Reformulation: a Way to Combine Dynamic Properties and *B* Refinement

F. Bellegarde, C. Darlot, J. Julliand, and O. Kouchnarenko

Laboratoire d'Informatique de l'Université de Franche-Comté
16, route de Gray, 25030 Besançon Cedex
Ph:(33) 3 81 66 64 52, Fax:(33) 3 81 66 64 50
{bellegar,darlot,julliand,kouchna}@lifc.univ-fcomte.fr,
Web : http://lifc.univ-fcomte.fr

Abstract. We are interested in verifying dynamic properties of reactive systems. The reactive systems are specified by B event systems in a refinement development. The refinement allows us to combine proof and model-checking verification techniques in a novel way.
Most of the *PLTL* dynamic properties are preserved by refinement, but in our approach, the user can also express how a property evolves during the refinement. The preservation of the abstract property, expressed by a *PLTL* formula F_1, is used as an assumption for proving a *PLTL* formula F_2 which expresses an enriched property in the refined system. Formula F_1 is verified by model-checking on the abstract system. So, to verify the enriched formula F_2, it is enough to prove some propositions depending on the respective patterns followed by F_1 and F_2. In this paper, we show how to obtain these *sufficient* propositions from the refinement relation and the semantics of the *PLTL* formulae. The main advantage is that the user does not need to express a variant or a loop invariant to obtain automatic proofs of dynamic properties, at least for finite state event systems. Another advantage is that the model-checking is done on an abstraction with few states.
Keywords: Verification of *PLTL* properties, Combination of proof and model-checking, Refinement development.

1 Introduction

Most properties of reactive systems [12, 2, 10] are dynamic. In our approach for the design and verification, reactive systems are expressed by *B* finite state event systems and their dynamic properties are formulated in the Propositional Linear Temporal Logic [12] (*PLTL*). Recall that the *B* method is essentially a refinement design.

Our methodological approach as well as our verification techniques for addressing the introduction of dynamic constraints in *B* (see Fig. 1) is quite different from the propositions of J.–R. Abrial and L. Mussat in [2]. The *B* event systems can be associated with finite or infinite transition systems. In *B*, the verification of invariants, *dynamic invariants* and of liveness modalities *leads to* and *until* uses a proof technique which requires explicit *variants* and *loop invariants* from the users. *Variants* and *loop invariants* are needed for the verification of the liveness modalities. Moreover, a global *variant* is necessary for proving that the refinement introduces no livelock. Our proposals deal only with the specification and the verification of finite state systems but the specification and the verification of any *PLTL* property (safety, liveness, fairness) is possible.

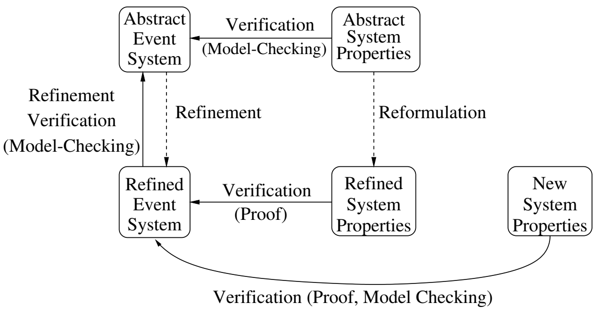

Fig. 1. Specification and verification approach

They differ from [2] in the four essential following points.

1. The dynamic properties are expressed in *PLTL*.
2. As for the events, the dynamic properties that are introduced at the abstract level can be enriched through the refinement. Then, they are formulated anew at the refined level. We say that the formula is *reformulated*.
3. The abstract dynamic properties are model-checked and their *reformulations* are verified by proof.

The motivation behind these propositions is threefold. First and above all, we want to set the user free from looking for a *variant* and a *loop invariant* when expressing dynamic properties. Second, we want to be able to use model-checking and proof for the verification in a way which utilizes them at their best. Third, the user can express its modalities freely using the expressive power of the *PLTL* logic.

***B* + *PLTL* versus *B* + *Modalities* (see Point 1 above).** In [2], J.–R. Abrial and L. Mussat use three patterns of dynamic properties: the *dynamic invariant* and the two modalities *leads to* and *until*. If p and q are propositional predicates, the *dynamic invariants* have the same expressive power as the set of *PLTL* formulae $\Box(p \Rightarrow \bigcirc q)$. The modalities have the same expressive power as a fragment of the *PLTL* using the two kinds of properties $\Box(p \Rightarrow \Diamond q)$, and $\Box(p \Rightarrow (p \mathcal{U} q))$. Moreover, besides the instances for p and q, the user has to specify a *variant*, often a *loop invariant*, and a list of the events which may be taken during the loop.

We use the *PLTL* future operators *Always*, *Next*, *Eventually*, and *Until* (denoted respectively by $\Box$, $\bigcirc$, $\Diamond$, and $\mathcal{U}$) with the following meaning. For a path σ,

- $\Box p$ means that the property p holds in each state of σ;
- $\bigcirc p$ means that the property p holds in the next state;
- $\Diamond p$ means that there exists a state in the future which satisfies p;
- $p \mathcal{U} q$ means that p holds until q eventually happens.

Model-Checking and Proof (see Points 3 above). We choose the *PLTL* logic because its verification can be done by *PLTL* model-checking [8] which is entirely automatic for the totality of the logic. The main drawback is that it cannot handle very large systems and, worse yet, infinite state systems. A solution for large finite systems may consist of using jointly proof and model-checking. So, the model-checking explosion is avoided as well as the requirement consisting in providing clues such as *variants* and *loop invariants* to a theorem prover. To better explain how we propose to join both techniques to verify the reformulated properties consider Fig. 1.

First, the user specifies the abstract event system with its invariant and its dynamic properties expressed in *PLTL*. The invariant is proof-checked as in *B*. The dynamic properties are model-checked on the event system operational model, i.e., on the set of paths of a finite state transition system with a small number of states.

Second, the user specifies its refinements introducing new variables. The relation between the set of states S_2 of the refined system and the set of states S_1 of the abstract system is expressed by a *gluing invariant*. New events are introduced, old events are formulated once more, new *PLTL* formulae are introduced, and old *PLTL* formulae are formulated anew.

We do not want to use the *PLTL* model-checking again for the verification of the reformulated properties. So, we propose to use proof techniques, but without requiring a *loop invariant* and of a *variant*. In the paper, we present two kinds of propositions which are associated systematically according to the shapes (called *refinement patterns*) of the abstract *PLTL* formula and the refined formula. The first kind—a weak form, includes propositional sub-formulae and the invariants of the event systems. When they are valid, we know that the refined formula holds. The failure does not mean that the refined formula does not hold. So, the second kind—a strong form, includes either the old or the new events. Again, the success means that the refined formula holds but, from a failure, we cannot conclude. Therefore, these propositions are *sufficient conditions* and not *proof obligations*.

In the paper, we show that if these propositions (weak or strong) are valid, then the reformulated properties hold without the help of neither an user-given *variant* nor a *loop invariant*.

Reformulation versus Preservation (see Point 2 above). In [2], the modalities *leads to* and *until* which hold on the abstract paths are preserved on the refined paths. However, *dynamic invariants*, which could be expressed by an instance of the *PLTL* pattern $\Box(p \Rightarrow \bigcirc q)$, are not preserved on the refined paths since new events may not verify the *dynamic invariant*. Moreover, because the new events are interwoven among the old ones, the refined system does not satisfy the pattern $\Box(p \Rightarrow \bigcirc q)$ but it satisfies an instance of the *PLTL* pattern $\Box(p \Rightarrow \Diamond q)$—a weaker formula. More generally, the preservation technique does not allow the user to indicate how the new events are interwoven among the old events. However, the reformulation can do it. For example, the reformulation of the *PLTL* pattern $\Box(p_1 \Rightarrow \bigcirc q_1)$ by the *PLTL* pattern $\Box(p_2 \Rightarrow \bigcirc q_2)$, allows us to specify that it is forbidden to introduce some of the new events before the old events which are enabled when p_1 holds. The reformulation of the pattern $\Box(p_1 \Rightarrow \bigcirc q_1)$ by the pattern $\Box(p_2 \Rightarrow (r_2 \mathcal{U} q_2))$ allows introducing only the new events which maintain r_2. So, the purpose of the reformulation of a property is that the formula of the refined property specifies explicitly how the new events can be interwoven among the old events. Therefore, the reformulated formula may be richer than the preserved formula. The effect of a reformulated formula compares with the effect of a gluing invariant in the following manner. A *gluing invariant* specifies a relation between the refined and the abstract states whereas a reformulated formula, together with the *gluing invariant*, specifies a relation between the refined and the abstract paths of the operational model. It is redundant with the expression of the events but we think that it is important that the design allows such redundancies in a specification so that the verification can exploit them.

Paper Organization. The paper is organized as follows. Section 2 illustrates our approach on an example. After a short presentation of our refinement relation, we explain how to verify the *reformulated* dynamic properties through refinement in Section 3. Finally, we situate our concerns and give some ideas about future works in Section 4.

2 Example

In this section, we introduce a robot as an example. We will examine the operational specification, and then, we will express some dynamic properties to be verified on this system.

2.1 Operational Description

Figure 2 shows the physical system. The robot must move some parts from the *arrival device* (called *AD*) to one of the *exit devices* (called resp. *LED* and *RED* for the left and right exit devices) using the *carrier device* (called *CD*).

Here, we show the abstract specification as well as two further levels of refinement called *first refinement* and *second refinement*. We express the specifications using a *B* event system syntax extended with *PLTL*. Notice that the variables are annotated with a number corresponding to the level of refinement (here: 0, 1 and 2).

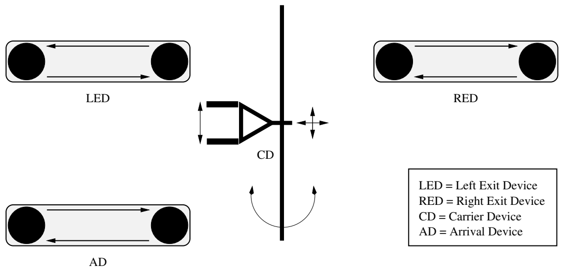

Fig. 2. Physical system

Abstract Specification. The very abstract specification only formalizes the transportation of the parts ignoring the arrival and exit devices and the carrier device movements which will be considered in further refinements. In other words, we only observe the carrier device state. Figure 3 gives the operational semantics of the abstract level specification described as the event system below. There are two events:

- *Load*: The carrier device takes a part, then it is busy (value b);
- *Unload*: The carrier device drops a part, then it is free (value f).

EVENT SYSTEM ROBOT$_0$
SETS: DEVICE_STATE=$\{$f,b$\}$
VARIABLES: CD$_0$
INVARIANT: CD$_0$ $\in$ DEVICE_STATE
INITIALIZATION: CD$_0$:=f
EVENTS:
Load $\hat{=}$**SELECT** CD$_0$=f **THEN** CD$_0$:=b **END**;
Unload $\hat{=}$**SELECT** CD$_0$=b **THEN** CD$_0$:=f **END**
END

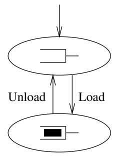

Fig. 3. Transition system for the abstract level

First Refinement. We consider now the left and right exit devices. The variables of the abstract and the refined specifications are linked together by a *gluing invariant*. We observe two new events (*LEvac* and *REvac*) which set the exit devices free. These events can happen whenever the exit devices are busy. In the refined specification, the old events keep the same labels. Notice that the guards of the old events (e.g. *Unload*) are strengthened. Also, notice that when both exit devices are free, the carrier device unloads a part nondeterministically either toward the left exit device or toward the right exit device.

The transition system in Fig. 4 shows the first refinement operational semantics. We can notice that if we group the states according to the gluing invariant, we obtain two *modules* (one where in each state the carrier device is empty and one where it is busy) corresponding to the two states of the abstract level.

EVENT SYSTEM ROBOT_1 **REFINES** ROBOT_0
VARIABLES: LED_1, RED_1, CD_1
INVARIANT:
$\mathrm{LED}_1 \in \mathrm{DEVICE\_STATE} \wedge \mathrm{RED}_1 \in \mathrm{DEVICE\_STATE} \wedge \mathrm{CD}_1 = \mathrm{CD}_0$
INITIALIZATION: LED_1:=f || RED_1:=f || CD_1:=f
EVENTS:
/* Old events */
Load $\widehat{=}$**SELECT** CD_1=f **THEN** CD_1:=b **END**;
Unload $\widehat{=}$**SELECT** (LED_1=f $\vee$ RED_1=f) $\wedge$ CD_1=b
    THEN IF LED_1=b **THEN** RED_1:=b
        ELSE IF RED_1=b **THEN** LED_1:=b
            ELSE CHOICE LED_1:=b
                OR RED_1:=b **END**
            END
        END
        || CD_1:=f
    END;

/* New events */
LEvac $\widehat{=}$**SELECT** LED_1=b **THEN** LED_1:=f **END**;
REvac $\widehat{=}$**SELECT** RED_1=b **THEN** RED_1:=f **END**
ENDROBOT_1

Second Refinement. Now, we observe two new events (*LRotate* and *RRotate*) which change the carrier device side which is registered in the variable *CDS*. The side value is either left—denoted *l*, or right—denoted *r*. We remove the nondeterminism by giving priority to the left exit device whenever possible—this for minimizing the carrier device movements, i.e., we do not unload on the right exit device if the left one is free.

EVENT SYSTEM ROBOT_2 **REFINES** ROBOT_1
SETS: SIDE={l,r}
VARIABLES: LED_2, RED_2, CD_2, CDS_2

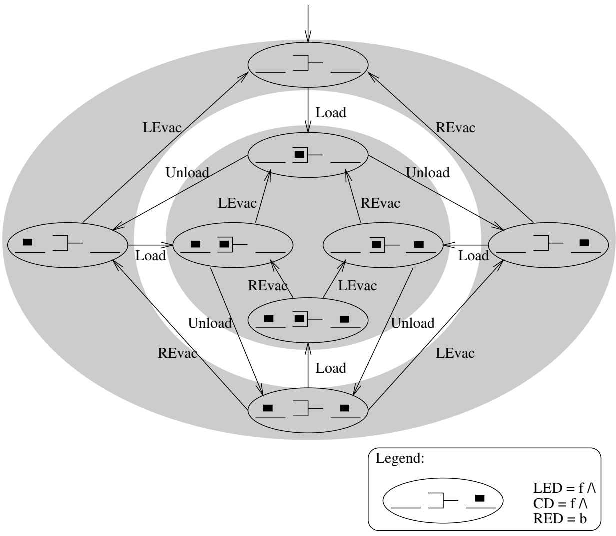

Fig. 4. First refinement transition system

INVARIANT:
LED_2=LED_1 ∧ RED_2=RED_1 ∧ CD_2=CD_1 ∧ CDS_2 ∈ SIDE ∧
(CDS_2=r ⇒(CD_2=f ∨ RED_2=f))
INITIALIZATION: LED_2:=f ‖ RED_2:=f ‖ CD_2:=f ‖ CDS_2:=l
EVENTS:
/* Old events */
Load $\hat{=}$**SELECT** CD_2=f ∧ CDS_2=l **THEN** CD_2:=b **END**;
Unload $\hat{=}$**SELECT** CD_2=b ∧ ((LED_2=f ∧ CDS_2=l) ∨
(RED_2=f ∧ CDS_2=r ∧ LED_2=b))
THEN **IF** LED_2=f ∧ CDS_2=l **THEN** LED_2:=b
ELSE RED_2:=b
END
‖ CD_2:=f
END;
LEvac $\hat{=}$**SELECT** LED_2=b **THEN** LED_2:=f **END**;
REvac $\hat{=}$**SELECT** RED_2=b **THEN** RED_2:=f **END**;

/* New events */
LRotate $\hat{=}$**SELECT** CDS_2=r ∧ (CD_2=f ∨ LED_2=f)
THEN CDS_2:=l
END;
RRotate $\hat{=}$**SELECT** CDS_2=l ∧ CD_2=b ∧ LED_2=b ∧ RED_2=f
THEN CDS_2:=r
END
END

Figure 5 shows the second refinement transition system. We can notice that if we group the states according to the gluing invariant, we obtain eight modules.

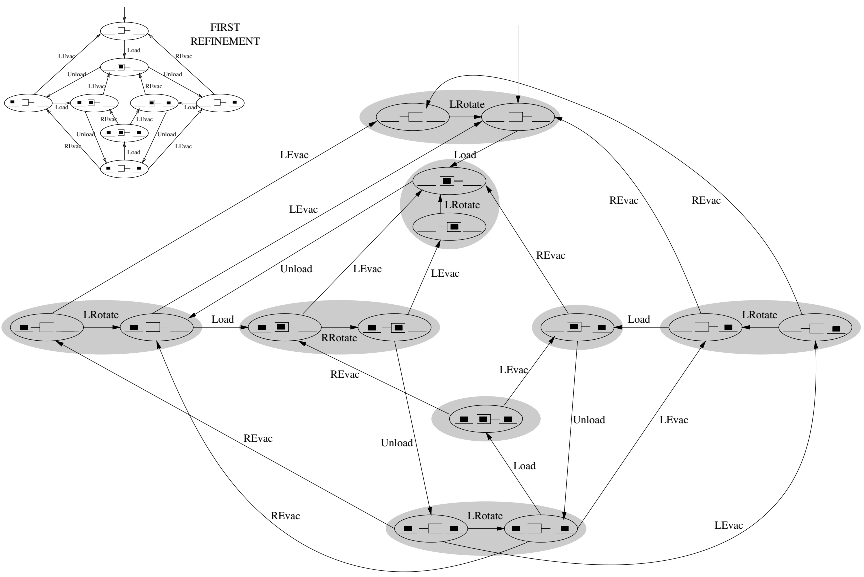

Fig. 5. Second refinement transition system

2.2 Dynamic Properties

We want to express some dynamic properties to be verified on the systems previously described.

Dynamic Properties of the Abstraction. The carrier device should immediately unload. This is expressed by Formula 1 which can be read as follows: "At any time, if the carrier is busy, it is free in the next state".

$$\square(CD_0 = b \Rightarrow \bigcirc(CD_0 = f)) \tag{1}$$

Dynamic Properties for the First Refinement. The previous property is reformulated either into Formula 2 or into Formula 3.

$$\square(CD_1 = b \Rightarrow \lozenge(CD_1 = f)) \tag{2}$$

$$\square(CD_1 = b \Rightarrow \lozenge(CD_1 = f \wedge (LED_1 = b \vee RED_1 = b))) \tag{3}$$

Formula 2 considers that any event which does not modify the carrier state may happen before unloading. If the refinement holds, Formula 2 holds without any further verification since any instance of a formula $\Box(p \Rightarrow \bigcirc q)$ refines into the same instance of $\Box(p \Rightarrow \Diamond q)$. So, the reformulation is unnecessary. However, Formula 3 expresses more about how the unloading may happen. It says explicitly that a busy carrier device will eventually unload on one of the two exit devices. So, the reformulation allows precising the abstraction.

Remark 1. In both cases, the temporal operator $\bigcirc$ becomes a $\Diamond$ in the reformulation. We say that the reformulation follows a refinement pattern $(\Box(p \Rightarrow \bigcirc q), \Box(p \Rightarrow \Diamond q))$, for short $\bigcirc\Diamond$.

We also have the two following new properties:

- The carrier device holds the part until one of the exit devices becomes free.

$$\begin{aligned}\Box(&CD_1 = b \\ &\Rightarrow ((CD_1 = b)\mathcal{U}(CD_1 = b \wedge (LED_1 = f \vee RED_1 = f))))\end{aligned} \tag{4}$$

  Formula 4 could also be considered as a reformulation of the abstract property 1.
- If the three devices are busy, then, the carrier device remains busy. In other words, only the evacuation is allowed.

$$\Box((CD_1 = b \wedge LED_1 = b \wedge RED_1 = b) \Rightarrow \bigcirc(CD_1 = b)) \tag{5}$$

Dynamic Properties for the Second Refinement. The above properties can be reformulated as follows by taking the carrier device side into consideration.

- Formula 3 is reformulated into Formula 6, following a $\Diamond\Diamond$ pattern:

$$\begin{aligned}\Box(CD_2 = b \Rightarrow \Diamond(CD_2 = f \wedge ((LED_2 = b \wedge CDS_2 = l) \vee \\ (RED_2 = b \wedge CDS_2 = r))))\end{aligned} \tag{6}$$

  The enrichment of Formula 3 in Formula 6 consists in expressing that "the carrier device is turned to side of the preceding unloading".
- Formula 4 is reformulated into Formula 7, following a $\mathcal{U}\mathcal{U}$ pattern:

$$\begin{aligned}\Box((CD_2 = b \wedge LED_2 = b \wedge CDS_2 = l) \Rightarrow ((CDS_2 = l \wedge CD_2 = b) \\ \mathcal{U}(CDS_2 = l \wedge CD_2 = b \wedge (LED_2 = f \vee RED_2 = f))))\end{aligned} \tag{7}$$

  It can be read as: "If the carrier device is busy and directed toward its left, it stays directed toward the left until it can turn to the right.
- Formula 5 is reformulated into Formula 8. It follows a $\bigcirc\bigcirc$ pattern by expressing that the *new* events *LRotate* and *RRotate* must preserve the $\bigcirc$.

$$\begin{aligned}\Box((&CD_2 = b \wedge LED_2 = b \wedge RED_2 = b) \\ &\Rightarrow \bigcirc(CD_2 = b))\end{aligned} \tag{8}$$

It can be read as: “The carrier device must stay busy when rotating”.

The next section describes how to deduce invariants providing sufficient verification conditions from the *PLTL* property syntax at both the refined and the abstract levels together with the path refinement relation .

3 Reformulated Dynamic Property Verification

In this section, we explain how to verify the reformulated dynamic properties through refinement. We suppose that the system TS_2 of state space S_2 refines the abstract system TS_1 of state space S_1 and we exploit this refinement to show that if a property P_1 holds on the abstract system then a reformulated property P_2 holds on a refined system.

We hope to avoid the *PLTL* property model-checking explosion which is likely to happen during the refinement by providing sufficient conditions to verify the reformulated *PLTL* property P_2 using that η holds between TS_2 and TS_1 and P_1 holds on TS_1 as assumptions.

These conditions are first-order predicate formulae where the predicate domains are limited to finite sets so that these conditions are easily decidable by any theorem prover. Moreover, the conditions depend on the formulation of the *PLTL* property at both levels.

We determine two kinds of conditions. The first kind does not take into account the events. These conditions are often too weak to prove the *PLTL* formulae. So, we consider how the new events are interwoven among the old events to exhibit stronger conditions. Therefore, as in the B proof obligations for refinement, these last conditions are formulated using the guards and the generalized substitutions of the events.

3.1 About Refinement

We use transition systems as operational semantics of the B event systems because of the *PLTL* semantics. At the abstract level, a *PLTL* property P is verified for the event system by model-checking on a transition system which is its operational model. This supposes that the set of states of this transition system is finite.

In this section, we give an intuitive presentation of a refinement relation between the set of states S_2 of the refined transition system TS_2 and the set of states S_1 of the abstract transition system TS_1 which determines a relation between the paths of the transition systems modeling the corresponding event systems. This relation has been studied thoroughly in [4].

As in B, the important assumption is that the new events do not take control forever. However, in our approach, this is verified by a model state enumeration. The refinement verification is linear in the number of states of the refined system. This way we prevent the state explosion coming from the *PLTL* model-checking itself.

As for the refinement in B, the conjunction of the abstract system invariant I_1 and the gluing invariant I_2 determines a relation μ between the refined and abstract states. The refinement relation η restricts μ taking into account that the new events do not take the control forever, and that the non determinism may decrease. The relation η between states implies a relation between the refined paths and some abstract paths of the transition systems. Figure 6 gives an example of two related paths. As usual, the *PLTL* model-checking is based on the labeling of each state by a set of the propositions holding on it. By the refinement definition from [4], it is very important to ensure that any event which is taken on the abstract path is eventually taken on the refined path preceded by some new events.

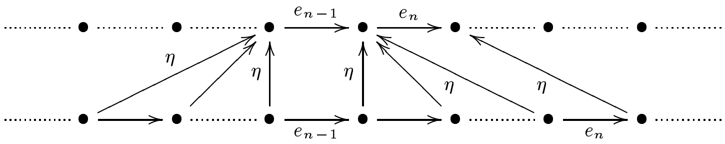

Fig. 6. Path refinement example

3.2 Refinement and Dynamic Properties

The dynamic properties that can be expressed in the B event systems are

- either a dynamic invariant which indicates how the variables of the system are authorized to evolve; this corresponds roughly to a *PLTL* formula involving the next operator as $\Box(p \Rightarrow \bigcirc q)$;
- or the B modalities which have *PLTL* equivalences as the patterns $\Box(p \Rightarrow \Diamond q)$ and $\Box(p \Rightarrow (p \mathcal{U} q))$.

Generally, a *PLTL* formula following the pattern $\Box(p_1 \Rightarrow \bigcirc q_1)$ is formulated again at a refined level either as the pattern $\Box(p_2 \Rightarrow \bigcirc q_2)$ or $\Box(p_2 \Rightarrow \Diamond q_2)$ or $\Box(p_2 \Rightarrow (r_2 \mathcal{U} q_2))$. It can be a more complicated expression. We call a *refinement pattern* a pair of a *PLTL* pattern and its reformulated pattern. Notice that in a given pattern, the variables are propositional variables.

Our approach allows the user to have more possibilities to express properties through refinement than in B.

On the one hand, the preservation of dynamic invariant through the B refinement seems to correspond to the refinement pattern $\big(\Box(p_1 \Rightarrow \bigcirc q_1), \Box(p_2 \Rightarrow \Diamond q_2)\big)$ (for short, $\bigcirc\Diamond$).

On the other hand, the B refinement preserves the modalities patterns. Again the reformulation offers more possibilities. First, the pattern $\mathcal{U}$ is $\Box(p_1 \Rightarrow (r_1 \mathcal{U} q_1))$ whereas the B modality *until* corresponds to a pattern $\Box(p_1 \Rightarrow (p_1 \mathcal{U} q_1))$. Second, a pattern $\mathcal{U}$ may evolve into a pattern $\Diamond$. Third, the reformulation allows enriching gradually a property through refinement. In contrast, when a property P does not need to be enriched, it is preserved by refinement so that, in such a case, it is useless to reformulate P during further refinements.

Notice that it is inconceivable that a pattern $\Diamond$ evolves into a pattern $\bigcirc$. The direction of the implication between the patterns of the pair is naturally mirrored by the direction of the refinement. We have discussed the pattern evolution through refinement in [9].

The sufficient proof conditions are deduced from the *PLTL* refinement pattern semantics. So, we are not limited to a small set of refinement patterns. Our experience shows that in most applications the same few refinement patterns are often used. However, a small number of more complicated refinement patterns may be novel to a particular application but it is generally easy to build a corresponding sufficient condition set as it is shown in the next section.

3.3 Weak Sufficient Conditions

Consider the refinement pattern $\mathcal{U}\mathcal{U}$. Suppose a formula of pattern $\Box(p_1 \Rightarrow (r_1 \mathcal{U} q_1))$ holds on the paths of the abstract transition system TS_1. We want to have sufficient conditions for the pattern $\Box(p_2 \Rightarrow (r_2 \mathcal{U} q_2))$ holding on the paths of a refined transition system TS_2.

From the semantics of $\mathcal{U}$ and from the path refinement relation as shown in Fig. 6, we deduce the following set of sufficient conditions.

- *A beginning condition.* Assume p_2 is satisfied on a state s_2, and s_1 be the abstract state such that s_2 together with s_1 satisfy $I_2 \wedge I_1$. Then, p_1 must be satisfied by s_1. From that we deduce a first condition $p_2 \wedge I_2 \wedge I_1 \Rightarrow p_1$.
- *A maintenance condition.* The proposition r_1 holds on each state s of any path of the abstract system beginning in s_1 before the satisfaction of q_1. So, r_2 must also hold on each state s' of any path of a refined system beginning in s_2 before the satisfaction of q_2. By refinement, s and s' satisfy $I_2 \wedge I_1$. From that we deduce a second condition $r_1 \wedge I_2 \wedge I_1 \Rightarrow r_2$.
- *An ending condition.* On any path after s_1 there exists a state t satisfying q_1. So, if q_2 holds on a state t' such that t and t' satisfy $I_2 \wedge I_1$, we are done. We deduce the third condition $q_1 \wedge I_2 \wedge I_1 \Rightarrow q_2$.

We see that we have two kinds of implications, one from an abstract system property to a refined system property (either for an ending condition or a maintenance condition), and the other from a refined system property to an abstract system property (for a beginning condition) (see Fig. 7).

We now give theorems providing a building block for a beginning condition.

Theorem 1. *Given an abstract transition system TS_1 of state space S_1, and a transition system TS_2 of state space S_2 refining TS_1, let I_1 be the invariant of TS_1, and let I_2 be the gluing invariant. Each state $s_1 (\in S_1)$ glued with a state $s_2 (\in S_2)$ on which a proposition p_2 holds satisfies a proposition p_1 if the condition $p_2 \wedge I_2 \wedge I_1 \Rightarrow p_1$ holds on $s_2 \wedge s_1$.*

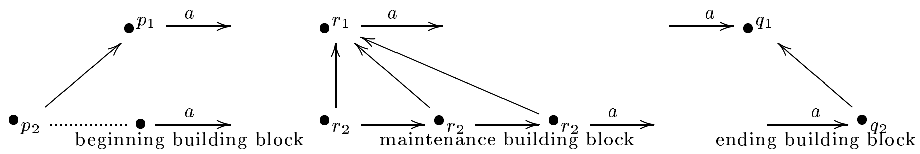

Fig. 7. Building blocks

Proof. Immediate by the following. Let $s_2 \in S_2$ be a state satisfying p_2. Let $s_1 \in S_1$ be a state glued with s_2. Then, s_2 and s_1 satisfy $p_2 \wedge I_1 \wedge I_2$. Since $p_2 \wedge I_2 \wedge I_1 \Rightarrow p_1$, the property p_1 which contains only variables of TS_1, holds on s_1.

Theorem 2. *The condition stated by Theorem 1 is a building block for a beginning condition of any refinement pattern* $\big(\Box(p_1 \Rightarrow Q_1), \Box(p_2 \Rightarrow Q_2)\big)$ *where* Q_1 *and* Q_2 *are PLTL formulae.*

Proof. Immediate by the following. If a refined path begins in a state satisfying p_2 then it is necessarily glued with all the states in S_1 satisfying p_1.

We propose another building block either for a maintenance condition or for an ending condition.

Theorem 3. *Given an abstract transition system* TS_1 *and a transition system* TS_2 *refining* TS_1*, let* I_1 *be the invariant of* TS_1*, and let* I_2 *be the gluing invariant. Each state* $s_2 (\in S_2)$ *glued with a state* $s_1 (\in S_1)$ *on which a proposition* q_1 *holds satisfies a proposition* q_2 *if the condition* $q_1 \wedge I_2 \wedge I_1 \Rightarrow q_2$ *holds on* $s_2 \wedge s_1$*.*

Proof. The proof is the same as for Theorem 1.

Theorem 4. *The condition stated by Theorem 3 is a building block for an ending condition of any refinement pattern* $\big(\Box Q_1, \Box Q_2\big)$ *where* q_1 *is an eventuality which occurs in the PLTL formula* Q_1 *and* q_2 *is an eventuality which occurs in the PLTL formula* Q_2*.*

Proof. Immediate by the following. If an abstract state s_1 satisfying q_1 occurs in a path of S_1 then all the states in S_2 glued with s_1 are compelled to satisfy q_2.

As a consequence of Theorem 3, a maintenance condition can be deduced according to the following argument. Let s_1 be a state in S_1 for which r_1 holds. In a path of TS_2, all the states which begin a transition refining *skip* (these transitions are labeled by new events) that are glued with s_1 verify r_2. Notice how important are the *non* τ*-divergence* and the *lack of new deadlock* clauses of the refinement relation for reaching the ending condition. However, it may be too weak when new events appear since, in this case, the condition may delay the termination q_2 too long.

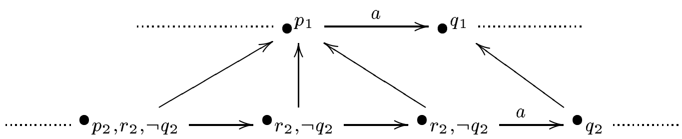

Fig. 8. Pattern $\bigcirc\mathcal{U}$

We now have a way to construct the set of weak sufficient conditions associated to one of the following often used refinement patterns $\bigcirc\bigcirc$, $\bigcirc\mathcal{U}$, $\bigcirc\Diamond$, $\mathcal{U}\mathcal{U}$, $\mathcal{U}\Diamond$, $\Diamond\Diamond$. For instance, the set of weak sufficient conditions for the refinement pattern $\big(\Box(p_1 \Rightarrow \bigcirc q_1), \Box(p_2 \Rightarrow (r_2 \mathcal{U} q_2))\big)$ is (see Fig. 8)

$$\begin{array}{l} p_2 \wedge I_2 \wedge I_1 \Rightarrow p_1, \ \textit{a beginning condition} \\ p_1 \wedge I_2 \wedge I_1 \Rightarrow r_2, \ \textit{a maintenance condition} \\ q_1 \wedge I_2 \wedge I_1 \Rightarrow q_2, \ \textit{an ending condition} \end{array}$$

Consider the following dynamic property for the event system from Section 2. If the carrier device is busy, and so are both exit devices then, in the next state, the carrier device remains busy while one of the exit devices becomes free.

$$\begin{array}{l} \Box((CD_1 = b \wedge LED_1 = b \wedge RED_1 = b) \Rightarrow \bigcirc((CD_1 = b) \\ \quad \wedge (LED_1 = f \vee RED_1 = f))) \end{array} \tag{9}$$

Notice that this implies Property 5 from Section 2.2. The above property is reformulated into

$$\begin{array}{l} \Box((CD_2 = b \wedge LED_2 = b \wedge RED_2 = b) \Rightarrow ((CD_2 = b) \\ \quad \mathcal{U}(CD_2 = b \wedge (LED_2 = f \vee RED_2 = f)))) \end{array} \tag{10}$$

because of the new events. The above set of weak sufficient conditions is enough to ensure the satisfaction of the reformulated property.

Moreover, the building blocks can also be used to deduce weak sufficient conditions for more complex refinement patterns. Consider, for example, a refinement pattern $\big(\Box(p_1 \Rightarrow \bigcirc(r_1 \mathcal{U} q_1)), \Box(p_2 \Rightarrow \Diamond(r_2 \Rightarrow r_2 \mathcal{U} q_2))\big)$. Its set of weak sufficient conditions is the following:

$$\begin{array}{l} p_2 \wedge I_2 \wedge I_1 \Rightarrow p_1, \ \textit{a beginning condition} \\ r_1 \wedge I_2 \wedge I_1 \Rightarrow r_2, \ \textit{a maintenance condition} \\ q_1 \wedge I_2 \wedge I_1 \Rightarrow q_2, \ \textit{an ending condition} \end{array}$$

Unfortunately, some of these building blocks are often too weak for the proof because there do not express the semantics of the refinement patterns precisely enough. The next section presents strong sufficient conditions which are used in our verification process when the weak sufficient conditions fail in proving an instance of a refinement pattern.

Obviously, the cause of the failure may not come from the conditions but either from the incorrectness of the refined formula (error of expression or error in the pattern evolution), or even from the incorrectness or the insufficiency of the gluing invariant. The problem with the invariant happens only if the modular refinement relation does not hold. So, the refinement verification eliminates this cause of failure.

3.4 Strong Sufficient Conditions

In the failure case, we have to try a strong sufficient condition based on the new events (refining *skip*) interwoven among the old events. For example, with the refinement pattern $\bigcirc\Diamond$ followed by Formula 1 and its reformulation into Formula 3 (see Section A.2 in Appendix), the proof of the set of weak conditions fails for the ending condition $q_1 \wedge I_2 \wedge I_1 \Rightarrow q_2$. Then, the weak ending condition has to be replaced by the following strong ending sufficient condition:

$$\forall\, a \in \mathit{OldEvents}, p_1 \wedge G_{1a} \wedge G_{2a} \wedge I_1 \wedge I_2 \Rightarrow [S_{2a}]q_2$$

where S_{2a} is a generalized substitution of an old event a in a refined system, G_{2a} is its guard, and G_{1a} is the guard of the event a in the abstract system. In other words, each old event a enabled by a refined state s_2 which satisfies r_2 and is glued with an abstract state s_1 satisfying p_1, changes s_2 into a state satisfying q_2.

The above condition also can be used as an ending condition for the refinement pattern $\bigcirc\mathcal{U}$. It is used in proving Formula 4 as a reformulation of Formula 1. However, for the pattern $\bigcirc\mathcal{U}$, the left-hand side of the implication often need to be strengthened with the conjunction of r_2 to better fit the semantics of $\mathcal{U}$.

Strong sufficient conditions are also required by the persistence of the *PLTL* operator $\bigcirc$ in a refinement pattern.

The persistence of the *PLTL* operator $\bigcirc$ in a pattern evolution. We can imagine three plausible refinement patterns coming from the abstract pattern $p_1 \Rightarrow \bigcirc q_1$.

- The more likely the pattern $\bigcirc$ evolves into eventuality patterns either $\Diamond$ (e.g Formula 1 reformulated into Formula 3) or $\mathcal{U}$ (e.g. Formula 1 reformulated into Formula 4) because of transitions refining *skip*.
- However, in a few cases, it may happen that the property is not concerned with the new events. For instance, Formula 5 of the second refinement is satisfied by both of the new events *LRotate* and *RRotate*. With this point of view, it is reformulated into Formula 8.

For a $\bigcirc$ persistence, the weakest set of sufficient conditions we can build is composed of the weak beginning condition, the weak ending condition, and a condition saying that no new event can precede an old event concerned with $\bigcirc$. This condition is

$$\forall\ a \in \mathit{NewEvents}, (p_1 \wedge I_1 \wedge I_2) \Rightarrow \neg G_a$$

where G_a is the guard of an event a. Therefore, any state s_2 satisfying p_2 must not satisfy a guard of a new event which is sufficient to ensure the persistence of $\neg p_2$.

More about strong sufficient conditions. Let us examine strong sufficient conditions for the pattern refinement $\mathcal{UU}$ followed by Formula 4 and its reformulation into Formula 7 as expressed in Section 2.2. The following strong ending condition

$$\forall\ a \in \mathit{OldEvents}, r_1 \wedge \neg q_1 \wedge r_2 \wedge \neg q_2 \wedge G_{1a} \wedge G_{2a} \wedge I_1 \wedge I_2 \Rightarrow [S_{2a}]q_2$$

is easily built for this refinement pattern (see Fig. 9). However, this condition is likely to fail because of the old events which may be taken before the $\mathcal{U}$ ends. So, the refinement pattern $\mathcal{UU}$ requires the following stronger sufficient ending

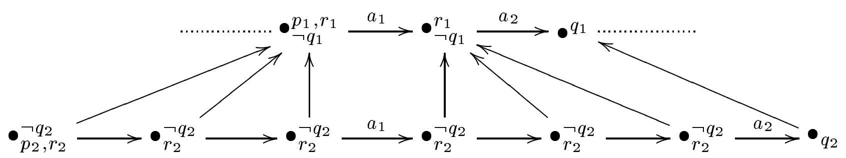

Fig. 9. Pattern $\mathcal{UU}$

condition which eliminates from the condition the events which might be taken before the $\mathcal{U}$ ends, i.e., the events which does not change the abstract state to an abstract state which satisfies q_1. For that, it is enough to strengthen the implication with the conjunction of $[S_{1a}]q_1$.

$$\begin{array}{l} \forall\ a \in \mathit{OldEvents}, \\ \quad r_1 \wedge \neg q_1 \wedge r_2 \wedge \neg q_2 \wedge G_{1a} \wedge G_{2a} \wedge I_1 \wedge I_2 \wedge [S_{1a}]q_1 \Rightarrow [S_{2a}]q_2 \end{array}$$

This stronger sufficient ending condition proves the reformulated Formula 7 of Section 2.2 (see A.1 in Appendix). The refinement pattern $\Diamond\Diamond$ follows the same ending stronger sufficient condition, condition which allows proving that, assuming the satisfaction of Formula 3, its reformulation by Formula 6 holds.

Notice that the strong sufficient conditions are universally quantified on either the set of the new events or the set of the old events of a refined system.

As for the deduction of the weak sufficient conditions, we can exhibit building blocks but they take into account the guards and the generalized substitutions of the involved events. Again, for a given refinement pattern, we get a constructive way to find the set of strong sufficient conditions by using building blocks according to their respective semantics.

3.5 Dynamic Properties Reformulation versus Variants and Loop Invariants

In this section, we have shown that the verification of *PLTL* reformulation of the dynamic properties can be fully automatic for finite state systems using no variant and no loop invariant.

Given a *PLTL* formula P_1 and its reformulation P_2, we get a systematic construction for finding a set of propositions (building blocks) which suffices to ensure that the refined property holds on the refined system. So, failure does not mean that the property is false. There are two main causes of failure:

- either the gluing invariant is too weak,
- or the property is false, but may be only outside the reachable state set.

Notice that it is the same as in B where an invariant which does not hold on the whole state space could be true on the reachable state space.

The sufficient conditions can be viewed as another way to use the preservation of P_1 by refinement as an assumption to verify P_2. Formulating again a property through refinement is useful for three reasons. First, in order to establish the gluing invariant, the user can have a path refinement style of reasoning and not only a variable connection one. Second, it allows us to deal with the model-checking explosion problem since we avoid to model-check the reformulated property by proving either weak or strong sufficient conditions. Third, it opens up an original solution to combine proof and model-checking techniques. This solution is based on the refinement.

4 Conclusion and Perspectives

We have proposed a distinctive approach for the verification of finite state reactive systems. The specifications are expressed as B finite state event systems, i.e., the variables range over finite domains. A significant number of reactive system, particularly communication and synchronization protocols, are in essence finite state systems. Moreover, the refinement design of event systems requires a decreasing variant for verifying that the new events—the events introduced in the specification during the refinement, do not take the control forever. Our goal is to liberate the design from such features. For that purpose, the verification consists in confronting the labeled transition system which models the finite state specification with its dynamic properties expressed by *PLTL* formulae. This can be done automatically by model-checking without additional features. Moreover, the refinement has to relate an abstract transition system and its refinement by the *modular refinement relation* η which is easily verified algorithmically in the particular case of finite state systems.

The combination of proof techniques and model-checking relies on a distinction between *reformulated* and *new properties*, the first being verified by a proof technique and the second, by model-checking.

Given a *PLTL* formula and its reformulation, we get a systematic construction for finding a set of propositions (building blocks) which suffices to ensure that the refined property holds on the refined system. So, failure does not mean that the property is false. There are two main causes of failure:

- either the gluing invariant is too weak,
- or the property is false, but may be only outside the reachable state set.

Notice that the same applies to B where an invariant which does not hold on the whole state space could be true on the reachable state space.

Reformulating a property through refinement is useful for three reasons. First, in order to establish the gluing invariant, the user can have a path refinement style of reasoning and not only a variable relationship one. Second, it allows us to deal with the model-checking explosion problem since we avoid model-checking the reformulated property by proving either weak or strong sufficient conditions. Third, it opens up a peculiar approach for combining proof and model-checking techniques. This approach is based on refinement.

We are currently improving and completing our toolkit so that we can validate the verification process on larger examples. This is also needed for validating the methodology on industrial size applications. This is essential for our claim about *reformulated/new properties*. We are also working on an extension of the class of *modular* properties that can be recognized. Furthermore, we would like to extend some of the above ideas from finite state systems to a class of parameterized systems.

References

1. J.-R. Abrial. Constructions d'automatismes industriels avec B. In *Congrès AFADL*, ONERA-CERT - Toulouse, France, May 1997. Invited lecture.
2. J.-R. Abrial and L. Mussat. Introducing dynamic constraints in B. In *Second Conference on the B method, France*, volume 1393 of *Lecture Notes in Computer Science*, pages 83–128. Springer Verlag, April 1998.
3. F. Bellegarde, C. Darlot, J. Julliand, and O. Kouchnarenko. Reformulate dynamic properties during B refinement and forget variants and loop invariants. In *Proc. Int. Conf. ZB'2000, York, Angleterre*. Springer-Verlag, August 2000. LNCS to appear.
4. F. Bellegarde, J. Julliand, and O. Kouchnarenko. Ready-simulation is not ready to express a modular refinement relation. In *Proc. Int. Conf. on Fondamental Aspects of Software Engineering, FASE'2000*, volume 1783 of *Lecture Notes in Computer Science*, pages 266–283. Springer-Verlag, April 2000.
5. F. Bellegarde, J. Julliand, and H. Mountassir. Model-based verification through refinement of finite B event systems. In *Formal Method'99 B User Group Meeting*, CD-ROM publication, 1999.
6. J. Dingel and T. Filkorn. Model checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving. In *Computer Aided Verification, CAV'95*, volume 939 of *Lecture Notes in Computer Science*, pages 54–69, Liège, Belgique, July 1995. Springer-Verlag.
7. R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. IFIP-WG6.1 Symposium On Protocols Specification, Testing and Verification (PSTV95)*, pages 2–21, Warsaw - Poland, 1995.

8. G. Holzmann. *Design and validation of protocols.* Prentice hall software series, 1991.
9. J. Julliand, F. Bellegarde, and B. Parreaux. De l'expression des besoins à l'expression formelle des propriétés dynamiques. *Technique et Science Informatiques*, 18(7), 1999.
10. L. Lamport. A temporal logic of actions. *ACM Transactions On Programming Languages And Systems, TOPLAS*, 16(3):872–923, May 1994.
11. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–35, January 1995.
12. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag - ISBN 0-387-97664-7, 1992.
13. D. A. Peled. Combining partial order reduction with on-the-fly model checking. In *Proc. 6th Int. Conf. On Computer Aided Verification (CAV94)*, San-Francisco, USA, 1994.
14. P. Wolper and P. Godefroid. Partial-order methods for temporal verifications. In *Conference on Concurrency Theory, CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246, Hidesheim, aug 1993. Springer Verlag.

A Proofs of Reformulation

A.1 A Proof of Reformulation Following a Refinement Pattern $\mathcal{UU}$

In this section, I_1 is the conjunction of the invariants of the event systems Robot_0 and Robot_1. Also, I_2 is the invariant of the event system Robot_2.

Formula 4 is reformulated into Formula 7 following a pattern $\mathcal{UU}$ as below.

$$\begin{array}{l}\Box(CD_1 = b \Rightarrow ((CD_1 = b)\mathcal{U}(CD_1 = b \wedge (LED_1 = f \vee RED_1 = f)))),\\ \Box(CD_2 = b \wedge LED_2 = b \wedge CDS_2 = l) \Rightarrow (CDS_2 = l \wedge CD_2 = b)\\ \quad\quad \mathcal{U}(CDS_2 = l \wedge CD_2 = b \wedge (LED_2 = f \vee RED_2 = f))).\end{array}$$

So, we have

$$\begin{array}{l}p_1 \hat{=} CD_1 = b\\ r_1 \hat{=} CD_1 = b\\ q_1 \hat{=} CD_1 = b \wedge (LED_1 = f \vee RED_1 = f)\\ p_2 \hat{=} CD_2 = b \wedge LED_2 = b \wedge CDS_2 = l\\ r_2 \hat{=} CDS_2 = l \wedge CD_2 = b\\ q_2 \hat{=} CDS_2 = l \wedge CD_2 = b \wedge (LED_2 = f \vee RED_2 = f).\end{array}$$

Beginning Condition. Consider the weak sufficient condition $p_2 \wedge I_1 \wedge I_2 \Rightarrow p_1$ defined in Section 3.

Proof. Immediate because of the gluing invariant $(CD_1 = CD_2)$.

Maintenance Condition. We have to prove the following three implications which have not been studied in Section 3.

1. $p_1 \wedge p_2 \wedge I_1 \wedge I_2 \Rightarrow r_2$
2. $\forall a \in OldEvents, \neg q_1 \wedge [S_{1_a}]\neg q_1 \wedge r_1 \wedge G_{1_a} \wedge G_{2_a} \wedge I_1 \wedge I_2 \Rightarrow [S_{2_a}]r_2$
3. $\forall a \in NewEvents, r_1 \wedge \neg q_1 \wedge G_{2_a} \wedge I_1 \wedge I_2 \Rightarrow [S_{2_a}]r_2$

We can explain these three conditions in looking at the refinement pattern in Fig. 9. The first condition ensures that r_2 holds in the first state which satisfies the propositions p_1 and p_2. The second condition guarantees that among the old events which could be taken before $\mathcal{U}$ ends, each one but the last maintains r_2. The last old event leads to a state satisfying q_2. The third condition verifies that all the new events which could be taken inside the $\mathcal{U}$ maintain r_2.

Proof. 1. Obvious because $p_2 \Rightarrow r_2$

2. Consider all the *old events.*

   (a) *Load*:

$$\begin{aligned}
&CD_1 = f \vee (LED_1 = b \wedge RED_1 = b)\\
\wedge&[CD_1 := b](CD_1 = f \vee (LED_1 = b \wedge RED_1 = b))\\
\wedge&CD_1 = b\\
\wedge&CD_1 = f\\
\wedge&CD_2 = f \wedge CDS_2 = l\\
\wedge&I_1 \wedge I_2\\
\Rightarrow&[CD_2 := b](CDS_2 = l \wedge CD_2 = b)
\end{aligned}$$

   There is a contradiction between $r_1 \hat{=} CD_1 = b$ and $G_{1_a} \hat{=} CD_1 = f$ in the left part of the implication. So, since $false \Rightarrow anything$ is $true$, the condition is verified for the old event *Load.*

   (b) *Unload*:

$$\begin{aligned}
&CD_1 = f \vee (LED_1 = b \wedge RED_1 = b)\\
\wedge&[S_{1_{\mathrm{Unload}}}](CD_1 = f \vee (LED_1 = b \wedge RED_1 = b))\\
\wedge&CD_1 = b\\
\wedge&(LED_1 = f \vee RED_1 = f) \wedge CD_1 = b\\
\wedge&(CD_2 = b \wedge ((LED_2 = f \wedge CDS_2 = l)\\
&\vee (RED_2 = f \wedge CDS_2 = r \wedge LED_2 = b))\\
\wedge&I_1 \wedge I_2\\
\Rightarrow&[S_{2_{\mathrm{Unload}}}](CDS_2 = l \wedge CD_2 = b)
\end{aligned}$$

   Notice that q_1 and G_{1_a} are the same. The condition is proved because we cannot have $\neg q_1$ (see above the first line) and G_{1_a} (see above the fourth line)—a contradiction in the left-hand side of the implication.

(c) *LEvac*:

$$
\begin{aligned}
&CD_1 = f \vee (LED_1 = b \wedge RED_1 = b)\\
&\wedge[LED_1 := f](CD_1 = f \vee (LED_1 = b \wedge RED_1 = b))\\
&\wedge CD_1 = b\\
&\wedge LED_1 = b\\
&\wedge LED_2 = b\\
&\wedge I_1 \wedge I_2\\
&\Rightarrow[LED_2 := f](CDS_2 = l \wedge CD_2 = b)
\end{aligned}
$$

Consider above the second line where one of the two operands of the disjunction should be true. The right operand $[LED_1 := f](LED_1 = b \wedge RED_1 = b)$ cannot be true because the substitution leads to $f = b$ which is false. Moreover, the left operand $CD_1 = f$ contradicts the third line above. So, the condition is proved.

(d) *REvac*:

$$
\begin{aligned}
&CD_2 = f \vee (LED_1 = b \wedge RED_1 = b)\\
&\wedge[RED_1 := f](CD_1 = f \vee (LED_1 = b \wedge RED_1 = b))\\
&\wedge CD_1 = b\\
&\wedge RED_1 = b\\
&\wedge RED_2 = b\\
&\wedge I_1 \wedge I_2\\
&\Rightarrow[RED_2 := f](CDS_2 = l \wedge CD_2 = b)
\end{aligned}
$$

The proof is the same as above.

3. Consider all the *new events*.

(a) *LRotate*:

$$
\begin{aligned}
&CD_2 = b \wedge CDS_2 = l\\
&\wedge CDS_2 = r \vee CD_2 = f \vee (LED_2 = b \wedge RED_2 = b)\\
&\wedge CDS_2 = r \wedge (CD_2 = f \vee LED_2 = f)\\
&\wedge I_1 \wedge I_2\\
&\Rightarrow[CDS_2 := l](CDS_2 = l \wedge CD_2 = b)
\end{aligned}
$$

Consider the second line above. At least one part of the disjunction must be true. There are two cases. In the first case, $CD_2 = f$ or $CDS_2 = r$. Then, the first line above contradicts the second line. In the second case, $(LED_1 = b \wedge RED_1 = b)$. Consider the third line. One of the operands of the disjunction must hold. Since we do not have $LED_2 = f$, $CD_2 = f$ is true. This contradicts $CD_2 = b$ in the first line. So, the condition is proved.

(b) *RRotate*:

$$
\begin{aligned}
&CD_2 = b \wedge CDS_2 = l\\
&\wedge CDS_2 = r \vee CD_2 = f \vee (LED_2 = b \wedge RED_2 = b)\\
&\wedge CDS_2 = l \wedge CD_2 = b \wedge LED_2 = b \wedge RED_2 = f\\
&\wedge I_1 \wedge I_2\\
&\Rightarrow[CDS_2 := r](CDS_2 = l \wedge CD_2 = b)
\end{aligned}
$$

Obviously, the second line contradicts the third line. So, the condition is proved.

Ending Condition. In what follows, I_1 is the invariant of the event system Robot$_0$ and I_2 is the invariant of the event system Robot$_1$. We have to prove the following implication–the second formula in the subsection "More about strong sufficient conditions" in Section 3: $\forall a \in OldEvents, r_1 \wedge \neg q_1 \wedge r_2 \wedge \neg q_2 \wedge G_{1_a} \wedge G_{2_a} \wedge I_1 \wedge I_2 \wedge [S_{1_a}]q_1 \Rightarrow [S_{2_a}]q_2$.

Proof. Consider the *old events.*

1. *Load*:

$$
\begin{aligned}
&CD_1 = b \\
\wedge &CD_1 = f \vee (LED_1 = b \wedge RED_1 = b) \\
\wedge &CD_2 = b \wedge CDS_2 = l \\
\wedge &CDS_2 = r \vee CD_2 = f \vee (LED_2 = b \wedge RED_2 = b) \\
\wedge &CD_1 = f \\
\wedge &CD_2 = f \wedge CDS_2 = l \\
\wedge &I_1 \wedge I_2 \\
\wedge &[CD_1 := b](CD_1 = b \wedge (LED_1 = f \vee RED_1 = f)) \\
\Rightarrow &[CD_2 := b](CDS_2 = l \wedge CD_2 = b \wedge (LED_2 = f \vee RED_2 = f))
\end{aligned}
$$

The variable CD_1 has two contradictory values in the first and fifth above lines. So, the left-hand side of the implication is false. Therefore, the condition is proved.

2. *Unload*:

$$
\begin{aligned}
&CD_1 = b \\
\wedge &CD_1 = f \vee (LED_1 = b \wedge RED_1 = b) \\
\wedge &CD_2 = b \wedge CDS_2 = l \\
\wedge &CDS_2 = r \vee CD_2 = f \vee (LED_2 = b \wedge RED_2 = b) \\
\wedge &CD_1 = b \wedge (LED_1 = f \vee RED_1 = f) \\
\wedge &(CD_2 = b \wedge ((LED_2 = f \wedge CDS_2 = l) \\
&\quad \vee (RED_2 = f \wedge CDS_2 = r \wedge LED_2 = b)) \\
\wedge &I_1 \wedge I_2 \\
\wedge &[S_{1_{\text{Unload}}}]((CD_1 = b \wedge (LED_1 = f \vee RED_1 = f)) \\
\Rightarrow &[S_{2_{\text{Unload}}}](CDS_2 = l \wedge CD_2 = b \wedge (LED_2 = f \vee RED_2 = f))
\end{aligned}
$$

The second line contradicts the fifth line in the left-hand side of the implication. So, the condition is proved.

3. *LEvac*:

$$
\begin{aligned}
& CD_1 = b \\
\wedge & CD_1 = f \vee (LED_1 = b \wedge RED_1 = b) \\
\wedge & CD_2 = b \wedge CDS_2 = l \\
\wedge & CDS_2 = r \vee CD_2 = f \vee (LED_2 = b \wedge RED_2 = b) \\
\wedge & LED_1 = b \\
\wedge & LED_2 = b \\
\wedge & I_1 \wedge I_2 \\
\wedge & [LED_1 := f](CD_1 = b \wedge (LED_1 = f \vee RED_1 = f)) \\
\Rightarrow & [LED_2 := f](CDS_2 = l \wedge CD_2 = b \wedge (LED_2 = f \vee RED_2 = f))
\end{aligned}
$$

   The substitution reduces the right-hand side to $CDS_2 = l \wedge CD_2 = b$ which is equivalent to the third conjunct in the left-hand side of the implication.

4. *REvac*:

$$
\begin{aligned}
& CD_1 = b \\
\wedge & CD_1 = f \vee (LED_1 = b \wedge RED_1 = b) \\
\wedge & CD_2 = b \wedge CDS_2 = l \\
\wedge & CDS_2 = r \vee CD_2 = f \vee (LED_2 = b \wedge RED_2 = b) \\
\wedge & RED_1 = b \\
\wedge & RED_2 = b \\
\wedge & I_1 \wedge I_2 \\
\wedge & [RED_1 := f](CD_1 = b \wedge (LED_1 = f \vee RED_1 = f)) \\
\Rightarrow & [RED_2 := f](CDS_2 = l \wedge CD_2 = b \wedge (LED_2 = f \vee RED_2 = f))
\end{aligned}
$$

The condition is verified by the same proof as above.

A.2 A Proof of Reformulation Following a Refinement Pattern $\bigcirc\Diamond$

The formula 1 is reformulated following a $\bigcirc\Diamond$ pattern into the formula 3 as below.

$$
\begin{aligned}
& \Box(CD_0 = b \Rightarrow \bigcirc CD_0 = f), \\
& \Box(CD_1 = b \Rightarrow \Diamond(CD_1 = f \wedge (LED_1 = b \vee RED_1 = b))).
\end{aligned}
$$

We have

$$
\begin{aligned}
p_1 & \hat{=} CD_0 = b \\
q_1 & \hat{=} CD_0 = f \\
p_2 & \hat{=} CD_1 = b \\
q_2 & \hat{=} CD_1 = f \wedge (LED_1 = b \vee RED_1 = b).
\end{aligned}
$$

Beginning condition. Let us prove that $p_2 \wedge I_1 \wedge I_2 \Rightarrow p_1$

Proof. Immediate because of the gluing invariant ($CD_0 = CD_1$).

Ending condition. Let us prove that $\forall a \in OldEvents, p_1 \wedge G_{1_a} \wedge G_{2_a} \wedge I_1 \wedge I_2 \Rightarrow [S_{2_a}]q_2$

Proof. Consider all the *old events.*

1. *Load*:

$$\begin{aligned}
&CD_0 = b \\
&\wedge CD_0 = f \\
&\wedge CD_1 = f \\
&\wedge I_1 \wedge I_2 \\
&\Rightarrow [CD_1 := b](CD_1 = f \wedge (LED_1 = b \vee RED_1 = b)
\end{aligned}$$

   There is an obvious contradiction in the left-hand part of the implication. So, the implication is true.

2. *Unload*:

$$\begin{aligned}
&CD_0 = b \\
&\wedge CD_0 = b \\
&\wedge (LED_1 = f \vee RED_1 = f) \wedge CD_1 = b \\
&\wedge I_1 \wedge I_2 \\
&\Rightarrow [S_{1_{\text{Unload}}}](CD_1 = f \wedge (LED_1 = b \vee RED_1 = b)
\end{aligned}$$

   The implication is obviously true.