Modular Verification for a Class of PLTL Properties

Pierre-Alain Masson, Hassan Mountassir, Jacques Julliand

Modular Verification for a Class of *PLTL* Properties

Pierre-Alain Masson, Hassan Mountassir, and Jacques Julliand

LIFC - 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
{masson,mountass,julliand}@lifc.univ-fcomte.fr
Web : http://lifc.univ-fcomte.fr

Abstract. The verification of dynamic properties of a reactive systems by model-checking leads to a potential combinatorial explosion of the state space that has to be checked. In order to deal with this problem, we define a strategy based on local verifications rather than on a global verification. The idea is to split the system into subsystems called modules, and to verify the properties on each module in separation. We prove for a class of $PLTL$ properties that if a property is satisfied on each module, then it is globally satisfied. We call such properties *modular properties*. We propose a modular decomposition based on the B refinement process.
We present in this paper an usual class of dynamic properties in the shape of $\Box(p \Rightarrow Q)$, where p is a proposition and Q is a simple temporal formula, such as $\bigcirc q$, $\Diamond q$, or $q\mathcal{U}r$ (with q and r being propositions). We prove that these dynamic properties are modular. For these specific patterns, we have exhibited some syntactic conditions of modularity on their corresponding Büchi automata. These conditions define a larger class which contains other patterns such as $\Box(p \Rightarrow \bigcirc(q\mathcal{U}r))$.
Finally, we show through the example of an industrial Robot that this method is valid in a practical way.

Keywords. Refinement, modularity, Verification, model-checking, Büchi automata, Propositional linear temporal logic $PLTL$, B specification.

1 Introduction

This paper is about the problem of the verification of finite reactive systems [18, 3, 4]. The work that we present takes the B events system specification as a context [1], in which we introduce and verify dynamic properties of liveness and safety [16, 17].

A solution for the verification of dynamic properties is to use a model- checker [10, 11]. In comparison with proof techniques, such an approach offers the advantage of a possible and entire automatization, without the use of variants and loop invariant. The well known inconveniences of it are its limitation to finite

state systems and the potential combinatorial explosion of the number of states to be checked.

The problem of combinatorial explosion has been treated through various methods with time or memory space gain as a result. Let us cite by the way the memory compression techniques [12], the partial verification techniques based on heuristics [9], the memory efficient algorithms [6], the partial order techniques [21, 8] suppressing useless interleaving, the techniques of symbolic representation of the states with BDD and the techniques of state vector abstraction [7].

The approach that we expand below, based on the verification by model-checking, attacks the problem of the combinatorial explosion in a different way. We use a decomposition of the reachability graph into several modules. We then verify the properties module after module. We prove that an interesting class of $PLTL$ properties can be verified modularly. In others words, if the property holds on every module, we prove that it holds on the whole system.

All the decompositions are not equivalent. For some decompositions and for dynamic properties that can be verified in a modular way, the modular verification is valid whereas for some other decompositions, the modular verification might fail even when a property is true. Thus, the problem consists in finding a "good" decomposition. We use the notion of refinement induced by the B method to produce the modular split. The refinement of an abstract model by a concrete one induces a split of the concrete graph into subgraphs, according to the structure of the abstract graph. On such subgraphs, a modular verification is valid.

As a matter of fact, the B refinement process introduces new dynamic properties at each level of the refinement, together with the new events to which they are linked. So, the decomposition by refinement "keeps" the sequences of new events within subgraphs. This is the reason why such a split, which relies on a partition of the state space, is a "good" modular split. In other words, the refinement process produces a semantic split.

Some other modular approaches have been proposed that use compositional verification [13]. They are based on the parallel operationality which allows to split the model into components that contribute to the entire state space in a multiplicative way. Our approach splits the model of a component in an additive way. Since on one hand the B refinement allows the expression of parallel composition and, on the other hand, it preserves all the $PLTL$ properties but the ones that contain the next operator, the two techniques are compatible. It is possible to verify separately a process component with a modular approach in the sense of our proposition.

Conversely, in [15], Karen Laster and Orna Grumberg present a modular approach in our sense to temporal logic model-checking of software, in which a program text is partitioned into sequentially composed subprograms. The model-checking is then performed on each subprogram, separately but not independently. As a matter of fact, some information (by means of an *assumption function*) needs to be transmitted from one module to another. This is due to the fact that the partition proposed is a syntactic one.

Based on the refinement, the modularity that we propose is a semantic one. Furthermore, our modular approach is compatible with the use of the techniques that reduce the combinatorial explosion mentioned above.

Section 2 describes the concept of modular verification. In section 3, we define a class of dynamic properties by means of Büchi automata, and we prove that the properties in this class can be verified in a modular way. Section 4 defines the refinement relation between the abstract and refined specifications in B. To illustrate this work, we give in section 5 a simple example of an industrial Robot in order to exhibit the abstract and detailed specifications with their $PLTL$ properties. Finally, we conclude this work and give some future directions of research.

2 Modular Verification

2.1 Principle of the Modular Verification

The basic idea of the modular verification is simple. Since a transition system might be too large to be verified by model-checking in an exhaustive way, we shall split it into several smaller pieces called modules in order to perform the verification on each module in separation.

Notice that every state and transition of the original transition system must belong to one module at least, so that the modular split must consist of a partition for the transitions and of an overlapping for the states, that are possibly both the target of a last transition of a module and the source of a first transition of another module.

Thanks to the modular split, a property can then be verified by model-checking on each module in separation, so that it is never necessary to keep the whole transition system in memory. With such a verification, we want to be able to tell whether the property is globally true or not.

2.2 Modular Property

When a property is true on every module, it is not always possible to conclude that it is globally true. Further in this section, we look at the $PLTL$ property pattern $\Box(p \Rightarrow \Box q)$ which is possibly true on every module although globally false. Thus, we have to distinguish the properties that can be verified in a modular way and the ones that can not. We will say that a $PLTL$ property P is modular if and only if

$$P \text{ true on every module } \Rightarrow P \text{ globally true.}$$

We will give a formal definition of a modular property in definition 4.

Thus, given a specification with dynamic properties in $PLTL$, we first have to make sure that these properties are modular before we verify them in a modular way. Section 3.3 shows that the three $PLTL$ property patterns $\Box(p \Rightarrow \bigcirc q)$,

$\Box(p \Rightarrow \Diamond q)$ and $\Box(p \Rightarrow q\mathcal{U}r)$, where p, q and r are propositional formulae over the states of a transition system, are modular.

Notice that

$$P \text{ true on every module } \Rightarrow P \text{ globally true}$$

is equivalent to

$$P \text{ globally false } \Rightarrow P \text{ false on one module at least.}$$

Thus, in order to detect the modularity of a property, we will suppose that this property is globally false. If we can prove that in this case it also false on a module, then we know that the property is modular.

As examples, let us look at the two property patterns $\Box(p \Rightarrow \Diamond q)$ and $\Box(p \Rightarrow \Box q)$. The first pattern is modular, as will be proved in section 3.3. The second pattern is not modular. In both cases, we suppose that the properties are globally false. That is, we consider sequences that satisfy the negations of the properties. The modular split will possibly "cut" the sequences, and so we look whether the cut sequences still negate the properties or not.

A Modular Property: $\Box(p \Rightarrow \Diamond q)$ Let us suppose that the property $P = \Box(p \Rightarrow \Diamond q)$ is globally false. Then, the negation of the property is true. Notice that $\neg P$ is equivalent to $\Diamond(p \wedge \Box\neg q)$. This means that there is a sub-sequence satisfying $p \wedge \Box\neg q$. Fig. 1 shows such a sequence, and how it is cut by a two modules split.

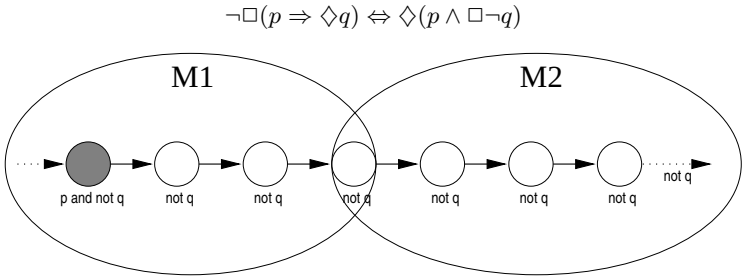

Fig. 1. Modular property $\Box(p \Rightarrow \Diamond q)$

In module M_2, there is no state satisfying p and so P is trivially true in this module. In module M_1, a state satisfies p which is followed by states all satisfying $\neg q$. As a consequence, P is indeed false in this module.

This is due to the fact that, with property pattern $\Box(p \Rightarrow \Diamond q)$, once a state satisfying p has occurred, a response (in the shape of a state satisfying q) is waited for. If this response never comes ($\Box\neg p$), one may cut the sequence, it still will not come.

A Non-modular Property: $\Box(p \Rightarrow \Box q)$ Let us suppose that the property $P = \Box(p \Rightarrow \Box q)$ is globally false. Then, the negation of the property is true. Notice that $\neg P$ is equivalent to $\Diamond(p \wedge \Diamond\neg q)$. This means that there is a subsequence satisfying $p \wedge \Diamond\neg q$. Fig. 2 shows such a sequence, and how it is cut by a two modules split.

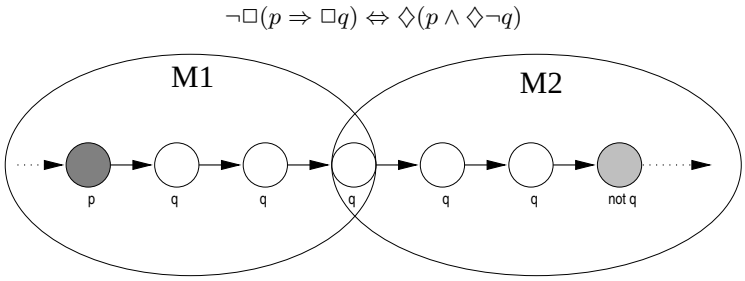

Fig. 2. Non-modular property $\Box(p \Rightarrow \Box q)$

In module M_2, there is no state satisfying p and so P is trivially true in this module. In module M_1, the state satisfying p is now followed by states all satisfying q, so that this cut sequence no longer negates P. As a consequence, P is possibly *true* in that module as well, although it is globally false.

This is due to the fact that, once a state satisfying p has occurred, the sequence remains possibly true as long as a state satisfying $\neg q$ has not occurred. If the sequence is cut before that state, then it possibly satisfies P.

3 Verification of the Modularity

Before we perform a modular verification, we have to prove that the dynamic properties that we want to verify are modular.

In this section, we are looking at a class of dynamic properties defined from the Büchi automata [20] that recognize their negation. We call this class BA_2. We prove that every property in the class BA_2 is a modular property.

As examples of properties that belong to the class BA_2, we prove that, amongst others, the dynamic properties suggested by the three patterns of dynamic constraints introduced by J.-R. Abrial in [2] are modular. They consist in the three following patterns of $PLTL$ properties: $\Box(p \Rightarrow \bigcirc q)$, $\Box(p \Rightarrow \Diamond q)$ and $\Box(p \Rightarrow p\mathcal{U}q)$ where p and q are propositional formulae over the states of a transition system.

Let P be a $PLTL$ property. We have ever noticed that we prove P modular equivalently by proving that *if P is false on the whole transition system, then it is false on one module at least.* For this we consider the Büchi automaton that recognizes the negation of P. We know that every path σ in the whole graph that

negates the property is recognized by this automaton. We then prove that there is a part of such a path σ that belongs to one module and that is recognized by the Büchi automaton. This proves that the property is false on that module.

3.1 Definitions for the Verification of the Modularity of Dynamic Properties

In this section, we introduce the notations and the definitions that we use to establish that the three above patterns are modular.

Definition 1 (Transition System) *A transition system is a 6-tuple* $TS = < A, S, S_0, \rightarrow, AP, L >$ *where* A *is an* alphabet *labelling the transitions,* S *is the set of* states, $S_0 \subseteq S$ *is the set of* initial states, $\rightarrow$ *is the* transition relation $(\rightarrow \subseteq S \times A \times S)$, AP *is a set of* propositions *and* $L : S \mapsto 2^{AP}$ *is the* assignment function.

Let $TS =< A, S, S_0, \rightarrow, AP, L >$ be a transition system. Let us consider a partition of the set S of states. Each part of the partition can be regarded as an equivalence class EQ_i such that $S = \bigcup_i EQ_i$. Some states in EQ_i are the source states of some transitions in $\rightarrow$ whose target states are *not* in EQ_i. We call these transitions the *exiting transitions* of EQ_i. A module M of a transition system TS is a transition system whose set of states is composed of the states in EQ_i augmented with the target states of the exiting transitions.

Definition 2 (Module) *Let* $TS =< A, S, S_0, \rightarrow, AP, L >$ *be a transition system. Let* $\mathcal{P}_S$ *be a partition of* S*. Each equivalence class* EQ *of* $\mathcal{P}_S$ *allows the definition of a* module *which is a transition system* $TS' =< A, S', S'_0, \rightarrow', AP, L' >$ *defined in the following way:*

- $S' = EQ \cup \{s \in S / \exists s_1 \xrightarrow{a} s \in \rightarrow \wedge s_1 \in S_1 \wedge s \notin S_1\}$
  S' *is the set of states of the class augmented with the exiting states,*
- $\rightarrow' = \{s_1 \xrightarrow{a} s \in \rightarrow / s_1 \in S' \wedge s \in S'\}$
  $\rightarrow'$ *is the set of transitions that link the states of* S'*,*
- $S'_0 = \{s' \in S' / \exists s \xrightarrow{a} s' \in \rightarrow \wedge s \xrightarrow{a} s' \notin \rightarrow'\}$
  S'_0 *is the set of states of* S' *reachable from a transition in* $\rightarrow$ *that is not in* $\rightarrow'$*,*
- L' *is the restriction of* L *on* S'*.*

We call *internal states* the states of the modules that are in EQ, whereas we call *exiting states* the target states of the exiting transitions. With such a definition, we have indeed a partition of the transitions of TS and an overlapping of the states of TS.

Notations 3 *Let* P *be a dynamic property in* $PLTL$ *and* TS *a transition system. We denote* $TS \vdash P$ *the fact that the property* P *is satisfied on* TS*. Its meaning is defined by the semantics of* $PLTL$*. We denote* $s \models p$ *the fact that a state* s *satisfies a boolean proposition* p*. Thus, we have* $s \models p$ *iff* $p \in L(s)$*.*

Given a $PLTL$ property P, a transition system TS and its split into a set $\mathcal{M}$ of modules M, we want to prove that P is modular.

Definition 4 (Modular Property) *Let P be a $PLTL$ property. Let $\mathcal{M}$ be the split of a transition system TS into modules. The property P is modular iff*

$$(\forall M \in \mathcal{M}, M \vdash P) \Rightarrow TS \vdash P$$

which is equivalent to

$$TS \nvdash P \Rightarrow \exists M \in \mathcal{M} \wedge \sigma \text{ path of } M \text{ s.t. } \sigma \vdash \neg P \ .$$

Definition 5 (Path of a Transition System) *A path $\sigma = s_0 s_1 \dots s_i \dots$ of a transition system $TS =< A, S, S_0, \rightarrow, AP, L >$ is defined as a sequence (finite or infinite) of states linked by a transition.*

$$\forall s_i \in \sigma, s_{i+1} \in \sigma, \exists t \in \rightarrow \ \text{ s.t. } t = s_i \rightarrow s_{i+1} \ .$$

Definition 6 (Büchi Automaton) *A Büchi automaton is a 5-tuple $\mathcal{B} = < b_0, B, AP, \rightarrow_{\mathcal{B}}, Accept >$ where:*

- *b_0 is the initial state,*
- *B is the finite set of states ($b_0 \in B$),*
- *AP is a set of boolean propositions (the labels of the transitions),*
- *$\rightarrow_{\mathcal{B}}$ is the finite set of transitions labeled by propositions of $P_{\mathcal{B}}$: $\rightarrow_{\mathcal{B}} \subseteq B \times AP \times B$,*
- *$Accept \subseteq B$ is the set of* accepting states *of the automaton.*

Remark. A Büchi automaton is defined as a transition system in which we distinguish accepting states, and whose transitions are labeled with boolean propositions. As the states of a Büchi automaton are not valued, we need not introduce the assignment function L in its definition.

Definition 7 (Recognition of a Finite Path) *A finite path $\sigma = s_0 s_1 \dots s_n$ of a transition system $TS =< A, S, S_0, \rightarrow, AP, L >$ is recognized by a path $\tau = b_0 b_1 \dots b_{n+1}$ of a Büchi automaton $\mathcal{B} =< b_0, B, AP, \rightarrow_{\mathcal{B}}, Accept >$ if and only if*

i) $\forall i, 0 \leq i \leq n, s_i \in \sigma \Rightarrow \exists p_i \in AP \text{ s.t. } b_i \xrightarrow{p_i} b_{i+1} \in T_{\mathcal{B}} \text{and } s_i \models p_i$
ii) $b_{n+1} \in Accept$

Definition 8 (Recognition of an Infinite Path) *An infinite path $\sigma = s_0 s_1 \dots$ of a transition system is recognized by a path $\tau = b_0 b_1 \dots$ of a Büchi automaton if and only if*

i) $\forall i \geq 0, s_i \in \sigma \Rightarrow \exists p_i \in AP \text{ s.t. } b_i \xrightarrow{p_i} b_{i+1} \in T_{\mathcal{B}} \text{and } s_i \models p_i$
ii) accepting states appear infinitely often in τ.

3.2 Intuitive Presentation of the Demonstration of the Modularity Verification Correctness

Given a $PLTL$ property P and a transition system TS split into a set $\mathcal{M}$ of modules M, we want to prove that:

$$TS \nvdash P \Rightarrow \exists M \in \mathcal{M} \wedge \sigma \text{ path of } M \text{ s.t. } \sigma \vdash \neg P \ .$$

For a path σ of the whole transition system that negates P, we prove that a part of σ exists that is entirely within a module and that negates P as well. In order to do this, we isolate in σ the suffix that is sufficient to negate P (we call it the *minimal suffix of the negation of* P and we prove that every prefix of this suffix still negates P. As a consequence, σ can be "cut" anywhere by a module, it still negates P.

Let us now justify the usefulness of this notion of minimal suffix of the negation of P. Let σ be a path of the whole graph that negates P: $\sigma \vdash \neg P$. The path σ possibly contains useless information as for the negation of P. Consider as an example the following property P: $\Box(p \Rightarrow \Diamond q)$. A path including a state s_k satisfying $p \wedge \neg q$ followed by states all satisfying $\neg q$ is a path that satisfies $\neg P$, whatever the states preceding s_k. So, we are only interested in the part of the path that starts with state s_k. Thus, given P and σ, we consider a suffix σ_m, which is the suffix of σ still satisfying $\neg P$ obtained by removing from σ the longest possible prefix. This suffix σ_m is called the *minimal suffix of negation of* P.

Definition 9 (Minimal Suffix of Negation of a Property) *Let P be a $PLTL$ property and let $\sigma = s_0 s_1 \dots s_i \dots$ be a path of a transition system such that $\sigma \vdash \neg P$. We denote σ_i the suffix $s_i s_{i+1} \dots$ of σ.*

If an integer m exists such that: $\forall i \in [0 \dots m], \sigma_i \vdash \neg P$ and $\forall i > m, \neg(\sigma_i \vdash \neg P)$, then the suffix σ_m is called the minimal suffix of negation of P.

Two situations have to be considered.

1. the states involved in σ_m all belong to a unique module M. Thus, $M \vdash \neg P$
2. the states involved in σ_m belong to distinct modules. We consider in this case a prefix $\sigma'_m = s_m s_{m+1} \dots s_k$ such that all the states involved in σ'_m belong to the same module. Thus, we have to prove that $\sigma'_m \vdash \neg P$.

3.3 Demonstration of the Modularity of the $PLTL$ Properties Encoded by a Büchi Automaton in the Class BA_2

Let us consider a class of Büchi automata that we call BA_2, for which all states of the automata are accepting states, with the exception of at most the initial state and its direct successors, provided that there is no transition between any of such (non-accepting) direct successors. In other words, every path $\tau = b_0 b_1 b_2 \dots$ of an automaton in the class BA_2 that recognizes the minimal suffix of negation of a property is such that $\forall i \geq 2, b_i \in Accept$.

Definition 10 (The Class BA_2) *Let $\mathcal{B} =< b_0, B, AP, \rightarrow_{\mathcal{B}}, Accept >$ be a Büchi automaton. $\mathcal{B} \in BA_2$ iff*

$$\forall \tau = b_0 b_1 \ldots \text{ path of } \mathcal{B}, \exists k > 0 \text{ s.t. } \forall i, 0 \leq i < k, b_i = b_0,$$

$$\text{and } \forall j > k, b_j \in Accept \ .$$

The automata encoding the properties $\Box(p \Rightarrow \bigcirc q)$, $\Box(p \Rightarrow \Diamond q)$ and $\Box(p \Rightarrow q \mathcal{U} r)$ (represented in figures 3, 4 and 5) belong to the class BA_2. We prove that all the $PLTL$ properties that can be encoded by an automaton of the class BA_2 are modular properties.

Let P be a $PLTL$ formula, and let σ be a path of a transition system TS such that $\sigma \vdash \neg P$. Let σ_m be the minimal suffix of σ that negates P, and let BA be the Büchi automaton that encodes $\neg P$. We suppose $BA \in BA_2$. Lemma 1 proves that a prefix of σ_m whose all the states belong to the same module always exists. Moreover, it proves that this prefix is at least two states long. Lemma 2 proves that when recognizing σ_m, BA can not get back to its initial state once it has left it. In other words, we eliminate the loop on the initial state of BA (see figures 3, 4, 5). Theorem 3 concludes that if the above conditions are true, P is still negated on a module. It is obvious as BA is only composed of accepting states from the third one, and so the recognition will be "cut" on an accepting state. In other words, $BA \in BA_2$ is a sufficient condition for P being a modular property.

Lemma 1 *Let P be a $PLTL$ formula, σ a path of a transition system TS such that $\sigma \vdash \neg P$, and $\sigma_m = s_m s_{m+1} \ldots$ the minimal suffix of σ that negates P.*

A prefix σ'_m of σ_m such that all the states in σ'_m belong to the same module always exists. The prefix σ'_m is at least 2 states long.

Proof. Consider the prefix only composed of the first two states of σ_m: $\sigma'_m = s_m s_{m+1}$. s_m and s_{m+1} belong to the same module. As a matter of fact, two cases have to be considered:

1. s_m and s_{m+1} belong to the same equivalence class. Then they basically belong to the same module ;
2. s_m and s_{m+1} belong to two distinct equivalence classes EQ and EQ'. Then s_{m+1} is the target state of an outgoing transition of EQ and thus s_m and s_{m+1} belong to the same module.

Lemma 2 *Let P be a $PLTL$ formula, σ a path of a transition system TS such that $\sigma \vdash \neg P$, σ_m the minimal suffix of σ that negates P, and BA the Büchi automaton that encodes $\neg P$.*

$\sigma_m = s_0 s_1 \ldots$ is recognized by a path $\tau = b_0 b_1 \ldots$ of BA where initial state b_0 only occurs once.

Proof. Let $i > 0$ be an integer such that $b_i = b_0$. Then, $\sigma_i = s_i s_{i+1} \ldots$ is a suffix of σ recognized by $\tau_i = b_i b_{i+1} \ldots$. So, $\sigma_i \vdash \neg P$, which is a contradiction since σ_m is the minimal suffix of σ that negates P.

Theorem 3 *All the PLTL properties encoded by a Büchi automaton in the class BA_2 are modular properties.*

Proof. Let P be a $PLTL$ formula, and let TS be a transition system which is split into a set $\mathcal{M}$ of modules. Let σ be a path of TS such that $\sigma \vdash \neg P$, and $\sigma_m = s_0 s_1 \dots$ be the minimal suffix of σ that negates P. Let finally $BA = < b_0, B, AP, \rightarrow_{BA}, Accept >$ be a Büchi automaton such that $BA \in BA_2$.

A prefix $\sigma'_m = s_0 s_1 \dots s_k$ of σ_m exists such that all the states involved in σ'_m belong to a module $M \in \mathcal{M}$. Moreover, σ'_m is at least 2 states long (lemma 1). Let us prove that $\sigma'_m \vdash \neg P$.

We have $\sigma_m \vdash \neg P$. So, a path $\tau_m = b_0 b_1 \dots$ of BA exists that recognizes σ_m. Since $BA \in BA_2$, we have: $\forall i \leq 2, b_i \in Accept$ (lemma 2). Let us consider the two following cases:

- let $\sigma'_m = s_0 s_1$. Then $\tau'_m = b_0 b_1 b_2$ recognizes σ'_m as $b_2 \in Accept$
- let $\sigma'_m = s_0 \dots s_k$ with $k > 1$. Then $\tau'_m = b_0 \dots b_{k+1}$ recognizes σ'_m as $b_{k+1} \in Accept$.

As a consequence, $\sigma'_m \vdash \neg P$. Finally,

$$TS \nvdash P \Rightarrow \exists M \in \mathcal{M} \wedge \sigma'_m \text{ path of } M \text{ s.t. } \sigma'_m \vdash \neg P \ .$$

3.4 Modularity of the Three Modalities Introduced in B

In order to prove that $\Box(p \Rightarrow \bigcirc q)$, $\Box(p \Rightarrow \Box q)$ and $\Box(p \Rightarrow q \mathcal{U} r)$ are modular, we simply prove that the automatons of the negations of these patterns belong to BA_2.

Pattern 1: $\Box(p \Rightarrow \bigcirc q)$ Suppose this property is false on the global graph. Then a path σ exists such that $\sigma \vdash \neg\Box(p \Rightarrow \bigcirc q)$ (which is equivalent to $\sigma \vdash \Diamond(p \wedge \bigcirc \neg q)$). The Büchi automaton that encodes such a property is given in

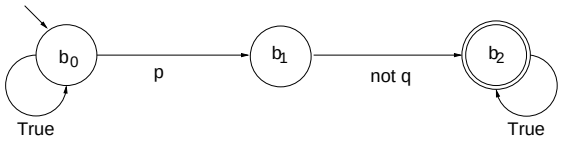

Fig. 3. Büchi automaton encoding $\neg\Box(p \Rightarrow \bigcirc q)$

Figure 3. It belongs to the BA_2 class and thus $\Box(p \Rightarrow \bigcirc q)$ is modular.

Pattern 2: $\Box(p \Rightarrow \Diamond q)$ Suppose this property is false on the global graph. Then a path σ exists such that $\sigma \vdash \neg\Box(p \Rightarrow \Diamond q)$ (which is equivalent to $\sigma \vdash \Diamond(p \wedge \Box \neg q)$). The Büchi automaton that encodes such a property is given in Figure 4. It belongs to BA_2 class and thus $\Box(p \Rightarrow \Diamond q)$ is modular.

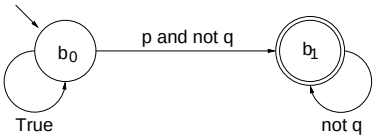

Fig. 4. Büchi automaton encoding $\neg\Box(p \Rightarrow \Diamond q)$

Pattern 3: $\Box(p \Rightarrow q\mathcal{U}r)$ Suppose this property is false on the global graph. Then a path σ exists such that $\sigma \vdash \neg\Box(p \Rightarrow q\mathcal{U}r)$ (which is equivalent to $\sigma \vdash \Diamond(p \wedge \neg(q\mathcal{U}r))$). The Büchi automaton which encodes such a property is

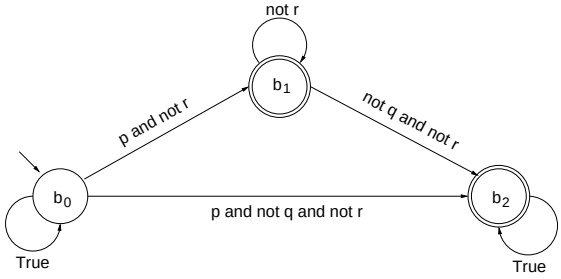

Fig. 5. Büchi automaton encoding $\neg\Box(p \Rightarrow q\mathcal{U}r)$

given in Figure 5. It belongs to the class BA_2 and therefore $\Box(p \Rightarrow q\mathcal{U}r)$ is modular.

4 Refinement and Modules

4.1 Modularity and Modules

Until now, we have only considered abstract partitions of the state space, without giving any indication on how to find a "good" partition. As a matter of fact, in order to prove in a modular way that a property P is true, P has to be true on *every* module. But with a random split of the transition system, it is possible that some modules find that P is false, whereas another split would have found P true on every module.

The question is, for a given modular property, *how can we find a partition that makes the verification module by module successful ?*

Our purpose here is to suggest a partition of the state space of a transition system into several components, this partition being based on a refinement relation. The refinement that we use is temporal in the sense that the system is observed more often. Thus, details such as new variables and new events are introduced step by step. They are observed between the old events in the abstract

specification. The new properties are concerned with sequences of news events and it is suitable to verify them only on derived subsystems. We call *old* what is concerned with the abstract specification and *new* what is concerned with the refined specification.

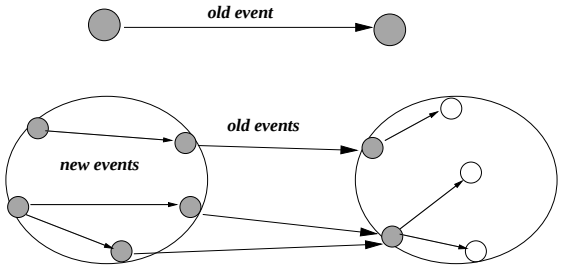

Fig. 6. Refinement of an abstract transition

Let us recall some notions given in a paper published at IFM'99 [14]. Intuitively, the figure 6 schematizes the refinement of an abstract transition t labeled with an old event into a family of refined transitions in which some conditions hold. We briefly define these notions informally.

- each labeled transition t is refined by a set of refined paths made of transitions labeled with the new events and terminated with a transition labeled by the label of t,
- the initial state of the abstract transition is refined by the initial states and all the intermediate states of the paths,
- the final state of the abstract transition is refined by the final states of all paths.

The modular verification of a $PLTL$ property P consists in

- computing the set of modules,
- verifying on each module by model-checking if P is satisfied or not,
- concluding that P is satisfied if all the modules satisfy P.

In the rest of this section we explain why the modular verification concept concerns the modules of the refined specification, rather than the full specification. The introduced dynamic properties concern the new events in the modules obtained at this step from the abstract specification. Each module is described as a transition system obtained from a state by application of the new events. In the modules, the new states are pointed in and pointed out by the old events of the abstract specification. The new transitions are labeled by the new events. The verification of the dynamic properties is only performed on this state space augmented with its immediate neighbour states, on the border. Thus, semantically, each old transition is refined by some chains of new events terminated by

a transition labeled by an old event. We establish the relation between a verification on modules and a verification on the whole system. In other words, we are essentially looking for conditions allowing us to perform local verifications rather than global ones.

4.2 *B* Events and Transitions Systems

We give some formal definitions for a B specification extended with $PLTL$ formulae, and its semantics with a labeled transition system TS. The transition system semantics of a B specification is directly obtained from the *Before/After* predicate semantics of the B event systems given in [1], for the class of finite systems.

Modules are defined as particular labeled transition systems that refine a state and the set of transitions whose this state is source in the abstract specification.

Definition 11 (Event System) *An* event system *is a 6-tuple* $ES =< V, I, F, Init, A, G_A >$ *which consists of*

- *a set* V *of variables,*
- *an invariant* I*,*
- *a set* F *of formulae,*
- *an initial action* $Init$ *to initialize variables,*
- *an alphabet* A *of label of events,*
- *a set* G_A *of events definitions in the shape* **select** g **then** a **end** *where g denotes a proposition and a is a generalized substitution as defined in the B method [1].*

Consider now two labeled transition systems $TS_1 =< A_1, S_1, S_{1_0}, \rightarrow_1, AP, L_1 >$ and $TS_2 =< A_2, S_2, S_{2_0}, \rightarrow_2, AP, L_2 >$ semantics of $ES_1 =< V_1, I_1, F_1, Init_1, A_1, G_{A_1} >$ and $ES_2 =< V_2, I_2, F_2, Init_2, A_2, G_{A_2} >$.

Definition 12 (Refinement of an Abstract Transition) *A path* σ_2 *of* TS_2 *refines an abstract transition* $t = s \overset{a}{to} s' \in \rightarrow_1$ *of* TS_1 *denoted* $\sigma_2 \sqsubseteq t$ *iff*

- $\sigma_2 = s_1 \dots s_n$ *is a finite path s.t. the transitions* $t_1 = s_0 \overset{a_1}{\rightarrow} s_1, \dots, t_{n-1} = s_{n-2} \overset{a_{n-1}}{\rightarrow} s_{n-1}$ *are labeled with new events*

$$\forall t_i = s_{i-1} \overset{a_i}{\rightarrow} s_i \in \rightarrow_2 \ \ s.t.\ 1 \leq i < n, a_i \notin A_1 \wedge a_i \in A_2 \ ,$$

- *the label of the transition* $t_n = s_{n-1} \overset{a_n}{\rightarrow} s_n$ *is the same than the one of t:* $a_n = a$*,*
- *the valuations of all the states, except the last one, of the path* σ_2*, do not contradict the valuation of the source state of t*
- *the valuation of the final state of* σ_2 *does not contradict the valuation of the target state of t*

Definition 13 (Refinement of a Transition System) *Let TS_1 and TS_2 be two labeled transition systems. We define the set Σ of paths σ_2 of LTS_2 which refine each abstract transition $t \in \rightarrow_1$ enabled from $s \in S_1$ as follows:*

$$\forall t = s \xrightarrow{a} s' \in \rightarrow_1, \forall \sigma_2 = s_0 \dots s_n \text{ path of } TS_2, \sigma_2 \sqsubseteq t \Rightarrow \sigma_2 \in \Sigma \ .$$

Definition 14 (Module Derived from the Refinement) *Let I_2 be the gluing invariant of ES_2. A module of TS_2, associated with a state $s \in S_1$ and an abstract transition $t \in \rightarrow_1$ is a labeled transition system $TS =< A, S, S_0, \rightarrow, AP, L >$where*

- $\rightarrow$ *is the set of transitions such that*

$$\forall \sigma_2 = t_1 \dots t_n \in \Sigma, \forall i \ s.t. \ 1 \leq i \leq n, t_i \in \rightarrow \ ,$$

- S *is the set of states such that*

$$S = \{s'/s' \in S_2 \text{ and } s' \wedge I_2 \Rightarrow s\} \cup$$

$$\{s''/\forall t = s_{i-1} \xrightarrow{a_i} s_i \in \rightarrow, s'' = s_{i-1} \vee s'' = s_i\} \ ,$$

- S_0 *is the set of initial states such that*

$$S_0 = \{s'/s' \in S \wedge \forall t = s_{i-1} \xrightarrow{a_i} s_i \in T, s' \neq s_i\} \ .$$

Informally, a module is a labeled transition system defining all paths σ_2 of TS_2 which refines each abstract transition t enabled from one state $s \in S_1$. Notice that the number of modules in TS_2 is the number of states in TS_1.

The specification refinement guarantees essentially three points for a labeled transition system

- the refinement of each transition of the abstract transition system by a set of refined paths,
- the connectivity between the paths of the modules with the abstract transitions,
- the modules are free from deadlocks and livelocks.

All these definitions, and the consequences of the refinement of specification are established and proved in [5].

5 Specifications of the Robot Example

In order to make these notions clear, we illustrate them with the example of an industrial robot composed of an *Arrival Device*, an *Exit Device* and a *Carrier Device*. This robot carries some parts by moving from the Arrival Device to the Exit Device. We first give an abstract specification and then we give two refinement levels of the system. The indices 0, 1 and 2 of the variables indicate the level of refinement.

5.1 Informal Specification

The figure 7 represents the physical system. The Carrier Device (called CD) takes a part from the Arrival Device (called AD) and places it on the Exit Device (called ED). We first describe it as an abstract system, considering only two operations to load and unload a part. It simply consists in the Carrier Device taking a part and putting it onto the Exit Device.

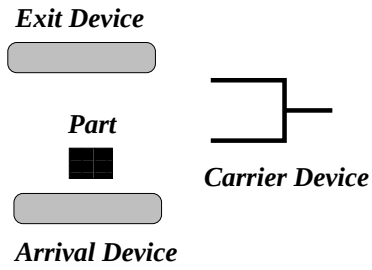

Fig. 7. The robot

Informal Presentation of an Extended *B* Specification We describe a specification as an abstract system as in B[1]. It is composed of two parts.

- a descriptive specification that indicates what the system does,
- an operational specification that indicates how the system proceeds.

The descriptive specification is essentially composed of a list of variables and of an invariant expressing safety properties restricting the set of valid states. In order to complete this part, we have proposed an extension of those systems with dynamic properties such as liveness, in the shape of $PLTL$ formulae. In the example produced below, we use the four future temporal operators called usually *Always, Next, Eventually* and *Until*, denoted respectively by the symbols $\Box$, $\bigcirc$, $\Diamond$ and $\mathcal{U}$.

The operational specification is also composed of two parts.

- the description of the initial states in the *Initialization* section,
- the description of all the events in the shape of guarded actions. The semantic of such an event is that it is enabled when the guard is true, and in that case the action is performed, so that it transforms the state of the system.

The verification phase makes sure that both invariant and dynamic properties hold with the operational specification. As in the B method, the invariant is proved by means of a theorem-prover, but we will use a model-checker in order to verify the dynamic properties on the derived transition systems.

Abstract Specification In this first approach of the system, we focalize our observation on one component and we only look at the behaviour of the Carrier Device. We ignore what happens to the other devices. There are only two states: one when the device is free and one when it is busy. The specification proposed in figure 8 uses only one variable that gives the status of the Carrier Device denoted $CD_0 \in \{free, busy\}$.The invariant property gives the sorting of the variables. Two events are introduced in order to load and unload the part.

System	$Robot_0$
Variables	$V_0 \hat{=} \{CD_0\}$
Invariant	$I_0 \hat{=} CD_0 \in \{free, busy\}$
Dynamic properties	$P_{0_1} \hat{=} \Box(CD_0 = busy \Rightarrow \Diamond CD_0 = free)$
Initialization	$Init_0 \hat{=} CD_0 := free$
Events	$Load_0 \hat{=}$ Select $CD_0 = free$ then $CD_0 := busy$ end
	$Unload_0 \hat{=}$ Select $CD_0 = busy$ then $CD_0 := free$ end
end	$Robot_0$

Fig. 8. Abstract specification of the robot

The dynamic property P_{0_1} is a liveness property that states that when the Carrier Device is busy, then eventually it becomes free.

The corresponding transition system is represented in figure 9. It has two states and two events. It represents all the execution paths by means of a finite state system [3]. In this case, there is only one variable with two possible values. The transitions are labeled with the events *Load* and *Unload* representing the evolution of the system from one state to another.

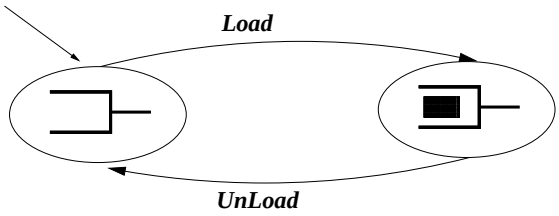

Fig. 9. The abstract transition system

5.2 Refined Specification

We now present a possible refined specification of the specification given above.

Informal Presentation of a Refinement of Specification As in the B method [1], we use the concept of refinement. The idea is to enhance the detail level of observation of the system, so that the refined specification gives further details on what the system does and how it proceeds. Thus, refinement after refinement, the description will go from a high level specification to a specification that can be directly implemented. The refined specification contains the same sections than the abstract one: *Variables, Invariant, Dynamic Properties, Initialization* and *Events*.

The figure 10 gives a refined specification of the Robot. Both abstract and refined specifications must respect the following constraints:

- the two sets of variables are disjoint,
- the new invariant, also called the gluing invariant, expresses a relation between the two sets of variables of the two levels,
- the dynamic properties are new properties concerning the behaviour induced by the new events,
- the old events are described again in such a way that they are refined,
- the new events are introduced.

In the B method, the refinement relation is verified and warrants the abstract invariant properties. The old dynamic properties are preserved by refinement notion as defined in [2]. This question is not treated in this paper in which we consider only the verification of new dynamic properties.

First Refinement of the Robot The refined specification in figure 10 settles the following point: the Carrier Device moves into two directions denoted Up and $Down$. The Up movement specifies that the Carrier Device is busy and goes up to the Exit Device in order to put a part. The $Down$ movement specifies that it goes to the Arrival Device in order to take another part. We only need one new variable $PosCD_1 \in \{Down, Up\}$ which statues the position of the Carrier Device.

The variables of the levels 0 and 1 are glued by the gluing invariant. The old events are refined with the new constraints and keep the same label. The guards of the events are reinforced. We have two new properties for this specification and they concern only the movement of the Carrier Device. They are safety properties. The properties P_{1_1} and P_{1_2} ensure that when the Carrier Device goes up it is busy, and when it goes down it is free. The transition system associated to this first refinement of the specification is given in figure 11.

Second Refinement of the Robot From this specification and with the same process, we give another refinement through a third detailed specification by introducing the component *Exit Device*. When a part is in it, it can be removed. Two news variables ED_2 and AD_2 are introduced with the possible values *free* or *busy*. Once again, the old events are refined and the guards are reinforced. Two new events are introduced, denoted $ArrivalPart_2$ and $Evac_2$. The first one expresses that when a part is on the Arrival Device, the Carrier Device

System	$Robot_1$ refines $Robot_0$
Variables	$V_1 \hat{=} \{CD_1, PosCD_1\}$
Invariant	$I_1 \hat{=} \{CD_1 = CD_0 \wedge PosCD_1 \in \{Down, Up\}\}$
New Dynamic Properties	$P_{1_1} \hat{=} \Box(PosCD_1 = Down \wedge \bigcirc(PosCD_1 = Up)$
	$\Rightarrow CD_1 = busy)$
	$P_{1_2} \hat{=} \Box(PosCD_1 = Up \wedge \bigcirc(PosCD_1 = Down)$
	$\Rightarrow CD_1 = free)$
Initialization	$Init_1 \hat{=} CD_1 := free \wedge PosCD_1 := Down$
Old Events	
$Load_1 \hat{=}$	Select $CD_1 = free \wedge PosCD_1 = Down$
	then $CD_1 := busy$ end
$Unload_1 \hat{=}$	Select $CD_1 = busy \wedge PosCD_1 = Up$
	then $CD_1 := free$ end
New Events	
$StopUp_1 \hat{=}$	Select $CD_1 = busy \wedge PosCD_1 = Down$
	then $PosCD_1 := Up$ end
$StopDown_1 \hat{=}$	Select $CD_1 = free \wedge PosCD_1 = Up$
	then $PosCD_1 := Down$ end
End	$Robot_1$

Fig. 10. First refinement of the robot specification

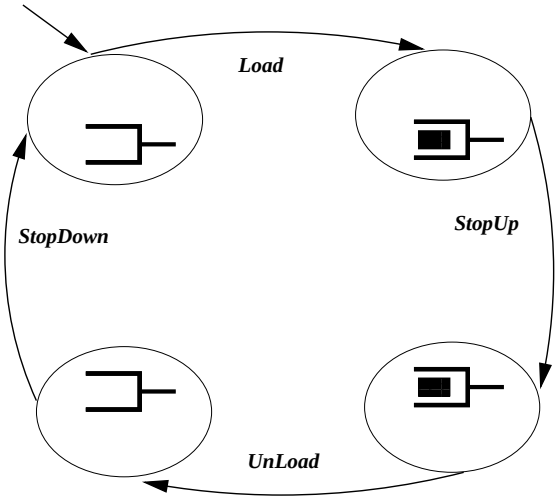

Fig. 11. The first refinement transition system

transports it to the Exit Device. The second event removes the part from the Exit Device. The figure 12 gives the obtained specification.

System	$Robot_2$ refines $Robot_1$
Variables	$V_2 \hat{=} \{CD_2, PosCD_2, AD_2, ED_2\}$
Invariant	$I_2 \hat{=} \{CD_2 = CD_1 \wedge PosCD_2 = PosCD_1$
	$\wedge AD_2 \in \{free, busy\} \wedge ED_2 \in \{free, busy\}\}$
New Dynamic properties	$P_{2_1} \hat{=} \Box(CD_2 = busy \Rightarrow CD_2 = busy$
	$\mathcal{U}\ (CD_2 = busy \wedge ED_2 = free))$
Initialization	$Init_2 \hat{=} CD_2 := free \wedge PosCD_2 := Down$
Old Events	
$Load_2 \hat{=}$	Select $CD_2 = free \wedge PosCD_2 = Down$
	then $CD_2 := busy$ end
$Unload_2 \hat{=}$	Select $CD_2 = busy \wedge PosCD_2 = Up$
	then $CD_2 := free$ end
$StopUp_2 \hat{=}$	select $CD_2 = busy \wedge PosCD_2 = Down$
	then $PosCD_2 := Up$ end
$StopDown_2 \hat{=}$	Select $CD_2 = free \wedge PosCD_2 = Up$
	then $PosCD_2 := Down$ end
New Events	
$PartArrival_2 \hat{=}$	Select $AD_2 = free$ then $AD_2 := busy$ end
$Evac_2 \hat{=}$	Select $ED_2 = busy$ then $ED_2 := free$ end
End	$Robot_2$

Fig. 12. The second refinement specification

The gluing invariant states that the variables are identical to their abstraction. The dynamic property P_{2_1} states that the Carrier Device remains busy until the Exit Device is free. The labeled transition system given in figure 13 refines the labeled transition system of figure 11. It is divided into 4 modules. The modules M_1' and M_3' both contain 6 states and 6 transitions, and the modules M_2' and M_4' both contain 8 states and 8 transitions.

5.3 Verification of the Properties

We summarize the results of the modular verification of the properties P_{0_1}, P_{1_1}, P_{1_2} and P_{2_1} presented above. These 4 dynamic properties have the Büchi automata of their negations in BA_2 and so they are modular properties. It is sufficient that we verify them on each module in separation. They can be verified by a simple model-checking.

- $P_{0_1} \hat{=} \Box(CD_0 = busy \Rightarrow \Diamond CD_0 = free)$ is satisfied in the abstract specification at the level 0.
- $P_{1_1} \hat{=} \Box(PosCD_1 = Down \wedge \bigcirc(PosCD_1 = Up) \Rightarrow CD_1 = busy)$ from level 1 is satisfied in M_2. In the module M_1, there is only one state such

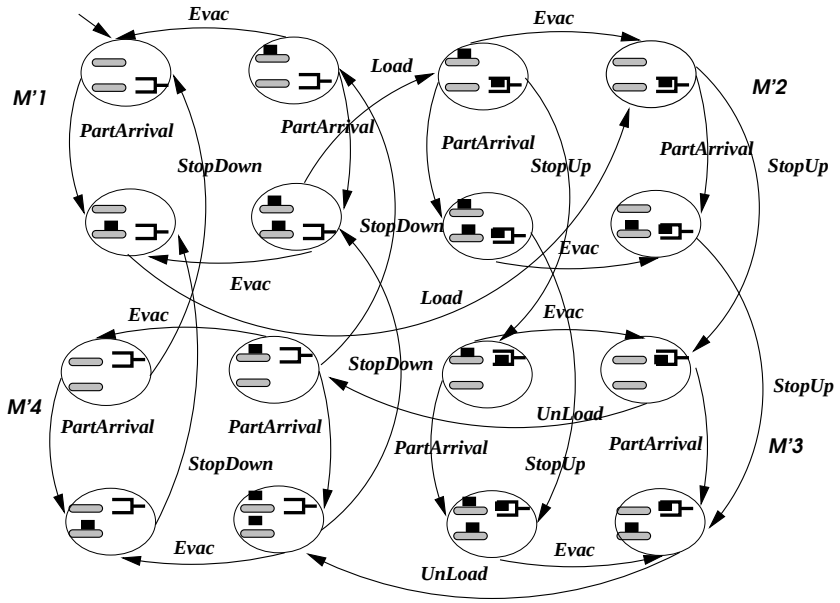

Fig. 13. The second refined transition system

that $PosCD_1 = Down$ and there is no successor. We then conclude that $PosCD_1 = Down \wedge \bigcirc(PosCD_1 = Up)$ is false and globally P_{1_1} is satisfied.

- $P_{1_2} \hat{=} \Box(PosCD_1 = Up \wedge \bigcirc(PosCD_1 = Down) \Rightarrow CD_1 = free)$ from level 1 is satisfied in M_1. We use the same reasoning than P_{1_1}.
- $P_{2_1} \hat{=} \Box(CD_2 = busy \Rightarrow CD_2 = busy \mathcal{U}(CD_2 = busy \wedge ED_2 = free))$ defined on level 2 is satisfied modularly on M'_2 and on M'_3. In the other modules M'_1 and M'_2 we have $\Box(CD_2 = free)$. We then conclude that P_{2_1} is globally satisfied.

6 Conclusion

We propose a technique that allows the modular verification of a class of usual dynamic properties of reactive systems. These properties are safety and liveness properties, and they are checked by means of a model-checker. The main problem in applying this technique based on reachability analysis is the potential combinatorial explosion of the state space. In order to attack this problem, we define a strategy based on local verifications which uses the refinement process. From the abstract specification of a B event system, we derive a detailed specification by introducing news events. We then look at the new properties that have to be checked on the chains of new events. These properties are verified on modules rather than on the whole transition system, and consequently the verification of a $PLTL$ property reduces to the verification of a local $PLTL$ property.

For some specific dynamic property patterns, we have exhibited sufficient conditions to assert that if a $PLTL$ property is satisfied on each module, then it is globally satisfied. The negations of these modular properties are recognized by particular Büchi automata that we describe. However, in some cases, if the verification of the property fails on a module, we cannot conclude about the satisfaction of this property. This situation is possibly due to the fact that the property should have been established at an abstract level of abstraction because it uses a chain of old events. This point has already been discussed and published at the Conference IFM'99 in York [14].

We have shown in [19] another possibility to reduce the complexity of verification by model-checking for large systems which consists of a combination of model-checking and theorem-proving techniques. Patterns in the shape of $\Box(p \Rightarrow Q)$ can be studied in two verifications. Invariance properties $\Box\neg p$ can be verified with the prover and temporal properties $\Diamond p \wedge \Box(p \Rightarrow Q)$ with a model-checker. Then, there are two kinds of modules: those in which we have $\Box\neg p$ and those in which some states verify p. In the first case, the B prover is the proper tool for proving$\Box\neg p$. Thus, the modules are treated separately without communication. In practice, we are working at the implementation of a verification tool using both the B and *SPIN* environments.

The work that we present in this paper only uses the syntactic way to characterize the modular dynamic properties, without taking into account their semantics. The modular class is formed of patterns in the shape of $\Box(p \Rightarrow Q)$, where p is a proposition and Q is a temporal formula such as $\bigcirc q$, $\Diamond q$, and $q\mathcal{U}r$. We project in future research to extend this class of properties by using the semantics level of the specification. We know that a module is exited with transitions that correspond to the occurrence of old events. The idea is to use this information in order to prove that some properties (not in BA_2) are modular in the very context of such a semantic modular split. This class of $PLTL$ properties could combine many different temporal operators.

References

1. J.-R. Abrial. *The B Book : Assigning Programs to Meanings.* ISBN 0521-496195. Cambridge University Press, 1996.
2. J. R. Abrial and L. Mussat. Introducing dynamic constraints in *b*. In *Second Conference on the B method, France*, LNCS 1393, pages 83–128. Springer Verlag, April 1998.
3. A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants.* Masson, 1992.
4. A. Arnold and S. Brlek. Automatic verification of properties in transition systems. *Software-Practice and Experience*, 25(6):579–596, 1995.
5. 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.

6. C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.
7. Cuéllar, I. Wildgruber, and D. Barnard. Combining the design of industrial systems with effective verification techniques. In *FME'94*, LNCS n. 873, pages 639–658. Springer Verlag, 1994.
8. P. Godefroid. Partial-order methods for the verification of concurrent systems. *LNCS*, 1032, 1996.
9. P. Godefroid and G.-J. Holzmann. On the verification of temporal properties. In *PSTV'93*, June 1993.
10. G.-J. Holzmann. *Design and validation of computer protocols*. 1991.
11. G.-J. Holzmann. The model checker spin. In *IEEE Trans. On Software Engineering*, volume 23, 1996.
12. G.-J. Holzmann. State compression in *spin*. In 3^{rd} *SPIN Workshop*, Twente University, April 1997.
13. H. Hungar. Combining model checking and theorem proving to verify parallel processes. In C. Courcoubetis, editor, *5th International Conference on Computer Aided Verification: CAV'93*, number 697 in LNCS, Elounda, June/July 1993.
14. J. Julliand, P.A. Masson, and H. Mountassir. Modular verification of dynamic properties for reactive systems. In *International Workshop on Integrated Formal Methods (IFM'99)*, pages 89–108, York, UK, June 1999. Springer Verlag.
15. K. Laster and O. Grumberg. Modular model-checking of software. In *TACAS'98, Lisbon*, March-April 1998.
16. Z. Manna and A. Pnuelli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. ISBN 0-387-97664-7. Springer-Verlag, 1992.
17. Z. Manna and A. Pnuelli. *Temporal verification of reactive systems*. ISBN 0-387-94459-1. Springer Verlag, 1995.
18. R. Milner. *Communication and Concurrency*. Computer Science. Prentice-Hall, 1989.
19. H. Mountassir, F. Bellegarde, J. Julliand, and P.A. Masson. Coopération entre preuve et model-checking pour vérifier des propriétés LTL. In *congrès AFADL'2000, Grenoble*, December 2000.
20. D. Peled and W. Penczeh. Using asynchronous büchi automata for efficient verification of concurrent systems. In *Symposium on Protocol Specification Testing and Verification*, pages 90–100, Warsaw, Pologne, June 1995.
21. D. A. Peled. Combining partial order reduction with on-the-fly model-checking. In *CAV'94*, LNCS n. 818, pages 377–390. Springer Verlag, June 1994.