Formal Development Method of Automated Systems using the Temporal Logic of Actions TLA

Olfa Mosbahi, Leila Jemni Ben Ayed, Jacques Jaray

6e Conférence Francophone de MOdélisation et SIMulation - MOSIM'06 - du 3 au 5 avril 2006 - Rabat - Maroc
«Modélisation, Optimisation et Simulation des Systèmes : Défis et Opportunités»

Formal development method of automated systems using the temporal logic of Actions TLA+

O. MOSBAHI
Faculté des Sciences de Tunis, INRIA Lorraine
Université Tunis El Manar II, LORIA-INPL
Tunis, Tunisie, Nancy, France
olfa.mosbahi@loria.fr

L. JEMNI BEN AYED
Faculté des Sciences de Tunis
Université Tunis El Manar II
Tunis, Tunisie
leila.jemni@fsegt.rnu.tn

J. JARAY
INRIA Lorraine
LORIA-INPL
Nancy, France
jacques.jaray@loria.fr

Abstract : *The paper presents a method for control systems formal development. We focus on the refinement process used for the development of a control part controlling an operative part of an automated (controlled) system satisfying requirements. We first build an abstract model of both operative and control parts and complete this model to get a model of the automated system. The next steps consists in refining the control part and the operative one to get a model of the automated system capturing every important feature. The method is developed through a case study : a parcel sorting system. We use the temporal logic of actions TLA+ which deals with refinement and proved usufull for the specification and the verification of safety and liveness properties.*
Key-Words : *Modelling, Model checking, Refinement, Automated systems, TLA+, Validation.*

1 INTRODUCTION

Reactive systems are systems that react continuously to their environment and require description of allowed patterns of behaviour for their specification, rather than a simple input/output relation. Reactive systems often involve concurrent execution of processes (in order to achieve responsiveness requirements, or because of inherent distribution in the application) and requirements on system states that must not arise (safety constraints), the reachability of some states (liveness constraints) and timing constraints between system responses to events. Among reactive systems are most of the industrial real-time and embedded systems, such as control and supervision systems. They are critical systems and require a high level of safety and reliability. To reach a necessary degree of reliability and safety, it would be quite interesting to lay out a specification approach which simplifies the requirement description, deals with mathematical notations inducing verification and validation.

In this paper, we propose a development method of automated (controlled) systems. In such systems, we distinguishe a software part : the "controller" and an operative part formed by a physical device and its environment which the behavior can be observed through variables and on which it is possible to react with effectors. The controller and the operative part form what is called the controlled (or automated) system. The goal is that the controller acts on the operative part such as the controlled system satisfies required properties. So, we are concerned with the correctness of the controller. The usual way to deal with the correctness of a program is to prove that the interaction with its operative part satisfies some required properties.

We take a different point of view and notice that the user is satisfied by the behaviour of the controller as soon as the automated system behaves in the expected way. Our development method consists in building an abstract discrete model of both operative and control parts. The model of the operative part is completed to obtain a controlled or automated system. Correctness concerns the controlled system, provided the model of the operative part is accurate. Successive refinements are used to produce final model for each of the control and the operative parts. At each refinement level new required properties are added to the model and have to be verified by the automated system. The last refinement step shall verify all requirements. These insures that the final model will be correct since all required properties have been verify (Figure 2). The quality of the last refinement suposed to model the controlled system depends on the first abstract models of both of

the control and operative parts.

Several formal methods have been proposed for the development of control systems, most of them relay on set theory such as B, VDM and Z other ones relay on temporal logic such as PLTL and TLA. To build our models we use the temporal logic of actions TLA+ which has been proved usefull for the specification and the verification of safety,liveness and fairenss properties (Lamport 2002, Lamport 1998, Lamport 1993), deals with refinement process, wich we need in our approach, and is provided with a powerfull model checker TLC (Lamport 2002) which can be used to simulate the automated system and to prove the correctness of the development. The expected behaviour of the controlled system is expressed by means of invariants and temporal properties. The model is then refined such that its operative part gets closer to the real operative one.

The paper is organized as follows : section 1 presents an overview of the temporal logic of actions, section 2 a description of the development method that we propose. In section 3 the method is illustrated by a parcel sorting case study and the model checker TLC is used.

2 TLA+ : THE TEMPORAL LOGIC OF ACTIONS

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 1991, 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 state behaviors of variables. It can be viewed as a logic built in an incremental manner in three stages :

1. predicates having as free variables rigid and flexible variables and whose semantics is based on states. A state s satisfies the predicate P if and only if $s[|P|]$ is true. $s[|P|]$ is the value obtained by substituting in P variables by its values in the state s (for instance $s[|x = 0|] \equiv s[|x|]^1 = 0$),

2. actions which are logical formulas having primed flexible variables as well as free variables and whose semantics is based on pairs of states. A pair of state $< s, t >$ satisfies the action A if and only if $s[|A|]t$ is true. $s[|A|]t$ is the value obtained by substituing in A imprimed variables by its value in the state s and primed variables by its values in the state t. (for instance $s[|x' = x + 1|]t \equiv t[|x|] = s[|x|] + 1$). As predicates are actions we thus have $s[|P|]t \equiv s[|P|]$ and $s[|P'|]t \equiv t[|P|]$,

3. temporal formulas of actions (addition of the operator $\Box$) whose semantics is based on state behaviors of variables. $< s_0, s_1, ... > [|\Box T|]$ (a behavior $< s_0, s_1, ... >$ satisfies $\Box T$) is true if and only if $\forall n \in Nat :< s_n, s_{n+1}, ... > [|T|]$. As an action is a temporal formula we have $< s_0, s_1, ... > [|A|] \equiv s_0[|A|]s_1$.

A TLA specification looks like : $Init \wedge \Box[Next]_x \wedge L$ Where :

1. *Init* is the predicate which specifies initial states ($s_0[|Init|]$),

2. $\Box[Next]_x$ means that either two consecutive states are equal on $x, x' = x$ (stuttering), or *Next* is an action (a relation) which binds two consecutive states by using the variable (not primed) for the first state and the primed variable for the second state ($\forall n \in Nat : s_n[|Next|]s_{n+1}$),

3. L is a fairness assumption (strong or weak) on actions $A : A \Rightarrow Next$. $WF_{umprimed\ variables[|S|]}(s)$ defines the condition of weak fairness over the system S and $SF_{umprimed\ variables[|S|]}(s)$ defines the condition of strong fairness over the system S.

We use TLA to prove invariance properties ($Spec \Rightarrow \Box I$), eventuality properties ($Spec \Rightarrow \Diamond F$) or $Spec \Rightarrow (P \rightsquigarrow Q)$) and refinement properties ($Spec\_ref \Rightarrow Spec\_abs$). Unlike most other temporal logics, TLA is intended to support stepwise system development by refinement of specifications. The basic idea of refinement consists in successively adding implementation detail while preserving the properties required at an abstract level. In a refinement-based approach to system development, one proceeds by writing successive models, each of which introduces some additional datail while preserving the properties of the preceding model. Fundamental properties of a system can thus be established at high levels of abstraction, errors can be detected in early phases, and the complexity of formal assurance is spread over the entire development process. A system S_1 is refined by a system S_2, when : Specification(S_2) $\Rightarrow$ Specification(S_1). A refinement S_2 preserves all TLA properties of an abstract specification S_1 if and only if for every formula F, if $S_1 \Rightarrow F$ is valid, then so is $S_2 \Rightarrow F$. This condition is in turn equivalent to requiring the validity of $S_2 \Rightarrow S_1$.

Because S_2 will contain extra variables to represent the lower-level detail, and because these variables will change in transitions that have no counter-part at the abstract level, stuttering invariance of TLA formulas is essentiel to make validity of implication a reasonable definition of refinement.

Proofs in TLA (can be found in (Lamport 1991)) are carried out using proof rules and rules of the classical logic. It is clear that the main advantage is that every thing is stated as a logical object and relations as probability or refinement are stated using the logical implication. The refinement states a property between logical formulae interpreted over external behaviour.

TLA+ (Lamport 2002, Lamport 1998) is an extension of TLA including predicate calculus and ZF set theory and mechanisms for structuring a specification in a module. A module is a text containing a name, a list of definitions (constants, variables, operators, functions, predicates, assumptions, theorems, proofs). A specification is made up of several modules that are combined using clauses EXTENDS and INSTANCE. The clause EXTENDS imports the definitions of specified modules by a macro expansion mechanism; the clause INSTANCE provides the mechanism of parametrization and a bloc-like structure. A module TLA+ is represented as follows :

MODULE <*Name*>
CONSTANTS <*List of contants*>
VARIABLES <*List of variables*>
ASSUME <*properties of constants*>
TYPE INVARIANT
INIT <*initialization of variables*>
SPEC <$SpecInit \wedge \Box[Next]_x \wedge L$>
INVARIANT <*Safety properties*>
LIVENESS <*Liveness properties*>
THEOREM
END

3 THE DEVELOPMENT METHOD OF AUTOMATED SYSTEMS

In this section we present a development method of automated systems. The problem can be stated as follows: being given an operative part (physical device and an environment) which behavior is observed through a number of variables and on which it is possible to act by means of effectors. Some of the variables are controllable, they correspond to the results of some actuators, other are not controllable and their

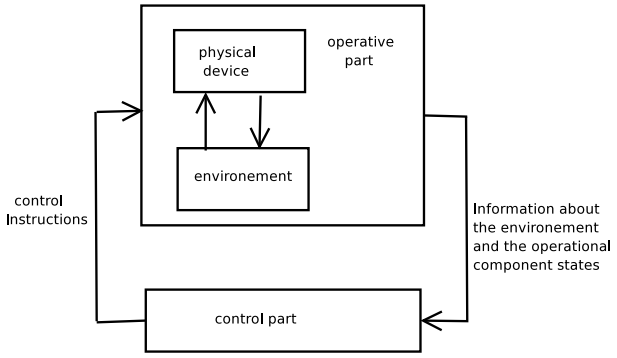

Figure 1. Automated system in closed loop

changes correspond to the effect of some physical process we do not control. The behavior of the automated system can be represented by a closed loop expressing relation between operative and control parts (Figure 1). We have to develop a control program which ob-serves the operative part and change controllable variables such that the automated system meets some requirements . The modeling method that we propose uses the temporal logic of actions (TLA+) and consists on the following steps:

1. First of all, we build an abstract model of the operative part, an abstract model of minimal controller in defining its actions with guards as weak as possible, just taking into account safety requirements. The resulting system is a *"permissive"* controlled system, it is likely not the case that all behaviors meet the automated system requirements but some should.

2. We have to prove the last part of this assertion, if not we can deduce that it is not possible to achieve the expected system. This point is similar to the notion of controllability in control theory (Ramadge & Wonham 1989). Then, we constrain the *permissive* controlled system as a module TLA+ and we use the model checker TLC to verify some safety requirements.

3. Successive refinements can be used to produce the concrete model of the automated system formed by physical device and its environment and the concrete control system through the addition of expected behaviour with the addition of new actions, variables and new properties for each of the operative and the control parts (Figure 2). Related verifications are discharged. At each refinement step, new module modeling operative and control parts is shown to satisfy new automated system requirements. The refinement process stops when the automated system model meets the desired properties. The quality of the last refinement supposed to model the controlled

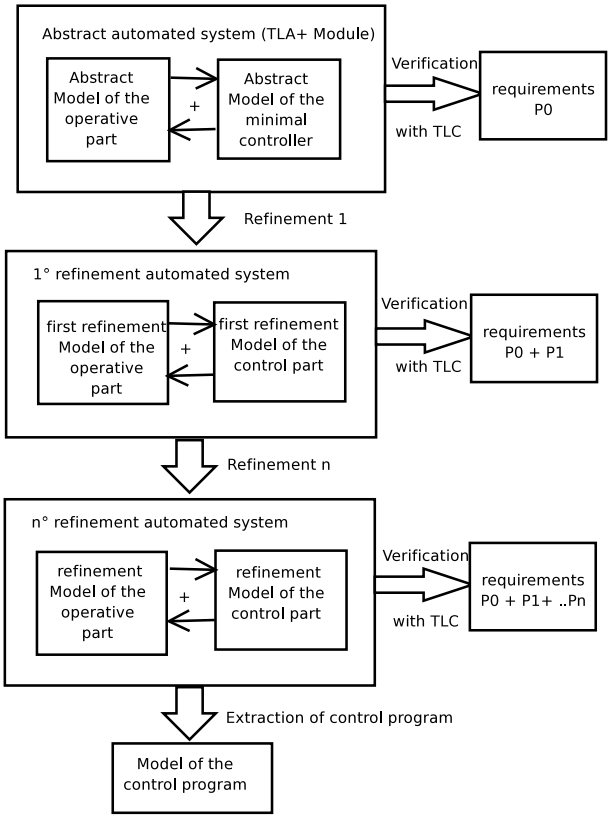

Figure 2. Development method of automated systems

system depends on the first operative part model.

4. The last step consists in separating what concerns the operative component from what concerns the controller in order to build a program of the controller,

4 CASE STUDY : A PARCEL SORTING DEVICE

Our method is illustrated by a case study : the development of a parcel sorting system (JARAY & A.Mahjoub 1994). We start with an informal description of the problem and then apply our method.

4.1 Informal description of the problem

The problem is to sort parcels into sorting 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 Figure 3. The switches are the nodes of the tree, pipes

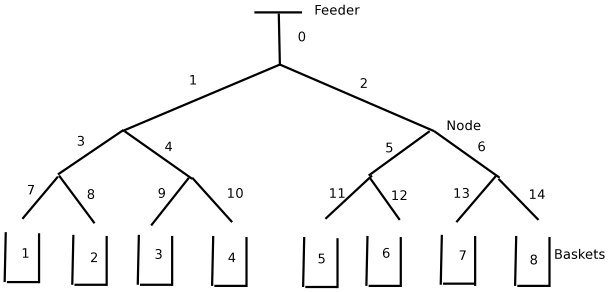

Figure 3. Sorting plant

are the edges and leaves are the baskets. A parcel, thanks to gravity, can slide down through switches and pipes to reach a basket. 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 a parcel at once 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 security 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.

4.2 Abstract model of the system

The controller extends the operative part in order to produce the controlled system. The first step of the method is to build an abstract model of the operative part and an abstract model of the control part using inputs from the informal presentation. The operative part is formed by the sorting device and the parcel behaviour.

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 actions *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 only one sorting basket. A basket is an element of a set named *Basket*. In future refinements, the tree structure of the sorting device will be introduced and the sorting baskets will be classed as nodes of the tree (leaves). Therefore, we introduce the set *Nodes* having *Basket* as a proper subset. 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 *Basket*. The *channel* value is changed by the action *set_channel*, defined in the control part. The full description of the actions will be given in the controller section. It is worth noticing that the abstraction forces a "sequential functionning" 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. Up to a point, it is indeed possible to sort more than one parcel simultaneously. The chosen abstraction does not allow such a concurrency.

Parcels. Parcels, as part of the environment, are components of the operative part and 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. The action concerning a selected parcel to sort is the crossing of the sorting device up to the basket the channel leads to. We give the status "*arrived*" to the parcel which has reached a sorting basket. The variable (*arrived*) is a total 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*. *pe* describes an entry parcel. 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 action *select_parcel* is introduced and is activated once the device is free and the variable *pe* is undefined, which means it does not exist a parcel being sorted.

$$
\begin{array}{l}
select\_parcel \triangleq \ \wedge sorting = free \\
\wedge\ pe \in NOPARCELS \\
\wedge\ \exists\, p \in UNSORTED : \wedge\ pe' = p \\
\wedge\ \text{UNCHANGED}\ \langle channel,\ arrived, \\
\qquad\qquad sorted,\ sorting,\ ready\_to\_sort \rangle
\end{array}
$$

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 action *cross_parcel* occurs, the current parcel sorting is finished and then, of course, the current parcel becomes undefined.

$$
\begin{array}{l}
cross\_parcel \triangleq \ \wedge sorting = busy \\
\wedge\ arrived' = [arrived\ \text{EXCEPT}\ ![pe] = channel] \\
\wedge\ sorted' = sorted \cup \{pe\} \\
\wedge\ pe' \in NOPARCELS \wedge sorting' = free \\
\wedge\ \text{UNCHANGED}\ \langle channel,\ ready\_to\_sort \rangle
\end{array}
$$

The safe operating physical layout. From the existing (physical) layout, it is possible to build different controlled systems having different behaviors. It is natural to dismiss systems having unsafe behaviors. The controlled system which respects only these safety constraints, is called the *permissive* controlled system. In this automated system, *permissive* controller system actions are the *set* and *release*.

$$
\begin{array}{l}
set\_channel \triangleq \ \wedge sorting = free \wedge pe \notin NOPARCELS \\
\wedge\ ready\_to\_sort = \text{FALSE} \wedge channel' \in Baskets \\
\wedge\ ready\_to\_sort' = \text{TRUE} \\
\wedge\ \text{UNCHANGED}\ \langle arrived,\ sorted,\ pe,\ sorting \rangle
\end{array}
$$

Here, the value assigned to the variable *channel* is randomly determined. This justifies the *"permissive"* qualification. The guard fulfills a safety constraint in preventing a change of the channel when a parcel crosses the sorting device, preventing the parcel to be damaged.

$$
\begin{array}{l}
release \triangleq \ \wedge ready\_to\_sort = \text{TRUE} \\
\wedge\ sorting = free \wedge pe \in PARCELS \\
\wedge\ sorting' = busy \wedge ready\_to\_sort' = \text{FALSE} \\
\wedge\ \text{UNCHANGED}\ \langle channel,\ arrived,\ sorted,\ pe \rangle
\end{array}
$$

The first part of the guard ensures that a single parcel, at once, is being sorted, the second that a parcel may be selected before being released.

4.3 The permissive controlled system

The actions section of the TLA system modelling the permissive controlled system is obtained by merging the controller actions. The action *select_parcel* assigns the (next) current parcel to the variable *pe*. Once selected this parcel remains the current one as long as it has not reached a sorting basket. This captures a non explicit property of the feeder in which the parcels are stored in a first out order.

4.3.1 *Controllability*

The permissive controller behavior will likely not satisfy user's needs, in the sense that we can not certify that every parcel arrives in the right basket. Conversely, if we can verify that it is never the case that a parcel arrives in the expected basket, we assert that the system is not controllable with respect of the following requirement **P** and the development process stops here.

$$\forall\, p \in PARCELS : p \in \text{DOMAIN}\ arrived \rightsquigarrow arrived[p] = adr[p]$$

4.3.2 *The first abstract controlled system*

As far as we did not take into account the destination address of the parcel we can not expect a parcel to reach the right basket. The expected behavior of the automated system is defined in the requirement P. We add it to the invariant part of the TLA model. When attempting to verify the model, we expect a failure and use the report to find out what should be changed to satisfy the requirements. We deduce that it is necessary to change $channel' \in Baskets$ into $channel' = adr[pe]$ in the action *set_channel* in order to transform the permissive controlled system into an abstract model of the expected controlled system. The technique may seem less convincing as the case study at the level of abstraction is rather simple, but we claim its interest when we are faced to combinatorial situations. The parcel routing is a particular routing, obtained while making the control action more deterministic. At this abstract level, we have to prove the property asserting that any parcel which is presented in entry will arrive at the destination bin expressed by the following property:

$$\forall p.(p \in PARCELS \ \wedge \ p \in dom(arrived(p)) \ \Rightarrow \ arrived(p) = adr(p))$$

The abstract model of the automated system will be presented in the annexe A.

4.4 First refinement

In the proposed method, we refine in parallel each of the operative and the control parts adding new variables, properties, actions or redefining some of them.

Refinement of the operative part. In the abstract model, the physical device of the system is reduced to a single variable *channel*. A channel value corresponds to a setting of node switches. In order to represent a more concrete sorting device, we introduce the nodes *Node* as elements in an interval of integers, the level of the nodes in the sorting device tree structure (the root node corresponding to the level 0) *Level*, a function *adr_level* which gives at a given level the node to be visited to reach a given basket, a variable function *path_level* which associates to each level the node visited. The left successor of the node i is labelled $2*i+1$ and the right one is $2*i+2$. The root node is labelled 0. A pipe has the label of the node it "feeds". The leaves (terminal nodes) of the tree correspond to the baskets introduced in the abstract model.

The action *cross_parcel1* was extended by strengthening its guard with the condition $(l = n)$, meaning that the variable l has reached the baskets level and the action has been modified. *arrived(pe)* is assigned the value *adr_level[adr[pe], n]*, the (basket) node where the current parcel ends its run. The actions *cross_parcel* and *select_parcel* are redefined as follows :

$$\begin{array}{l} cross\_parcel1 \triangleq \ \wedge sorting = busy \wedge l = n \\ \wedge arrived' = [arrived \ \text{EXCEPT} \ ![pe] = adr\_level[adr[pe], \ n]] \\ \wedge sorted' = sorted \cup \{pe\} \wedge pe' \in NOPARCELS \\ \wedge sorting' = free \\ \wedge \text{UNCHANGED} \ \langle channel, \ ready\_to\_sort, \\ \qquad path\_level, \ i, \ l, \ ctrl\_cross \rangle \\ select\_parcel1 \triangleq \ \wedge select\_parcel \wedge i' = 0 \\ \wedge l' = 0 \wedge path\_level' = [path\_level \ \text{EXCEPT} \ ![0] = 0] \\ \wedge \text{UNCHANGED} \ \langle ctrl\_cross \rangle \end{array}$$

In the abstract model, the crossing of the current parcel concerned a single action *cross_parcel*, here we are concerned by the passing of the parcel through different nodes. Two nex actions *cross_right* and *cross_left* are introduced to determine the next node to be visited which mean updating the variable i representing the current node. The variable l representing the current level is updated as well. Updating the current node variable depends on the value of *path_level(l+1)*. If the value is odd the next node to reach is the left son otherwise it is the right son. The coding of the nodes makes the computation of the left or right son easy.

$$\begin{array}{l} cross\_left \triangleq \ \wedge ll < = n \wedge i \in Internal\_N \\ \wedge sorting = busy \wedge ctrl\_cross = \text{TRUE} \\ \wedge path\_level[l+1] = 2*i+1 \wedge l' = l+1 \\ \wedge i' = 2*i+1 \wedge ctrl\_cross' = \text{FALSE} \\ \wedge \text{UNCHANGED} \ \langle channel, \ sorting, \ pe, \ arrived, \\ \qquad sorted, \ ready\_to\_sort, \ path\_level \rangle \\ cross\_right \triangleq \ \wedge ll < = n-1 \wedge i \in Internal\_N \\ \wedge sorting = busy \wedge ctrl\_cross = \text{TRUE} \\ \wedge path\_level[l+1] = 2*i+2 \wedge l' = l+1 \\ \wedge i' = 2*i+2 \wedge ctrl\_cross' = \text{FALSE} \\ \wedge \text{UNCHANGED} \ \langle channel, \ sorting, \ ready\_to\_sort, \\ \qquad pe, \ arrived, \ sorted, \ path\_level \rangle \end{array}$$

The actions *select_parcel1*, *cross_left*, *cross_right* and *cross_parcel1* form the operative part of the automated system.

Refinement of the control part. In this refinement, we have added two new actions *path_right* and *path_left*. The actions *path_right*, *path_left*, *set_channel1* and *release1* form the control part of the automated system. The function *path_level* concerning the current parcel is updated "on the fly" by two actions : *path_right*, *path_left*. The *select_parcel* gives the opportunity to initialize *path_level(0)* with 0.

$path\_left \triangleq \ \wedge ii \in Internal\_N \wedge ll < = n$
$\wedge pe \in PARCELS \wedge sorting = busy \wedge ctrl\_cross = \text{FALSE}$
$\wedge adr\_level[adr[pe],\ l+1] = \ 2*i+1$
$\wedge path\_level' = [path\_level \text{ EXCEPT } ![l+1] = 2*i+1]$
$\wedge ctrl\_cross' = \text{TRUE}$
$\wedge \text{UNCHANGED } \langle channel,\ ready\_to\_sort,$
$sorting,\ pe,\ arrived,\ sorted,\ i,\ l\rangle$
$path\_right \triangleq \ \wedge ii \in Internal\_N \wedge ll < = n$
$\wedge pe \in PARCELS \wedge sorting = busy$
$\wedge ctrl\_cross = \text{FALSE}$
$\wedge adr\_level[adr[pe],\ l+1] = 2*i+2$
$\wedge path\_level' = [path\_level \text{ EXCEPT } ![l+1] = 2*i+2]$
$\wedge ctrl\_cross' = \text{TRUE}$
$\wedge \text{UNCHANGED } \langle channel, \text{ready\_to\_sort},$
$sorting,\ pe,\ arrived,\ sorted,\ i,\ l\rangle$

The automated system has to verify the following property :

$\forall p \in PARCELS : p \in \text{DOMAIN } arrived$

$\rightsquigarrow arrived[p] = adr\_level[adr[p],\ n]$

The automated system has been processed by the model checker TLC and all properties were verified by the model. The first refinement model of the automated system will be presented in the annexe B.

4.5 Second refinement

This refinement consists in adding details to the nodes. A node is provided with the following sensors : an input sensor, two outputs sensors and a gate which can be set to open the exit to the left pipe or to the right pipe.

Refinement of the operative part : an even more concrete model of the operative part. The position of the gates is represented by a function *gate* returning for each node a value in : {L, R}. The input sensor of a node is represented by a boolean variable *in* which is initially set to *false*. The output sensors are represented by the Boolean variables *out_l* and *out_r*. These variables become true when a parcel is detected and as we cannot have two outputs at the same time, the output sensors must verify the property $\neg\ (out\_l \wedge out\_r)$.

Two new actions were added to the operative part : *crossing_right* and *crossing_left* modelling the node crossing to go towards the left pipe, respectively right pipe, resulting from this node. The action *cross_parcel1*, concerning the operative part is refined by adding to the level of each node the sensor which detects the entry of parcels.

$crossing\_left \triangleq \ \wedge i \in Internal\_N$
$\wedge in[i] = \text{TRUE} \wedge gate[i] = L$
$\wedge out\_l' = [out\_l \text{ EXCEPT } ![i] = \text{TRUE}]$
$\wedge out\_r' = [out\_r \text{ EXCEPT } ![i] = \text{FALSE}]$
$\wedge in' = [in \text{ EXCEPT } ![i] = \text{FALSE}]$
$\wedge \text{UNCHANGED } \langle channel,\ pe,\ arrived,\ sorted,\ sorting,$
$ready\_to\_sort,\ i,\ l,\ path\_level,\ ctrl\_cross,\ gate,\ gate\_set\rangle$
$cross\_left2 \triangleq \ \wedge cross\_left$
$\wedge (gate\_set = \text{TRUE} \vee l \geq n-2) \wedge out\_l[i] = \text{TRUE}$
$\wedge in' = [in \text{ EXCEPT } ![2*i+1] = \text{TRUE}]$
$\wedge gate\_set' = \text{FALSE} \wedge \text{UNCHANGED } \langle out\_l,\ out\_r,\ gate\rangle$
$crossing\_right \triangleq \ \wedge i \in Internal\_N$
$\wedge in[i] = \text{TRUE} \wedge gate[i] = R$
$\wedge out\_l' = [out\_l \text{ EXCEPT } ![i] = \text{FALSE}]$
$\wedge out\_r' = [out\_r \text{ EXCEPT } ![ii] = \text{TRUE}]$
$\wedge in' = [in \text{ EXCEPT } ![i] = \text{FALSE}]$
$\wedge \text{UNCHANGED } \langle channel,\ pe,\ arrived,\ sorted,\ sorting,$
$ready\_to\_sort,\ i,\ l,\ gate,\ path\_level,\ ctrl\_cross,\ gate\_set\rangle$
$cross\_right2 \triangleq \ \wedge cross\_right$
$\wedge (gate\_set = \text{TRUE} \ \vee ll \geq n-2) \wedge out\_r[i] = \text{TRUE}$
$\wedge in' = [in \text{ EXCEPT } ![2*i+2] = \text{TRUE}]$
$\wedge gate\_set' = \text{FALSE} \wedge \text{UNCHANGED } \langle out\_l,\ out\_r,\ gate\rangle$

The actions *cross_right2*, respectively *cross_left2* model the pipe crossing, indicating that the parcel passed towards the next left node, respectively towards the right following node. The variable *gate_set*, is used to guarantee that the parcel crossing is possible only if the node gate is positioned in advance. It is assigned the value TRUE by the actions *cross_left2* and FALSE by the action *cross_right2*.

Refinement of the control part. Two new actions were added to refine control part,*set_gate_right* and *set_gate_left*. The action *set_gate_left* model gate opening on the left, by putting *gate(adr_level(adr(pe), ll+2)):=L*, where *adr_level(adr(pe), ll+2)* indicates the current node. The same behavior but towards right side is modelled by the action *set_gate_right*.

$set\_gate\_left \triangleq \ \wedge ll < = n-1 \wedge pe \in PARCELS$
$\wedge adr\_level[adr[pe],\ l+1] = 2*adr\_level[adr[pe],\ l]+1$
$\wedge gate\_set' = \text{TRUE}$
$\wedge gate' = [gate \text{ EXCEPT } ![adr\_level[adr[pe],\ l]] = L]$
$\wedge \text{UNCHANGED } \langle channel,\ pe,\ arrived,\ sorted,\ sorting,$
$ready\_to\_sort,\ i,\ l,\ path\_level,\ ctrl\_cross,\ in,$
$out\_l,\ out\_r\rangle$
$set\_gate\_right \triangleq \ \wedge ll < = n-1 \wedge pe \in PARCELS$
$\wedge adr\_level[adr[pe],\ l+1] = 2*adr\_level[adr[pe],\ l]+2$
$\wedge gate\_set' = \text{TRUE}$
$\wedge gate' = [gate \text{ EXCEPT } ![adr\_level[adr[pe],\ l]] = R]$
$\wedge \text{UNCHANGED } \langle channel,\ pe,\ arrived,\ sorted,\ sorting,$
$ready\_to\_sort,\ i,\ l,\ path\_level,\ ctrl\_cross,\ in,$
$out\_l,\ out\_r\rangle$

The actions *path_left2*, respectively *path_right2* having for effect to calculate, the next node to be visited by checking that the gate is opened in the good orientation *(gate(i) = L (left), gate(i)= R (right))*. The

action *select_parcel1*, is also refined en *select_parcel2* by taking into account the sensor which models the parcel presence on the level of the first entry basket.

$$
\begin{array}{l}
path\_left2 \triangleq \ \wedge path\_left \\
\wedge\ in[i] = \text{TRUE} \wedge gate[i] = L \\
\wedge\ \text{UNCHANGED}\ \langle in,\ out\_l,\ out\_r,\ gate,\ gate\_set\rangle \\
path\_right2 \triangleq \ \wedge path\_right \\
\wedge\ in[i] = \text{TRUE} \wedge gate[i] = R \\
\wedge\ \text{UNCHANGED}\ \langle in,\ out\_l,\ out\_r,\ gate,\ gate\_set\rangle
\end{array}
$$

Once the parcel is in a node, three actions can be started consecutively, the first is *set_gate_left*, which models gate opening on the left. The same behavior but towards right side is modelled by the action *set_gate_right*. The second executed action is *crossing_left*, respectively *crossing_right*, modelling the node crossing to go towards the left pipe, respectively right pipe, resulting from this node. The third action concerns the pipe crossing and it is modelled by *cross_right2*, respectively *cross_left2*.

The automated system has been processed by the model checker TLC and all properties were verified by the model. These properties are fairness asserting that each action able to be executed is activated at least one time and safety represented as invariant and asserting that we cannot have two outputs at the same time ($\neg\ (out\_l \wedge out\_r)$). The second refinement model of the automated system will be presented in the annexe C. As our goal in the proposed method is to develop a control system, we separate what concerns the operative component from what concerns the controller in order to build a program of the controller,

5 Conclusion

In this paper, we have proposed the first steps of a formal method for the development of automated systems using the temporal logic of actions TLA+. The main contribution of the paper consists in building a model of the expected automated system starting from an abstract model of the operative part. In the stepwise approach, the first model consists of an abstract model of the operative part with the guards of the action kept as week as possible to prevent unsafe behaviors. The only property one can expect from a system, we have called permissive, is that it does not prevent the expected behaviour of the final automated system to occur. If it is the case, we deduce that the system is controllable and that is is possible to strengthen the guards in order to obtain an abstract model of the required controlled system. The controlled system is then refined in the usual way. The refinement, indeed, concerns the operative part and the control part. At the end of the process, we have a detailed model of the automated system. The validation of the operative part, using the animator has been experienced in another work. The proposed method has been illustrated by the example of a parcel sorting system and our approach has simplified the specification of an automated system which meets desired properties.

We have specified also this example with the event based B approach (Abrial 1996b, Abrial 1996a, Abrial & Mussat 1998) and we have verified all proof obligations with the powerful tool AtelierB. In the event based B approach, only invariance and safety properties are considered but fairness and eventuality properties are not considered. The temporal Logic of Actions TLA+, in the other hand, expresses well eventuality and fairness and supports refinement and this is why we have used it in our method. In future works we will develop a method using in the same approach the B method and TLA+ to take benefits of the powerful tool of B and to formulate in TLA+ more natural properties that are not straightforward to express with B and to verify them with the model checker TLC.

References

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

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

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

JARAY, J. & A.Mahjoub (1994). Développement formel et incrémental de systèmes de commande de procédés industriels inspiré de l'approche unity, *Informatique répartie : Etat de l'art et Perspectives. JISI'94*, pp. 41–58.

Lamport, L. (1991). The Temporal Logic of Actions, *Technical Report 79*, Digital Equipment Corporation, Systems Research Centre.

Lamport, L. (1993). Hybrid systems in TLA$^+$, *in* R. L. Grossman, A. Nerode, A. P. Ravn & H. Rischel (eds), *Hybrid Systems*, Vol. 736 of

Lecture Notes in Computer Science, Springer-Verlag, pp. 77–102.

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

Lamport, L. (1998). The module structure of TLA+, *Technical Report SRC-TN-1996-002A*, Hewlett Packard Laboratories.

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

Ramadge, P. J. G. & Wonham, W. M. (1989). The control of discrete event systems, *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems* **77, 1**: 81–98.

Annexe A

MODULE *Parcel_Sorting*

EXTENDS *Naturals*

CONSTANTS *PPARCELS*, *PARCELS*, *Baskets*, *noBaskets*, *adr*, *free*, *busy*

VARIABLES *channel*, *sorting*, *pe*, *sorted*, *arrived*, *ready_to_sort*

ASSUME *adr* $\in [PARCELS \to Baskets] \land PARCELS \neq \{\} \land Baskets \neq \{\}$

$\land$ *PARCELS* $\subseteq$ *PPARCELS* $\land$ *PARCELS* $\neq$ *PPARCELS*

$\land$ *noBaskets* $\notin$ *Baskets*

SortingState $\triangleq$ {*free*, *busy*}

TypeInvariant $\triangleq$ $\land$ *channel* $\in$ *Baskets* $\cup$ {*noBaskets*}

$\land$ *sorting* $\in$ *SortingState* $\land$ *pe* $\in$ *PPARCELS* $\land$ *ready_to_sort* $\in$ BOOLEAN

$\land$ *arrived* $\in [PARCELS \to Baskets] \land$ sorted $\subseteq$ PARCELS

UNSORTED $\triangleq$ *PARCELS* $\setminus$ *sorted*

NOPARCELS $\triangleq$ *PPARCELS* $\setminus$ *PARCELS*

Init $\triangleq$ $\land$ *channel* = *noBaskets* $\land$ *sorting* = *free* $\land$ *pe* $\in$ *NOPARCELS*

$\land$ *arrived* $= [p \in PARCELS \mapsto noBaskets] \land$ *sorted* $= \{\} \land$ *ready_to_sort* = FALSE

select_parcel $\triangleq$ $\land$ *sorting* = *free*

$\land$ *pe* $\in$ *NOPARCELS* $\land \exists\, p \in UNSORTED : \land pe' = p$

$\land$ UNCHANGED $\langle$*channel*, *arrived*, *sorted*, *sorting*, *ready_to_sort*$\rangle$

set_channel $\triangleq$ $\land$ *sorting* = *free*

$\land$ *pe* $\notin$ *NOPARCELS* $\land$ *ready_to_sort* = FALSE $\land$ *channel*$'$ = *adr*[*pe*]

$\land$ *ready_to_sort*$'$ = TRUE $\land$ UNCHANGED $\langle$*arrived*, *sorted*, *pe*, *sorting*$\rangle$

release $\triangleq$ $\land$ *ready_to_sort* = TRUE

$\land$ *sorting* = *free* $\land$ *pe* $\in$ *PARCELS* $\land$ *sorting*$'$ = *busy*

$\land$ *ready_to_sort*$'$ = FALSE $\land$ UNCHANGED $\langle$*channel*, *arrived*, *sorted*, *pe*$\rangle$

cross_parcel $\triangleq$ $\land$ *sorting* = *busy*

$\land$ *arrived*$'$ = [*arrived* EXCEPT ![*pe*] = *channel*]

$\land$ *sorted*$'$ = *sorted* $\cup$ {*pe*} $\land$ *pe*$'$ $\in$ *NOPARCELS* $\land$ *sorting*$'$ = *free*

$\land$ UNCHANGED $\langle$*channel*, *ready_to_sort*$\rangle$

Next $\triangleq$ $\lor$ *select_parcel* $\lor$ *set_channel* $\lor$ *release* $\lor$ *cross_parcel*

---

trvars $\triangleq$ $\langle$*channel*, *sorting*, *pe*, *arrived*, *sorted*, *ready_to_sort*$\rangle$

TrFairness $\triangleq$ $\mathrm{WF}_{trvars}(Next)$

Spec $\triangleq$ *Init* $\land \Box[Next]_{trvars} \land$ *TrFairness*

Invariant $\triangleq$ $\land$ *sorting* = *busy* $\Rightarrow$ *channel* = *adr*[*pe*]

$\land$ *sorting* $\neq$ *free* $\Rightarrow$ *pe* $\in$ *PARCELS* $\land$ *sorting* = *busy* $\Rightarrow$ *ready_to_sort* = FALSE

$\land$ *ready_to_sort* = TRUE $\Rightarrow$ *channel* = *adr*[*pe*]

$\land$ *ready_to_sort* = TRUE $\Rightarrow$ *pe* $\in$ *PARCELS*

Liveness $\triangleq$ $\land$ $\forall\, p \in PARCELS : p \in$ DOMAIN *arrived* $\rightsquigarrow$ *arrived*[*p*] = *adr*[*p*]

$\land \forall\, p \in PARCELS : p \in$ DOMAIN *arrived* $\rightsquigarrow \Diamond$ *arrived*[*p*] $\in$ *Baskets*

THEOREM *Spec* $\Rightarrow \Box$ *Init* THEOREM *Spec* $\Rightarrow \Box$ *TypeInvariant*

THEOREM *Spec* $\Rightarrow \Box$ *Invariant* THEOREM *Spec* $\Rightarrow$ *Liveness*

Annexe B

MODULE *Trieuse1*

EXTENDS *Trieuse*, *TLC*

CONSTANTS *adr_level*, *n*, *Level*, *Node*, *noNode*, *Internal_N*

VARIABLES *path_level*, *i*, *l*, *ctrl_cross*

ASSUME *Level* $\subseteq$ *Nat* $\land$ *Level* $\neq$ *Nat* $\land$ *adr_level* $\in [(Baskets * Level) \to Node]$

$\land$ *Baskets* $\subseteq$ *Node* $\land$ *Baskets* $\neq$ *Node* $\land$ *Node* $\neq \{\} \land$ *Internal_N* $\subseteq$ *Node*

$\land$ *Internal_N* $\neq$ *Node* $\land$ *noNode* $\notin$ *Node*

$\land \quad \forall\, p \in PARCELS : adr\_level[adr[p], n] = adr[p]$

TypeInvariant1 $\triangleq$ $\land$ *TypeInvariant* $\land$ *path_level* $\in [Level \to Node \cup \{noNode\}]$

$\land$ *ii* $\in$ *Nat* $\land$ *ii* $\in$ *Node* $\land$ *ll* $\in$ *Nat* $\land$ *ll* $\in$ *Level* $\land$ *ctrl_cross* $\in$ BOOLEAN

Init1 $\triangleq$ *Init* $\land$ *i* = 0 $\land$ *ll* = 0

$\land$ *path_level* $= [i \in Level \mapsto noNode] \land$ *ctrl_cross* = FALSE

select_parcel1 $\triangleq$ $\land$ *select_parcel* $\land$ *i*$'$ = 0 $\land$ *l*$'$ = 0

$\land$ *path_level*$'$ = [*path_level* EXCEPT ![0] = 0] $\land$ UNCHANGED $\langle$*ctrl_cross*$\rangle$

set_channel1 $\triangleq$ $\land$ *set_channel* $\land$ UNCHANGED $\langle$*path_level*, *i*, *l*, *ctrl_cross*$\rangle$

release1 $\triangleq$ $\land$ *release* $\land$ UNCHANGED $\langle$*i*, *l*, *path_level*, *ctrl_cross*$\rangle$

path_left $\triangleq$ $\land$ *pe* $\in$ *PARCELS* $\land$ *sorting* = *busy* $\land$ *ctrl_cross* = FALSE

$\land$ *adr_level*[*adr*[*pe*], *l* + 1] = 2 $*$ *i* + 1 $\land$ *ii* $\in$ *Internal_N*

$\land$ *path_level*$'$ = [*path_level* EXCEPT ![*l* + 1] = 2 $*$ *i* + 1] $\land$ *ctrl_cross*$'$ = TRUE $\land$ *ll* $<=$ *n*

$\land$ UNCHANGED $\langle$*channel*, *ready_to_sort*, *sorting*, *pe*, *arrived*, *sorted*, *i*, *l*$\rangle$

cross_left $\triangleq$ $\land$ *ll* $<=$ *n* $\land$ *i* $\in$ *Internal_N*

$\land$ *sorting* = *busy* $\land$ *ctrl_cross* = TRUE $\land$ *path_level*[*l* + 1] = 2 $*$ *i* + 1

$\land$ *l*$'$ = *l* + 1 $\land$ *i*$'$ = 2 $*$ *i* + 1 $\land$ *ctrl_cross*$'$ = FALSE

$\land$ UNCHANGED $\langle$*channel*, *sorting*, *pe*, *arrived*, *sorted*, *ready_to_sort*, *path_level*$\rangle$

path_right $\triangleq$ $\land$ *ii* $\in$ *Internal_N* $\land$ *ll* $<=$ *n*

$\wedge\ pe \in PARCELS \wedge sorting = busy \wedge ctrl\_cross = \text{FALSE}$

$\wedge\ adr\_level[adr[pe],\ l+1] = 2*i+2$

$\wedge\ path\_level' = [path\_level \text{ EXCEPT } ![l+1] = 2*i+2] \wedge ctrl\_cross' = \text{TRUE}$

$\wedge\ \text{UNCHANGED } \langle channel,\ ready\_to\_sort,\ sorting,\ pe,\ arrived,\ sorted,\ i,\ l\rangle$

$cross\_right \triangleq \wedge\ ll < = n-1 \wedge i \in Internal\_N \wedge sorting = busy$

$\wedge\ ctrl\_cross = \text{TRUE} \wedge path\_level[l+1] = 2*i+2 \wedge l' = l+1$

$\wedge\ i' = 2*i+2 \wedge ctrl\_cross' = \text{FALSE}$

$\wedge\ \text{UNCHANGED } \langle channel,\ sorting,\ ready\_to\_sort,\ pe,\ arrived,\ sorted,\ path\_level\rangle$

$cross\_parcel1 \triangleq \wedge\ sorting = busy \wedge l = n$

$\wedge\ arrived' = [arrived \text{ EXCEPT } ![pe] = adr\_level[adr[pe],\ n]]$

$\wedge\ sorted' = sorted \cup \{pe\} \wedge pe' \in NOPARCELS \wedge sorting' = free$

$\wedge\ \text{UNCHANGED } \langle channel,\ ready\_to\_sort,\ path\_level,\ i,\ l,\ ctrl\_cross\rangle$

$Next1 \triangleq \vee select\_parcel1 \vee set\_channel1 \vee release1 \vee path\_left$

$\vee cross\_left \vee path\_right \vee cross\_right \vee cross\_parcel1$

$tr1vars \triangleq \langle channel,\ sorting,\ pe,\ arrived,\ sorted,\ ready\_to\_sort,\ path\_level$

$,\ i,\ l,\ ctrl\_cross\rangle$

$Tr1Fairness \triangleq \text{WF}_{tr1vars}(Next1)$

$Spec1 \triangleq Init1 \wedge \Box[Next1]_{tr1vars} \wedge Tr1Fairness$

$Liveness1 \triangleq \wedge\ \forall\, p \in PARCELS : p \in \text{DOMAIN } arrived$

$\rightsquigarrow arrived[p] = adr\_level[adr[p],\ n]$

THEOREM $Spec1 \Rightarrow Spec$ THEOREM $Spec1 \Rightarrow \Box Init1$

THEOREM $Spec1 \Rightarrow \Box TypeInvariant1$ THEOREM $Spec1 \Rightarrow \Box Liveness1$

Annexe C

MODULE $Trieuse2$

EXTENDS $Trieuse1,\ TLC$

CONSTANTS $L,\ R,\ indef$

VARIABLES $in,\ out\_l,\ out\_r,\ gate,\ gate\_set$ $Result \triangleq \{L,\ R,\ indef\}$

$TypeInvariant2 \triangleq \wedge\ TypeInvariant1 \wedge gate\_set \in \text{BOOLEAN}$

$\wedge\ in \in [Node \rightarrow \text{BOOLEAN}\ ] \wedge out\_l \in [Internal\_N \rightarrow \text{BOOLEAN}\ ]$

$\wedge\ out\_r \in [Internal\_N \rightarrow \text{BOOLEAN}\ ] \wedge gate \in [Node \rightarrow Result]$

$Init2 \triangleq Init1 \wedge in = [i \in Node \mapsto \text{FALSE}] \wedge out\_l = [i \in Internal\_N \mapsto \text{FALSE}]$

$\wedge\ out\_r = [i \in Internal\_N \mapsto \text{FALSE}] \wedge gate = [i \in Node \mapsto indef] \wedge gate\_set = \text{FALSE}$

$select\_parcel2 \triangleq \wedge\ select\_parcel1$

$\wedge\ in' = [in \text{ EXCEPT } ![0] = \text{TRUE}] \wedge \text{UNCHANGED } \langle out\_l,\ out\_r,\ gate,\ gate\_set\rangle$

$set\_channel2 \triangleq \wedge\ set\_channel1$

$\wedge\ \text{UNCHANGED } \langle out\_l,\ out\_r,\ gate,\ gate\_set,\ in\rangle$

$release2 \triangleq \wedge\ release1 \wedge \text{UNCHANGED } \langle out\_l,\ out\_r,\ gate,\ in,\ gate\_set\rangle$

$set\_gate\_left \triangleq \wedge\ ll < = n-1 \wedge pe \in PARCELS$

$\wedge\ adr\_level[adr[pe],\ l+1] = 2 * adr\_level[adr[pe],\ l] + 1$

$\wedge\ gate\_set' = \text{TRUE} \wedge gate' = [gate \text{ EXCEPT } ![adr\_level[adr[pe],\ l]] = L]$

$\wedge\ \text{UNCHANGED } \langle channel,\ pe,\ arrived,\ sorted,\ sorting,\ i,\ l,$

$ready\_to\_sort,\ path\_level,\ ctrl\_cross,\ in,\ out\_l,\ out\_r\rangle$

$path\_left2 \triangleq \wedge\ path\_left \wedge in[i] = \text{TRUE} \wedge gate[i] = L$

$\wedge\ \text{UNCHANGED } \langle in,\ out\_l,\ out\_r,\ gate,\ gate\_set\rangle$

$crossing\_left \triangleq \wedge\ i \in Internal\_N \wedge in[i] = \text{TRUE} \wedge gate[i] = L$

$\wedge\ out\_l' = [out\_l \text{ EXCEPT } ![i] = \text{TRUE}] \wedge out\_r' = [out\_r \text{ EXCEPT } ![i] = \text{FALSE}]$

$\wedge\ in' = [in \text{ EXCEPT } ![i] = \text{FALSE}] \wedge \text{UNCHANGED } \langle channel,\ pe,\ arrived,\ sorted,\ sorting,\ i,\ l,$

$ready\_to\_sort,\ path\_level,\ ctrl\_cross,\ gate,\ gate\_set\rangle$

$cross\_left2 \triangleq \wedge\ cross\_left \wedge (gate\_set = \text{TRUE} \vee l \geq n-2)$

$\wedge\ out\_l[i] = \text{TRUE} \wedge in' = [in \text{ EXCEPT } ![2*i+1] = \text{TRUE}]$

$\wedge\ gate\_set' = \text{FALSE} \wedge \text{UNCHANGED } \langle out\_l,\ out\_r,\ gate\rangle$

$set\_gate\_right \triangleq \wedge\ ll < = n-1 \wedge pe \in PARCELS$

$\wedge\ adr\_level[adr[pe],\ l+1] = 2 * adr\_level[adr[pe],\ l] + 2 \wedge gate\_set' = \text{TRUE}$

$\wedge\ gate' = [gate \text{ EXCEPT } ![adr\_level[adr[pe],\ l]] = R]$

$\wedge\ \text{UNCHANGED } \langle channel,\ pe,\ arrived,\ sorted,\ sorting,\ ready\_to\_sort,\ i,\ l,\ path\_level,$

$ctrl\_cross,\ in,\ out\_l,\ out\_r\rangle$

$path\_right2 \triangleq \wedge\ path\_right \wedge in[i] = \text{TRUE} \wedge gate[i] = R$

$\wedge\ \text{UNCHANGED } \langle in,\ out\_l,\ out\_r,\ gate,\ gate\_set\rangle$

$crossing\_right \triangleq \wedge\ i \in Internal\_N \wedge in[i] = \text{TRUE} \wedge gate[i] = R$

$\wedge\ out\_l' = [out\_l \text{ EXCEPT } ![i] = \text{FALSE}] \wedge out\_r' = [out\_r \text{ EXCEPT } ![ii] = \text{TRUE}]$

$\wedge\ in' = [in \text{ EXCEPT } ![i] = \text{FALSE}] \wedge \text{UNCHANGED } \langle channel,\ pe,\ arrived,\ sorted,\ sorting$

$,\ ready\_to\_sort,\ i,\ l,\ gate\ path\_level,\ ctrl\_cross,\ gate\_set\rangle$

$cross\_right2 \triangleq \wedge\ cross\_right \wedge (gate\_set = \text{TRUE} \vee ll \geq n-2)$

$\wedge\ out\_r[i] = \text{TRUE} \wedge in' = [in \text{ EXCEPT } ![2*i+2] = \text{TRUE}]$

$\wedge\ gate\_set' = \text{FALSE} \wedge \text{UNCHANGED } \langle out\_l,\ out\_r,\ gate\rangle$

$cross\_parcel2 \triangleq \wedge\ cross\_parcel1$

$\wedge\ in' = [in \text{ EXCEPT } ![channel] = \text{TRUE}] \wedge \text{UNCHANGED } \langle out\_l,\ out\_r,\ gate,\ gate\_set\rangle$

$Next2 \triangleq \vee select\_parcel2 \vee set\_channel2 \vee release2 \vee cross\_parcel2$

$\vee path\_left2 \vee path\_right2 \vee cross\_left2 \vee cross\_right2 \vee crossing\_left$

$\vee crossing\_right \vee set\_gate\_left \vee set\_gate\_right$

$tr2vars \triangleq \langle channel,\ sorting,\ pe,\ arrived,\ sorted,\ ready\_to\_sort,$

$path\_level,\ i,\ l,\ ctrl\_cross,\ in,\ out\_l,\ out\_r,\ gate,\ gate\_set\rangle$

$Tr2Fairness \triangleq \text{WF}_{tr2vars}(Next2)$

$Spec2 \triangleq Init2 \wedge \Box[Next2]_{tr2vars} \wedge Tr2Fairness$

$Invariant2 \triangleq \forall\, jj \in Internal\_N : \neg(out\_r[jj] \wedge out\_l[jj])$

THEOREM $Spec2 \Rightarrow Spec1$ THEOREM $Spec2 \Rightarrow \Box TypeInvariant2$

THEOREM $Spec2 \Rightarrow \Box Init2$ THEOREM $Spec2 \Rightarrow Invariant2$