondance between B syntax and TLA$^+$ one.

- *Clauses key word correspondance.* In the table below, we will give the different clauses from a temporal B model translated in a TLA$^+$ module.
- *Modelling of data.* Both B and TLA$^+$ are based on Zermelo-Frankel set theory in which every value is a set. The modelling of data is based on constants, variables, sets, relations and functions. In the tables below, we will give the translation rules of data from a temporal B model to a TLA$^+$ module.

Table 1: Correspondance with clauses

B model	*TLA+ module*
SYSTEM	MODULE
REFINEMENT	MODULE
REFINES	EXTENDS
CONSTANTS	CONSTANTS
VARIABLES	VARIABLES
PROPERTIES	ASSUME
INVARIANT	INVARIANT
INITIALISATION	INIT
ASSERTIONS	THEOREM
FAIRNESS	FAIRNESS
EVENTUALITY	LIVENESS

Table 2: Constants, Variables and Sets

B model	*TLA+ module*
c_1, c_2 in CONSTANTS	c_1, c_2 in CONSTANTS
v_1, v_2 in VARIABLES	v_1, v_2 in VARIABLES
E_1 in SETS $E_2 = \{x_1, .., x_n\}$ in SETS	E_1 in CONSTANTS $x_1, .., x_n$ in CONSTANTS and $E_2 = \{x_1, .., x_n\}$ occurs after the clause ASSUME
$\{e_1, ..., e_n\}$	$\{e_1, ..., e_n\}$
$\{x \mid x \in S \wedge P\}$	$\{x \in S : P\}$
$\{e \mid x \in S\}$	$\{e : x \in S\}$
$\mathbb{P}(S)$	SUBSET S

The event B method has used a simplification of classical set theory. The only basic set classical theoretic constructs axiomatized are the most natural ones : cartesian product ($\times$), power-set ($\mathbb{P}$) and set comprehension ($\{ \mid \}$). A set is defined in comprehension with respect to another set s when the members of the former are exactly the members of s satisfying a certain predicate. The set theory in TLA$^+$ is based on the simple forms of the constructs $\{x \in S : P\}$ (the subset of S consisting of all elements x satisfying property P) and $\{e : x \in S\}$ (the set of elements of the form e, for all x in the set S) as primitives, and the more general forms are defined in terms of them. In TLA$^+$, the two powerful operators of set theory are UNION and SUBSET, defined as follows : UNION S is the union of the elements of S and SUBSET S is the set of all subsets of S. $T \in$ SUBSET S iff $T \subseteq S$ (it is the power set of $\mathbb{P}(S)$ or 2^S). We find correspondance between the B and TLA$^+$ set theories.

In the table below, we will give correspondance between functions in B and TLA$^+$.

In the table 3, the symbols $\rightarrow, \nrightarrow, \twoheadrightarrow, S \twoheadrightarrow T, \rightarrowtail, S \rightarrowtail T$ describes respectively total and partial function, total and partial surjection and total and partial injection. We associate the value *undef* to the elements where the function is not defined because in TLA$^+$ we have just total function.

- *Events.* Events in B method are actions in the temporal

Table 3: Functions translation

B model	*TLA+ module*
$f(e)$	$f[e]$
$Dom(f)$	$Domain\ f$
$(\lambda x.x \in S \mid e)$	$[x \in S \mapsto e]$
$S \rightarrow T, S \nrightarrow T, S \rightarrowtail T$	$[S \rightarrow T]$
$S \nrightarrow T, S \nrightarrow\!\!\!\!\!\rightarrow T, S \nrightarrowtail T$	$[S \rightarrow T \cup undef]$
$f[S]$	$Image(f, S) \equiv \{f[x] : x \in S\}$
$f(x) := y$	$f' = [f\ EXCEPT\ ![x] = y]$

logic TLA$^+$; in the following, we give the translation of the three event types B into TLA$^+$ actions.

Table 4: Events translation

B event	*TLA+ action*
$E \equiv BEGIN$ $x : P(x_0, x)$ END;	$E \equiv \wedge P(x, x')$ $\wedge UNCHANGED\langle$vars$\rangle$
$E \equiv SELECT$ $G(x)$ $THEN$ $x : Q(x_0, x)$ END;	$E \equiv \wedge G(x)$ $\wedge Q(x, x')$ $\wedge UNCHANGED\langle$vars$\rangle$
$E \equiv ANY$ $t\ WHERE$ $G(t, x)$ $THEN$ $x : R(x_0, x, t)$ END;	$E \equiv \exists t : \wedge G(t, x)$ $\wedge R(x, x', t)$ $\wedge UNCHANGED\langle$vars$\rangle$

- *Safety properties.* The syntax of safety properties in B and TLA$^+$ are quite similar because the two methods are based on the first order logic. The table 5 provides a translation of safety formulae from B to TLA$^+$.
- *Liveness properties.* The translation is obvious because the extension is inspired by the syntax of TLA$^+$. In the following, we give the translation rules of liveness properties from a B model to a TLA$^+$ module.

  Where e is an event in B and an action in TLA$^+$, F and G are liveness properties.

The prototype system B2TLA$^+$.
The transformation of a B model into TLA$^+$ module is based on the technique of translation syntax-directed which consists in adding semantics actions in the B grammar to

Table 5: Translation of safety formulae from B to TLA$^+$.

B formula	*TLA$^+$ formula*
P	P
$\neg P$	$\neg P$
$P \wedge Q$	$P \wedge Q$
$P \vee Q$	$P \vee Q$
$P \Rightarrow Q$	$P \Rightarrow Q$
$P \Leftrightarrow Q$	$P \Leftrightarrow Q$
$!x.(P \Rightarrow Q)$	$\forall x : (P \Rightarrow Q)$
$!(x, y).(P \Rightarrow Q)$	$\forall (x, y) : (P \Rightarrow Q)$
$\exists x.(P \wedge Q)$	$\exists x : (P \wedge Q)$

Table 6: Translation of temporal formulae

B formula	*TLA+ formula*	*Definition*
$WF(e)$	$WF(e)$	weak fairness for the event e
$SF(e)$	$SF(e)$	strong fairness for the event e
$F \rightsquigarrow G$	$F \rightsquigarrow G$	F leads to G
$\Box F$	$\Box F$	F is always true
$\Diamond F$	$\Diamond F$	F is eventually true

obtain an equivalent TLA$^+$ module. Based on our approach to ensure the verification of safety and liveness properties, we implement a prototype system. The figure 6 describes the architecture of the system. The prototype system is implemented using the Flex-Yacc tool. Liveness properties are given by the user. He introduces fairness conditions and eventuality properties to be verified. He gives the fairness condition (weak fairness, strong fairness) for each event. These properties are added to a B model to obtain a temporal B model.

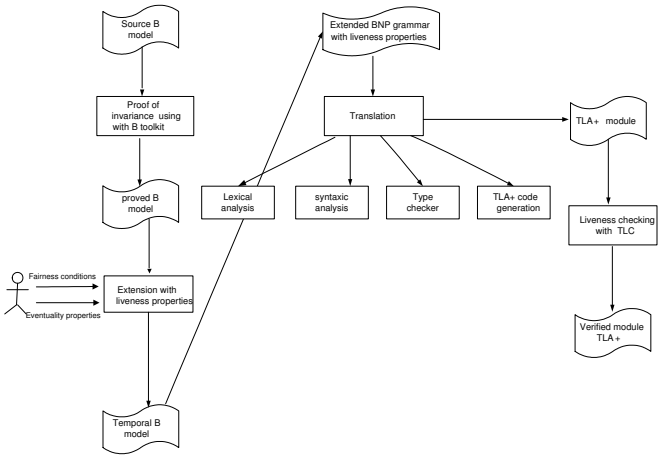

Figure 6: The proposed method using the prototype system B2TLA+

Application to the example.
In the following, we give the module TLA$^+$ obtained after translation of a B temporal model.

MODULE $Parcel\_Sorting$

EXTENDS $Naturals$
CONSTANTS $PPARCELS, PARCELS, Baskets, noBaskets, adr, free, busy$
VARIABLES $channel, sorting, pe, sorted, arrived, ready\_to\_sort$
ASSUME $adr \in [PARCELS \rightarrow Baskets]$
$\wedge\ PARCELS \neq \{\}$
$\wedge\ Baskets \neq \{\}$
$\wedge\ PARCELS \subseteq PPARCELS$
$\wedge\ PARCELS \neq PPARCELS$
$\wedge\ noBaskets \notin Baskets$
$SortingState \triangleq \{free, busy\}$

$TypeInvariant \triangleq$
$\wedge\ channel \in Baskets \cup \{noBaskets\}$
$\wedge\ sorting \in SortingState$
$\wedge\ pe \in PPARCELS$
$\wedge\ ready\_to\_sort \in$ BOOLEAN
$\wedge\ arrived \in [PARCELS \rightarrow Baskets]$
$\wedge\ sorted \subseteq PARCELS$

$$
\begin{array}{l}
UNSORTED \triangleq PARCELS \setminus sorted \\
NOPARCELS \triangleq PPARCELS \setminus PARCELS \\
Init \triangleq \land channel = noBaskets \land sorting = free \\
\qquad \land pe \in NOPARCELS \\
\qquad \land arrived = [p \in PARCELS \mapsto noBaskets] \\
\qquad \land sorted = \{\} \land ready\_to\_sort = \text{FALSE} \\
select\_parcel \triangleq \land sorting = free \\
\qquad \land pe \in NOPARCELS \\
\qquad \land \exists p \in UNSORTED : \land pe' = p \\
\qquad \land \text{UNCHANGED } \langle channel, arrived, sorted, \\
\qquad\qquad sorting, ready\_to\_sort \rangle \\
set\_channel \triangleq \land sorting = free \\
\qquad \land pe \notin NOPARCELS \\
\qquad \land ready\_to\_sort = \text{FALSE} \\
\qquad \land channel' = adr[pe] \\
\qquad \land ready\_to\_sort' = \text{TRUE} \\
\qquad \land \text{UNCHANGED } \langle arrived, sorted, pe, sorting \rangle \\
release \triangleq \land ready\_to\_sort = \text{TRUE} \\
\qquad \land sorting = free \land pe \in PARCELS \\
\qquad \land sorting' = busy \\
\qquad \land ready\_to\_sort' = \text{FALSE} \\
\qquad \land \text{UNCHANGED } \langle channel, arrived, sorted, pe \rangle \\
cross\_parcel \triangleq \land sorting = busy \\
\qquad \land arrived' = [arrived \text{ EXCEPT } ![pe] = channel] \\
\qquad \land sorted' = sorted \cup \{pe\} \land pe' \in NOPARCELS \\
\qquad \land sorting' = free \\
\qquad \land \text{UNCHANGED } \langle channel, ready\_to\_sort \rangle \\
Next \triangleq \lor select\_parcel \lor set\_channel \\
\qquad \lor release \lor cross\_parcel
\end{array}
$$

$$
\begin{array}{l}
trvars \triangleq \langle channel, sorting, pe, arrived, sorted, ready\_to\_sort \rangle \\
Fairness \triangleq \land \text{WF}_{trvars}(select\_parcel) \\
\qquad \land \text{WF}_{trvars}(set\_channel) \\
\qquad \land \text{WF}_{trvars}(release) \\
\qquad \land \text{WF}_{trvars}(cross\_parcel) \\
Spec \triangleq Init \land \Box[Next]_{trvars} \land Fairness \\
Invariant \triangleq \\
\qquad \land sorting = busy \implies channel = adr[pe] \\
\qquad \land sorting \neq free \implies pe \in PARCELS \\
\qquad \land sorting = busy \implies ready\_to\_sort = \text{FALSE} \\
\qquad \land ready\_to\_sort = \text{TRUE} \implies channel = adr[pe] \\
\qquad \land ready\_to\_sort = \text{TRUE} \implies pe \in PARCELS \\
Liveness \triangleq \\
\qquad \land \forall p \in UNSORTED \rightsquigarrow arrived[p] = adr[p] \\
\qquad \land \forall p \in UNSORTED \implies \Diamond arrived[p] \in Baskets
\end{array}
$$

$$
\begin{array}{ll}
\text{THEOREM} & \text{Spec} \implies \Box TypeInvariant \\
\text{THEOREM} & Spec \implies \Box Invariant \\
\text{THEOREM} & Spec \implies Liveness
\end{array}
$$

The TLC model checker can check a finite model of a specification obtained by instantiating the constant parameters and, if necessary, specifying constraints to make the set of reachable states finite. With TLC, we can check that a specification satisfies invariance or liveness specifications. Also, we can check refinement such that a TLA$^+$ specification implies another using TLC.

5 Conclusion

In this paper we propose a method for the specification and verification of safety and liveness properties using the event B method and TLA$^+$ with their associated tools. We first extende the expressivity of the event B method to deal with fairness and eventuality properties. Second we propose a semantics of the extension in terms of traces, in the same spirit as TLA$^+$ does. Third we give transformation rules from a temporal B model to a TLA$^+$ module and we develop a prototype system (B2TLA$^+$) which supports this translation. Then we verify these properties using the TLC model checker. Our main idea is to start with an abstract B model of the system under development and then we verify all invariants of the model. We use a proof tool which supports automatic and interactive proof procedures. For the expression and the verification of fairness and eventuality properties, we can't use the event B method because its semantics is expressed in the *weakest precondition calculus*. Our goal is to give a temporal meaning to a B model and then transform it to an equivalent TLA$^+$ module. TLA$^+$ and B are quite similar, both being based on simple mathematics and when a system is viewed as a set of events, it is easy to characterize it by a TLA$^+$ specification. TLA$^+$ stands as a semantical framework for an extension of B and the expression of fairness and eventuality constraints is a very powerful point of TLA$^+$. Then, we verify these properties thanks to the TLC model checker. TLC allows us to analyse finite-state instances.

In future work, for the verification of infinite-state systems, we propose the use of the predicate diagrams (Cansell et al., 2001). These diagrams are finite graphs whose nodes are labelled with sets of predicate and whose edges are labelled with action names. Besides abstract states and state transition which are necessary to reason about safety properties of systems, predicate diagrams also include annotation related to fairness conditions and well-founded orderings and can therefore be used to reason about liveness properties. We plan to use these predicate diagrams to verify infinite-state-model and also refinement between temporal B refinements. We plan to use also DIXIT, a graphical toolkit supporting the use of predicate diagrams, which includes annotations for proving liveness properties. It is a graphical editor for drawing predicate diagrams and proof obligations for proving correctness of the specifications. The toolkit also supports stepwise development of systems, based on a notion of refinement of predicate diagrams.

ACKNOWLEDGEMENTS Thanks to Stephan Merz for serious 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.

Cansell, D., Méry, D., and Merz, S. (2001). Diagram refinements for the design of reactive systems. *Journal of Universal Computer Science*, 7(2):159–174.

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.

Lamport, L. (2002). *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley.