Information and Computation 207 (2009)681–698

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

Partitioned PLTL model-checking for refined transition systems

J. Julliand, P.-A. Masson *, E. Oudot

LIFC – Laboratoire d'Informatique de l'Université de Franche-Comté 16, route de Gray, 25 030 Besancon Cedex, France

ARTICLE INFO

Article history:
Received 24 June 2005
Revised 17 July 2007
Available online 1 March 2009

Keywords:
Refinement
PLTL model-checking
State space partitioning

ABSTRACT

This paper is about the verification of dynamic properties by model-checking for finite state reactive systems. Properties are expressed as PLTL formulae. Systems are specified through a top-down refinement process. In order to cope with the state explosion problem, we propose partitioning the state space to be verified and to verify the properties independently on each part. Properties that are such that if they hold on every part then they hold for the whole system are called verifiable by parts.

In a previous paper, we presented a class of interesting PLTL properties that are always verifiable by parts. That is, they are verifiable by parts with any partitioning of the state space. In addition to these properties, some properties are verifiable by parts on a system provided with a particular partitioning.

In this paper, we propose a partitioning of the state space of a system that is guided by the refinement process. We introduce an extended class of PLTL properties that are verifiable by parts with regard to this partitioning. This class includes the first one. In particular, the new class includes liveness properties under fairness assumptions. This class is defined from Büchi automata that accept the language of the negations of the properties.

Our work is illustrated by its application to a chip card protocol called $T = 1$. This protocol is specified through successive refinements.

1. Motivations

Refinement is a specification method that aims to produce reliable software. A way to get into a system that is complex is to consider it first globally, with no details (as seen from the sky), and then to gradually get a more precise view by looking at it more and more closely. This is the approach that is considered when a system is specified through a top-down refinement process [13]. The specifier first gives an abstract specification of how the system works. Then, step by step, he introduces new operational details that were "hidden" at the previous level of specification. Each specification level is a refinement of the previous one. At the end of the process, the specifier gives a specification that is precise enough to be directly implemented.

In this paper, we take the specification by refinement as a context. We specify reactive systems as transition systems. We want to verify dynamic properties on these systems. In particular, we aim at verifying properties of safety and liveness with fairness assumptions. We propose to provide a set of new dynamic properties at every level of the refinement. These properties are the ones that have to be verified at this level of specification. They could not have been verified on the previous levels because they are concerned with details that were previously "hidden". For this verification method to be useful in practice, properties must be preserved by the refinement. That is, if a property holds at a given level of specification, then it must also hold on all future refinements of it (compatibility with the refinement).

* Corresponding author.
E-mail addresses: julliand@lifc.univ-fcomte.fr (J. Julliand), masson@lifc.univ-fcomte.fr (P.-A. Masson), oudot@lifc.univ-fcomte.fr (E. Oudot).
URL: http://lifc.univ-fcomte.fr

0890-5401/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2009.02.001

The dynamic properties expressed as formulae of Propositional Linear Temporal Logic (PLTL) [21] are compatible with the refinement [10]. We verify them by model-checking [24,6,8]. It is well known that the main drawback of PLTL model-checking [20,27] is that it cannot handle very large finite state systems. This problem is known as the exponential blow up of state space. To deal with this problem, many solutions have been proposed, such as partial order techniques [17, 29], abstraction techniques [9,7,12], modular techniques [14,19,2], symbolic representations by BDD [4,23], and SAT-based methods [3]. For a class of PLTL properties, we have proposed [22,16] another solution, which can be used in association with the previous ones. We have called this method *verification by parts*.

Verification by parts is an out-of-core [26] model-checking technique.[1] A transition system expresses the semantics of the system that we want to verify. Verifying this system by parts consists of partitioning the transition system into several parts, and verifying each part independently from the others. As every part is verified separately from the others, the other parts can be stored on disks while the part of interest is in the main memory.

We say that a property is verifiable by parts if, when it holds on every part, then it also holds on the whole transition system. Of course, verification by parts applies only to properties that are verifiable by parts. In [22,16], we have showed that some PLTL properties are always verifiable by parts, independently from the way the transition system is partitioned into parts. To decide if a property φ is (always) verifiable by parts, we have given a sufficient condition C on the Büchi automaton that accepts the ω-language of $\neg\varphi$. C is expressed as syntactic and propositional conditions on the Büchi automata. Safety and liveness properties such as $\Box\,(p \Rightarrow \bigcirc\, q)$, $\Box\,(p \Rightarrow \Diamond\, q)$ and $\Box\,(p \Rightarrow q\,\mathcal{U}\,r)$ are (always) verifiable by parts.

Now some PLTL properties are not, at least not always, verifiable by parts. In particular, if φ is a PLTL property and if f is the expression of a fairness assumption (f is a PLTL formula), then in general $f \Rightarrow \varphi$ is not always verifiable by parts.

In this paper, we state that a property does not have to be *always* verifiable by parts to be verified by parts. As a matter of fact, once the choice of a particular partitioning $\mathcal{P}$ of the transition system has been made, it does not matter that a property φ is not verifiable by parts with a partitioning different from the one that has been chosen. The fact that φ is verifiable by parts *with regard to* $\mathcal{P}$ is enough to verify φ by parts.

By doing so, we extend the class of the properties that are verifiable by parts by adding properties that are verifiable by parts only in the context of a given partitioning of the original transition system. In this paper, we propose a partitioning of the transition system that is based on the refinement process. We call it *refinement based partitioning*. For this, we express the semantics of the refinement as a relation between two transition systems (the abstract and the refined one), as we have proposed in [1].

We exhibit sufficient conditions to decide if a property φ is verifiable by parts with regard to a refinement based partitioning. These conditions are expressed from:

- the Büchi automaton that accepts the ω-language of $\neg\varphi$,
- the transitions of the system, as they appear in the refined transition system.

We show that the properties that are always verifiable by parts satisfy the conditions as well.

Such conditions allow for example to extend the method of verification by parts to PLTL properties expressed with fairness assumptions.

In Section 2, we review some background on transition systems and on PLTL and Büchi automata. Refinement is presented in Section 3. In Section 4, we present the partitioned model-checking technique. Section 5 extends the class of the properties verifiable by parts by considering that the transition systems are partitioned according to the refinement. Throughout the paper, the method is illustrated through the example of a chip card protocol ($T = 1$) [11]. Finally, we compare our method to related works and we present some possible future extensions in Section 6.

2. Preliminary definitions

In this section, we give formal definitions for transition systems, PLTL properties and their validity on executions of transition systems. Note that the notations we give in this section will be used throughout the whole paper.

2.1. Transition systems

Assume that V is a finite set of variables v with their respective finite domain D_v. Let $AP_V \stackrel{\text{def}}{=} \{v = d \mid v \in V \wedge d \in D_v\}$ be the set of atomic propositions over V. Let SP_V be a set of state propositions defined by the grammar

$$p ::= ap \mid p \vee p \mid \neg p \quad \text{where } ap \in AP_V.$$

Definition 1 (*Transition systems*)**.** A transition system $\mathrm{TS} \stackrel{\text{def}}{=} \langle S_0, S, A, T, \mathcal{L}\rangle$ interpreted over V has a set of initial states S_0 included in a finite set of states S, an alphabet A of labels, a labelled transition relation $T \subseteq S \times A \times S$ that must be total, and a state labelling function $\mathcal{L} : S \to 2^{AP_V}$.

[1] The idea of out-of-core algorithms is to store on disks data structures that are too large to fit in the main memory.

A label in A is the name of an *action* that modifies the state of the system. We consider transition systems that are labelled and interpreted. A transition (s,a,s') of T is written as $s \xrightarrow{a} s'$ and is labelled by an action a of A. The transition system is interpreted[2] as every state is decorated with a set of atomic propositions by means of function $\mathcal{L}$.

Remark 2. As the transition relation is total, there can be no deadlock in a transition system. If a state s has no successor, a transition $s \xrightarrow{\texttt{Skip}} s$ (where `Skip` does not belong to A) is added to obtain a transition system. Notice that we do not consider in practice transition systems where actions (other than `Skip`) could relate a state to itself. In other words, transitions $s \xrightarrow{a} s$ are forbidden, as they are of no interest in practice.

Definition 3 (*Validity of a state proposition*)**.** The validity of a state proposition $p \in SP_V$ on a state s of a transition system interpreted over V, written $s \models p$ (we say that *p holds on s*), is defined as

- $s \models ap$ iff $ap \in \mathcal{L}(s)$,
- $s \models p_1 \vee p_2$ iff $s \models p_1$ or $s \models p_2$,
- $s \models \neg p$ iff it is not true that $s \models p$ (written $s \not\models p$).

Definition 4 (*Execution*)**.** An execution of a transition system TS is an infinite sequence $\sigma \stackrel{\text{def}}{=} s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \cdots s_i \xrightarrow{a_i} s_{i+1} \cdots$ of pairs of states and actions such that $s_0 \in S_0$ and for every $i \geq 0$, we have $s_i \xrightarrow{a_i} s_{i+1} \in T$.

We note $Inf_s(\sigma)$ the set of states occurring infinitely often in an execution σ:

$$Inf_s(\sigma) \stackrel{\text{def}}{=} \{s \mid \sigma = s_0 \xrightarrow{a_0} s_1 \cdots s_i \xrightarrow{a_i} s_{i+1} \cdots \wedge s = s_i \text{ for infinitely many } i\}$$

We call Σ_{TS} the set of all the executions of a transition system TS. In an execution $\sigma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \cdots$, we denote by (σ,j) the state s_j, and by σ_j the suffix of σ starting in s_j.

2.2. PLTL properties

The Propositional Linear Temporal Logic (PLTL) is an extension of the propositional logic, introduced to specify properties with temporal aspects of the executions of a system. Future PLTL formulae are built with two temporal operators, '$\bigcirc$' (Next) and '$\mathcal{U}$' (Until), according to the following grammar:

$$\varphi ::= ap \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi.$$

Other operators can be used: $\Diamond\varphi \stackrel{def}{=} true\,\mathcal{U}\,\varphi$ (eventually φ), $\Box\varphi \stackrel{def}{=} \neg\Diamond\neg\varphi$ (always φ) and $\varphi_1 \,\mathcal{W}\, \varphi_2 \stackrel{\text{def}}{=} \varphi_1 \,\mathcal{U}\, \varphi_2 \vee \Box\varphi_1$ (φ_1 unless φ_2).

PLTL properties are interpreted on the executions of a transition system. The semantics of PLTL is as follows. Let φ, φ_1 and φ_2 be PLTL formulae. Let $\sigma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \cdots$ be an execution. We define that φ holds on the state s_j ($j \geq 0$) of σ, written $(\sigma,j) \models \varphi$, as

- $(\sigma,j) \models ap$ iff $ap \in \mathcal{L}(s_j)$,
- $(\sigma,j) \models \neg\varphi$ iff it is not true that $(\sigma,j) \models \varphi$, written $(\sigma,j) \not\models \varphi$,
- $(\sigma,j) \models \varphi_1 \vee \varphi_2$ iff $(\sigma,j) \models \varphi_1$ or $(\sigma,j) \models \varphi_2$,
- $(\sigma,j) \models \bigcirc\varphi$ iff $(\sigma,j+1) \models \varphi$,
- $(\sigma,j) \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff $\exists k \cdot (k \geq j \wedge (\sigma,k) \models \varphi_2 \wedge \forall i \cdot (j \leq i < k \Rightarrow (\sigma,i) \models \varphi_1))$.

When $(\sigma,0) \models \varphi$ we say that "φ holds on σ", and we write $\sigma \models \varphi$. A PLTL formula holds on a transition system TS if it holds on all the executions of TS.

A PLTL property φ defines an ω-language that is the set of all the executions on which φ holds. It is always possible to associate to a PLTL formula φ a non-deterministic Büchi automaton (see Definition 5) which recognizes the ω-language of φ [28].

Definition 5 (*Büchi automaton*)**.** Let SP_V be a set of state propositions over V. A Büchi automaton is a 5-tuple $\mathcal{B} \stackrel{\text{def}}{=} \langle q_0, Q, SP_V, T_{\mathcal{B}}, \mathcal{F}_{\mathcal{B}}\rangle$ where Q is a finite set of states ($q_0 \in Q$ is the initial state), $T_{\mathcal{B}}$ is a finite set of transitions labelled by elements of SP_V: $T_{\mathcal{B}} \subseteq Q \times SP_V \times Q$ and $\mathcal{F}_{\mathcal{B}} \subseteq Q$ is the set of *accepting states* of the automaton.

An infinite sequence $\pi = q_0 \xrightarrow{p_0} q_1 \xrightarrow{p_1} q_2 \cdots q_i \xrightarrow{p_i} \cdots$ such that $q_k \xrightarrow{p_k} q_{k+1} \in T_{\mathcal{B}}$ for $k \geq 0$, is called a *run* of $\mathcal{B}$. We denote by $\Sigma_{\mathcal{B}}$ the set of all the runs of $\mathcal{B}$. A run π of $\mathcal{B}$ is *accepting* if at least one of the accepting states appears infinitely often in the run : $Inf_s(\pi) \cap \mathcal{F}_{\mathcal{B}} \neq \emptyset$, where the notation $Inf_s(\pi)$ has the same meaning for runs and executions.

[2] This is a Kripke structure.

Definition 6 (*Synchronization of an execution and a run*)**.** Let $\sigma = s_0 \xrightarrow{a_0} s_1 \cdots s_i \xrightarrow{a_i} s_{i+1} \cdots$ be an execution of a transition system, and let $\pi = q_0 \xrightarrow{p_0} q_1 \cdots q_i \xrightarrow{p_i} q_{i+1} \cdots$ be a run of a Büchi automaton. We say that σ synchronizes with π if $\forall i (i \geq 0 \Rightarrow s_i \models p_i)$.

In the case where σ and π are finite sequences $\sigma = s_0 \xrightarrow{a_0} s_1 \cdots s_{n-1} \xrightarrow{a_{n-1}} s_n$ and $\pi = q_0 \xrightarrow{p_0} q_1 \cdots q_{n-1} \xrightarrow{p_{n-1}} q_n$, σ synchronizes with π if $\forall i (0 \leq i < n \Rightarrow s_i \models p_i)$.

For $n = 1$, we say that the transition $s \xrightarrow{a} s'$ synchronizes with $q \xrightarrow{p} q'$ if $s \models p$.

Definition 7 (*Acceptance of an execution*)**.** A run π accepts an execution σ if π is accepting and σ synchronizes with it. A Büchi automaton $\mathcal{B}$ accepts σ if there exists a run of $\mathcal{B}$ that accepts σ.

We denote by $\mathcal{B}_\varphi$ a Büchi automaton that recognizes the ω-language of φ. The set of executions satisfying φ are exactly those accepted by $\mathcal{B}_\varphi$.

3. Refinement

In this section, we consider the refinement of transition systems. We express the refinement as a relation between $\mathrm{TS}_2 \stackrel{\text{def}}{=} \langle S_{0_2}, S_2, A_2, T_2, \mathcal{L}_2 \rangle$ and $\mathrm{TS}_1 \stackrel{\text{def}}{=} \langle S_{0_1}, S_1, A_1, T_1, \mathcal{L}_1 \rangle$, which are, respectively, the refined and abstract transition systems. TS_2 and TS_1 are, respectively, interpreted over sets of variables V_2 and V_1.

Refining a transition system is achieved by:

- introducing new actions, so $A_1 \subseteq A_2$,
- introducing new variables, so that $V_1 \cap V_2 = \emptyset$,
- gluing the states of S_2 to the states of S_1, by means of a gluing predicate expressed on the variables of V_2 and V_1.

This is expressed on the transition systems by a particular kind of simulation of TS_2 by TS_1, which is a τ-simulation, as defined in [1]. The τ-transition system of TS_2 on A_1 is a transition system identical to TS_2 in which every transition label that is not in A_1 is replaced by τ. This means renaming every *new* action by τ in the refined transition system. A τ-transition is a transition labelled by τ. We say that TS_2 is a refinement of TS_1 by requiring that the τ-transition system of TS_2 on A_1 satisfy some conditions given in Section 3.2.

3.1. Gluing predicate

Consider the sets of variables V_1 and V_2 of two transition systems TS_1 and TS_2. The set $SP_{V_{12}}$ of state propositions over V_1 and V_2 is defined by the following grammar:

$$p ::= ap_1 \mid ap_2 \mid \neg p \mid p \vee p \quad \text{where } ap_1 \in AP_{V_1} \quad \text{and} \quad ap_2 \in AP_{V_2}.$$

Definition 8 (*Validity of a state proposition on a pair of states*)**.** A state proposition $p \in SP_{V_{12}}$ holds on a pair of states (s_1, s_2) of two transition systems $\mathrm{TS}_1 \stackrel{\text{def}}{=} \langle S_{0_1}, S_1, A_1, T_1, \mathcal{L}_1 \rangle$ and $\mathrm{TS}_2 \stackrel{\text{def}}{=} \langle S_{0_2}, S_2, A_2, T_2, \mathcal{L}_2 \rangle$, written $(s_1, s_2) \models p$, if

- $(s_1, s_2) \models ap_1$ iff $ap_1 \in \mathcal{L}_1(s_1)$, where $ap_1 \in AP_{V_1}$.
- $(s_1, s_2) \models ap_2$ iff $ap_2 \in \mathcal{L}_2(s_2)$, where $ap_2 \in AP_{V_2}$.
- $(s_1, s_2) \models p \vee q$ iff $(s_1, s_2) \models p$ or $(s_1, s_2) \models q$.
- $(s_1, s_2) \models \neg p$ iff it is not true that $(s_1, s_2) \models p$, also written $(s_1, s_2) \not\models p$.

We express the link between the states of TS_2 and TS_1 as a *gluing predicate* P_{12}, which is a state proposition of $SP_{V_{12}}$. It says that any state of TS_2 is linked to one and only one state of TS_1.

Definition 9 (*Gluing predicate*)**.** A state proposition $P_{12} \in SP_{V_{12}}$ is a gluing predicate of two transition systems $\mathrm{TS}_1 \stackrel{\text{def}}{=} \langle S_{0_1}, S_1, A_1, T_1, \mathcal{L}_1 \rangle$ over V_1 and $\mathrm{TS}_2 \stackrel{\text{def}}{=} \langle S_{0_2}, S_2, A_2, T_2, \mathcal{L}_2 \rangle$ over V_2 if

$$\forall s_2 \cdot (s_2 \in S_2 \Rightarrow \exists s_1 \cdot (s_1 \in S_1 \wedge (s_1, s_2) \models P_{12} \wedge \forall s_1' \cdot (s_1' \in S_1 \wedge (s_1', s_2) \models P_{12} \Rightarrow s_1' = s_1))).$$

3.2. Refinement relation

Now we define the refinement relation between TS_2 and TS_1 as a τ-simulation which respects the gluing predicate of TS_2 and TS_1. Intuitively, it is defined by the four following clauses:

- *strict refinement (abstract transitions)*: for every refined transition labelled by a former action, there is an abstract transition labelled by the same former action, and such that the source state of the refined transition is related to the source state of the abstract transition, and the target state of the refined transition is related to the target state of the abstract transition;

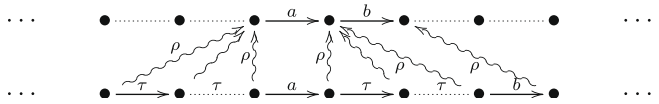

Fig. 1. Execution refinement.

- *stuttering of τ-transitions*: this clause says that the source and target states of a τ-transition must be related to the same abstract state;
- *satisfaction of the gluing predicate*: states which are in relation satisfy the gluing predicate;
- *abstract actions preservation*: for every abstract transition labelled by an action a, there is a refined transition labelled by a such that its source state is related to the source state of the abstract transition.

Definition 10 (*Refinement relation ρ*)**.** Let $TS_1 \stackrel{def}{=} \langle S_{0_1}, S_1, A_1, T_1, \mathcal{L}_1\rangle$ and $TS_2 \stackrel{def}{=} \langle S_{0_2}, S_2, A_2, T_2, \mathcal{L}_2\rangle$ be two transition systems where $A_2 = A_1 \cup \{\tau\}$[3] and such that TS_2 is τ-livelock free.[4] Let P_{12} be the gluing predicate between TS_2 and TS_1. We define the refinement relation ρ as the greatest relation included in $S_2 \times S_1$ that is such that ($s_2 \in S_2$, $s_1 \in S_1$, and we denote as $s_2\ \rho\ s_1$ that s_2 is related to s_1): if $s_2\ \rho\ s_1$ then

- strict refinement: $s_2 \stackrel{a}{\rightarrow} s_2' \wedge a \in A_1 \Rightarrow \exists s_1' \cdot (s_1 \stackrel{a}{\rightarrow} s_1' \wedge s_2'\ \rho\ s_1')$.
- stuttering of τ-transitions: $s_2 \stackrel{\tau}{\rightarrow} s_2' \Rightarrow s_2'\ \rho\ s_1$.
- satisfaction of the gluing predicate: $(s_1, s_2) \models P_{12}$.
- abstract actions preservation: $s_1 \stackrel{a}{\rightarrow} s_1' \wedge a \in A_1 \Rightarrow \exists s_2', s_2'' \cdot (s_2' \in S_2 \wedge s_2'' \in S_2 \wedge s_2' \stackrel{a}{\rightarrow} s_2'' \wedge s_2'\ \rho\ s_1)$.

Remark 11. In [1], one additional clause defines the refinement relation, namely the *lack of new deadlocks*. It means that there must not exist deadlocks in TS_2 which do not exist in TS_1. This additional clause allows to benefit of the preservation of all PLTL properties by the refinement relation [10], while only safety properties are preserved by the relation of Definition 10. Even if we do not consider this clause in the sequel, note that the results presented in the next sections for partitioned model-checking also hold when adding it in the definition of refinement.

Definition 12 (*Refinement*)**.** A transition system $TS_2 \stackrel{def}{=} \langle S_{0_2}, S_2, A_2, T_2, \mathcal{L}_2\rangle$ refines a transition system $TS_1 \stackrel{def}{=} \langle S_{0_1}, S_1, A_1, T_1, \mathcal{L}_1\rangle$, written $TS_2 \sqsubseteq TS_1$, if:

$$\forall s_2 \cdot (s_2 \in S_{0_2} \Rightarrow \exists s_1 \cdot (s_1 \in S_{0_1} \wedge s_2\ \rho\ s_1)).$$

Proposition 13 *To every abstract transition $s_1 \stackrel{a}{\rightarrow} s_1'$ of TS_1 (with $s_1' \neq s_1$) corresponds a fragment of an execution of TS_2 composed of a finite (possibly null) sequence of τ-transitions followed by a transition labelled by a.*

Proof Consider a transition $s_1 \stackrel{a}{\rightarrow} s_1'$ of TS_1. Due to the abstract actions preservation clause, there is a transition $s_2 \stackrel{a}{\rightarrow} s_2'$ of TS_2 such that $s_2\ \rho\ s_1$. Moreover, any fragment of execution made of a sequence of transitions leading to s_2 *via* states all related to s_1 is a sequence of τ-transitions. Indeed, the occurrence of an old action would lead to a state related to another state of TS_1 (due to the strict refinement and the satisfaction of the gluing predicate). Due to the τ-livelock freeness, the succession of τ-transitions is finite, and so action a finally occurs. □

Proposition 13 is illustrated in Fig. 1, where we represent the relation between an abstract (on top) and a refined (at the bottom) execution. The new actions performed by the refined system are seen as τ-actions in the figure.

3.3. Equivalence class

We require ρ to be a total function. This makes it possible to define an equivalence relation $\sim_\rho$ between the states of the refined transition system. This equivalence relation on the states of TS_2 then induces a partitioning of the state space, as presented in the next section. We say that two states s_2 and s_2' of TS_2 are equivalent w.r.t. ρ if they are related to the same state s_1 of TS_1.

Definition 14 (*Equivalence class*)**.** Consider a state $s_1 \in S_1$. The equivalence class $EC(s_1) \subseteq S_2$ of $S_2/\sim_\rho$ is defined as

$$EC(s_1) = \{s_2 \in S_2 \mid s_2\ \rho\ s_1\}.$$

[3] Actually, the refinement relation is defined w.r.t. the τ-transition system of TS_2 on A_1.

[4] This means that the new actions (seen as τ-transitions) cannot take control forever. So there is not infinitely successive τ-transitions in an execution, i.e. there is no τ-cycle in the refined system.

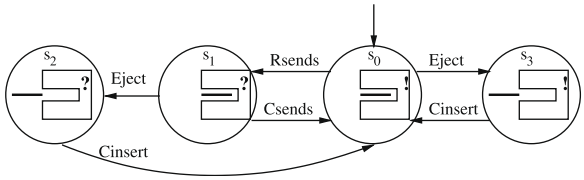

Fig. 2. Transition system of the abstract model of protocol $T = 1$.

3.4. An example: the protocol $T = 1$

We use as running example the protocol $T = 1$ [11]. This protocol defines the exchange of message in an asynchronous half duplex transmission protocol between a device (the *reader*) and a card. We present here a simplified modeling of this protocol on two refinement levels.

3.4.1. Abstract model of protocol $T = 1$

At this level of abstraction, we only consider the message transmission between the card and the reader, by using two variables: $Sender_1$ indicates which component is going to send the next message and $Cstatus_1$ indicates whether the card is inserted into the device or not.

Four actions are described at this level of abstraction. Action *Rsends* corresponds to the sending of a message by the device, whereas action *Csends* corresponds to the sending of a message by the card. The two actions *Cinsert* and *Eject* correspond, respectively, to the insertion and the ejection of the card.

The transition system for this model of the protocol is given in Fig. 2. In each state, the card and the reader are graphically represented, as well as the values of the variables. $Cstatus_1$ takes its values in $\{in, out\}$. In states s_2 and s_3 the card is outside the reader, whereas it is inside in states s_0 and s_1. The signs "!" and "?" are used to represent the value of $Sender_1$, by saying, respectively, that the reader sends ($Sender_1 = reader$) or receives ($Sender_1 = card$).

3.5. Refined model of protocol $T = 1$

At the refined level, each message is considered as a sequence of blocks. For each block sent (*bl*), one receives an acknowledgement of receipt (*ackb*). Each message is ended by a last block (*lb*). The term used to designate these three types of information is *frame*.

Two variables are added at the refined model: $CardF_2$ and $ReaderF_2$, representing the type of the last *frame* (in the domain $\{bl, lb, ackb\}$), respectively, sent by the card and the device. The variable $Cstatus_2$ represents the same thing as $Cstatus_1$, and $SenderF_2$ indicates which component (card or reader) is going to send the next *frame*.

Eight actions are described in this refined model. Actions *Cinsert* and *Eject* still represent the insertion and the ejection of the card. The two actions *Rsends* and *Csends*, also already described in the abstract model, make it possible here to end the sending of a message by sending the last block. The sending of blocks *bl* and of the acknowledgement of receipt *ackb* by the card and the device are treated in actions *Cblocksends*, *Rblocksends*, *Cacksends* and *Racksends*.

The gluing predicate between this refined model and the abstract one is the following:

$(Cstatus_2 = in) \Leftrightarrow (Cstatus_1 = in) \wedge$
$(Cstatus_2 = out) \Leftrightarrow (Cstatus_1 = out) \wedge$
$(ReaderF_2 = bl \vee ((CardF_2 = ackb \vee CardF_2 = lb) \wedge SenderF_2 = reader)) \Leftrightarrow (Sender_1 = reader) \wedge$
$(CardF_2 = bl \vee ((ReaderF_2 = ackb \vee ReaderF_2 = lb) \wedge SenderF_2 = card)) \Leftrightarrow (Sender_1 = card)$.

The transition system for this refined model is given in Fig. 3. In the states, the type of the last *frame* emitted by the card and the reader is represented as indicated by the legend.

Notice that the refined transition system as presented here does not meet the τ-livelock freeness hypothesis. Indeed, new actions *Cblocksends* and *Racksends* can "loop" between the states r_8 and r_9, and new actions *Rblocksends* and *Cacksends* between the states r_3 and r_4. But a fairness assumption stating that the card and the reader cannot infinitely exchange messages can be expressed, for the τ-livelock freeness to hold in this example.

4. Partitioned model-checking

In this section, we present the main results of the out-of-core model-checking technique that we have called verification by partitioned model-checking (see [22,16]). In order to perform model-checking on large transition systems, the partitioned verification technique relies on a simple idea: splitting the transition system into several smaller pieces, and performing the verification on each piece separately. The pieces are called *parts*. Parts are transition systems as well. The initial transition system is called the *global* transition system. Performing a partitioned verification means verifying a property on each part separately, and concluding that it is globally true when it is true on every part.

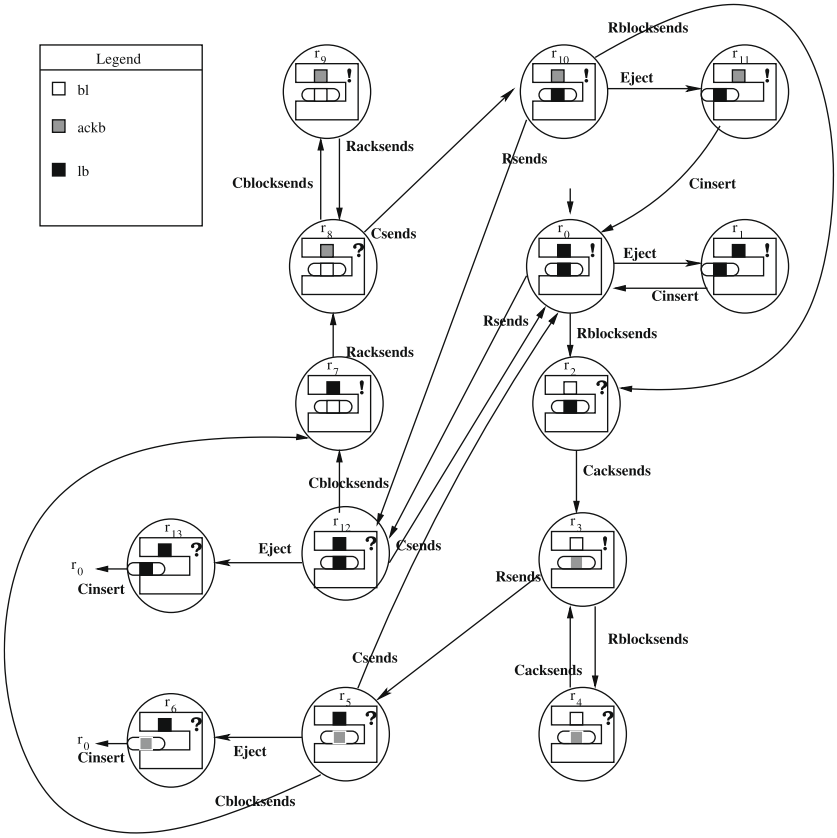

Fig. 3. Transition system of the refined model of the protocol $T = 1$.

In order that every transition belongs to a single part, the parts are constructed by partitioning the transitions of the global transition system. Some states may belong to two distinct parts: they can be the target state of a transition t in one part, and the initial state of a transition t' in another part. Due to the partitioning, some states may lose their successors. If this is the case, remember that a `Skip` loop is added to the state (see Remark 2).

4.1. Properties verifiable by parts

Consider a transition system split into a set of parts (transition systems) according to a partition of its set of transitions.[5] Some PLTL properties are globally true when they are true on every part. We call such properties *verifiable by parts*, and they are defined according to Definition 15.

Definition 15 (*Property verifiable by parts*)**.** Let φ be a PLTL property. Let TS be a transition system, and let $\mathbb{M}$ be a partitioning of TS. The property φ is verifiable by parts on TS based on partitioning $\mathbb{M}$ if

$$\forall M \cdot (M \in \mathbb{M} \Rightarrow M \models \varphi) \Rightarrow \mathrm{TS} \models \varphi.$$

Remark 16. We simply say *φ is verifiable by parts*, instead of *φ is verifiable by parts on TS based on partitioning $\mathbb{M}$*.

Remark 17. To say "if φ is true on every part, then it is true on the global transition system" is equivalent to saying "if φ is false on the global transition system, then it is false on at least one part". So, a definition of a property verifiable by parts, equivalent to that of Definition 15, is

$$\neg(\mathrm{TS} \models \varphi) \Rightarrow \exists M \cdot \exists \sigma \cdot (M \in \mathbb{M} \wedge \sigma \in \Sigma_M \wedge \sigma \models \neg\varphi).$$

[5] Actually, the fact that the parts are obtained by an overlapping of the transitions is sufficient.

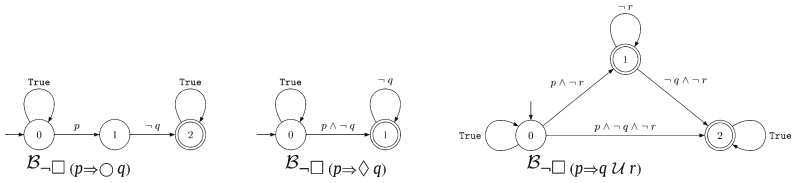

Fig. 4. The properties $\Box\,(p \Rightarrow \bigcirc q)$, $\Box\,(p \Rightarrow \Diamond q)$ and $\Box\,(p \Rightarrow q\,\mathcal{U}\,r)$ are verifiable by parts.

4.2. A class of PLTL properties verifiable by parts

By using Büchi automata, we give a sufficient condition for when a PLTL property is verifiable by parts. We define a class named $\mathcal{C}_{mod}$ (see Definition 18) of Büchi automata, and we prove that every PLTL property whose negation defines an ω-language recognized by an automaton in $\mathcal{C}_{mod}$ is a property verifiable by parts (see Theorem 20).

The Büchi automata in $\mathcal{C}_{mod}$ are defined in Definition 18 as Büchi automata for which the following requirements hold:

(1) The initial state is not an accepting one and there is a loop labelled `True` on it.
(2) Every transition leaving a non-initial state leads to a non-initial state that is an accepting state.
(3) For every transition with a label p leading to an accepting state, there is a transition leaving that state with a label p' such that $p \Rightarrow p'$ holds.

A consequence of requirement 2 is that every transition leaving an accepting state necessarily leads to an accepting state. Another consequence is that once the initial state is left, an accepting state is reached immediately after.

Definition 18 (*The $\mathcal{C}_{mod}$ class*)**.** Let $\mathcal{B} = \langle q_0, Q, \mathrm{SP}_V, T_\mathcal{B}, \mathcal{F}_\mathcal{B}\rangle$ be a Büchi automaton. The automaton $\mathcal{B}$ is in the $\mathcal{C}_{mod}$ class if

$$q_0 \xrightarrow{\texttt{True}} q_0 \in T_\mathcal{B} \wedge q_0 \notin \mathcal{F}_\mathcal{B} \tag{1}$$
$$q \xrightarrow{p} q' \in T_\mathcal{B} \wedge q \neq q_0 \Rightarrow q' \in \mathcal{F}_\mathcal{B} \tag{2}$$
$$q \xrightarrow{p} q' \in T_\mathcal{B} \wedge q' \in \mathcal{F}_\mathcal{B} \Rightarrow \exists (p', q'')\cdot (q' \xrightarrow{p'} q'' \in T_\mathcal{B} \wedge p \Rightarrow p') \tag{3}$$

Proposition 19. *If $\mathcal{B} = \langle q_0, Q, SP_V, T_\mathcal{B}, \mathcal{F}_\mathcal{B}\rangle$ is a Büchi automaton in $\mathcal{C}_{mod}$, then every accepting run $\pi = q_0 \xrightarrow{p_0} q_1 \cdots q_i \xrightarrow{p_i} q_{i+1} \cdots$ of $\mathcal{B}$ is such that*

$$\exists k \cdot (k > 0 \wedge \forall i \cdot ((i < k \Rightarrow q_i = q_0) \wedge (i > k \Rightarrow q_i \in \mathcal{F}_\mathcal{B}))).$$

Proof. Every accepting run is such that its first state is q_0. As $q_0 \notin \mathcal{F}_\mathcal{B}$, then every accepting run necessarily leaves q_0 in order to reach an accepting state. Consider q_k to be the first state of $\mathcal{B}$ to be reached just after the initial state is left for the first time in an accepting run. By construction, every state preceding the occurrence of q_k in the accepting run is the initial state.

As $q_k \neq q_0$, then every target state q'_k of a transition whose source state is q_k is such that $q'_k \in \mathcal{F}_\mathcal{B}$ (by clause 2 in Definition 18) and $q'_k \neq q_0$ (since by clause 1 in Definition 18, $q_0 \notin \mathcal{F}_\mathcal{B}$). Thus, by recurrence, every state q''_k reachable from q_k is such that $q''_k \neq q_0$ and $q''_k \in \mathcal{F}_\mathcal{B}$. □

Notice that clause 3 is not used to prove Proposition 19. But it is necessary for the proof of Theorem 20.

Theorem 20. *All the PLTL properties whose negation defines an ω-language recognized by a Büchi automaton in the $\mathcal{C}_{mod}$ class are verifiable by parts, regardless of the transition system and its partitioning.*

Proof. We refer the reader to [5] for a proof of Theorem 20. □

As examples, the properties $\Box\,(p \Rightarrow \bigcirc q)$, $\Box\,(p \Rightarrow \Diamond q)$ and $\Box\,(p \Rightarrow q\,\mathcal{U}\,r)$ (with p, q and r being state propositions about a transition system) are always verifiable by parts because the automata of the negations of these properties are all in $\mathcal{C}_{mod}$ (see Fig. 4).

4.3. A partitioning based on refinement

In order to perform the verification of PLTL properties by parts, it is necessary to partition the transition system to be verified into a set of parts. We are within the context of refined transition systems, and as a possible partitioning we propose a partitioning of a refined transition system that is based on the refinement relation.

Consider a refined transitions system TS_2 refining an abstract transition system TS_1, and consider a state s_1 of TS_1. A part issued from s_1 encloses, for all the abstract transitions leaving s_1, the fragments of executions of TS_2 defined according to Proposition 13, and followed by a `Skip` loop.

Intuitively, each state of the abstract transition system is exploded in the refined transition system as a set of states, related to each other by transitions labelled by new actions (i.e. the new actions, previously hidden, become visible at the refined level). This corresponds to the sequence of τ-actions represented in Fig. 1.

Then the occurrence of an action of the abstract level (see for example actions a and b in Fig. 1) causes the part to reach a "border" state of it. Each border state of a part is extended by a `Skip` loop according to Remark 2.

This partitioning is illustrated in Fig. 5, where the four states of the abstract transition system of protocol $T = 1$ (see Fig. 2) give birth to four parts. Since in Fig. 2 the state s_0 was possibly left by actions *Rsends* or *Eject*, then the border states of Part 1 in Fig. 5 are reached either by *Rsends* or *Eject*. The same thing goes for the other parts.

Definition 21 (*Refinement based part*)**.** Let $TS_1 \stackrel{\text{def}}{=} \langle S_{0_1}, S_1, A_1, T_1, \mathcal{L}_1\rangle$ and $TS_2 \stackrel{\text{def}}{=} \langle S_{0_2}, S_2, A_2, T_2, \mathcal{L}_2\rangle$ be two transition systems such that TS_2 refines ($\sqsubseteq$) TS_1. Consider $s_1 \in S_1$ and $EC(s_1)$, an equivalence class of $S_2/\sim_\rho$. The part of TS_2 based on $EC(s_1)$ is a transition system $TS_M = \langle S_{0_M}, S_M, A_M, T_M, \mathcal{L}_M\rangle$ defined as

- $S_{0_M} = \{s_2 \in EC(s_1) \mid s_2 \in S_{0_2} \vee \exists (s, a, s_2) \cdot (s \xrightarrow{a} s_2 \in T_2 \wedge s \notin EC(s_1))\}$,
- $S_M = \{s_2 \in EC(s_1)\} \cup \{s' \mid s_2 \xrightarrow{a} s' \in T_2 \wedge s_2 \in EC(s_1) \wedge s' \notin EC(s_1)\}$,
- $T_M = \{s_2 \xrightarrow{a} s' \in T_2 \mid a \in A_2 \wedge s_2 \in EC(s_1)\} \cup \{s' \xrightarrow{\texttt{Skip}} s' \mid s' \in S_M \setminus EC(s_1)\}$,
- A_M is the restriction of A_2 to the labels of T_M,
- $\mathcal{L}_M$ is the restriction of $\mathcal{L}_2$ on the states of S_M.

Proposition 22. *Any execution of a part is made of a succession of occurrences of new actions, ended by the occurrence of an old action, and followed by a* `Skip` *loop.*

Proof. By Definition 21, the states of a part (see the definition of S_M) are the states that are related to the same state s_1 of the abstract transition system (they belong to $EC(s_1)$), plus the states s' that can be reached by one occurrence of an old action. The `Skip` loop is added to s' as the transition relation is total (see Remark 2).

Due to the stuttering of τ-transitions (see Definition 10), the transitions between two states in $EC(s_1)$ are all labelled by a new action. Due to the τ-livelock freeness, an old action necessarily occurs in an execution. This occurrence of an old action necessarily terminates the execution (apart from the `Skip` loop) as it leads to a state not in $EC(s_1)$, from which no transition other than a `Skip` one is allowed (see the definition of T_M in Definition 21). □

Corollary 23. *Any execution of a part of the refined transition system is either a suffix of an execution of the refined system, or is made of a sub-sequence of an execution ending in a state s of the refined system, and extended by an infinite sequence* $s \xrightarrow{\texttt{Skip}} s \xrightarrow{\texttt{Skip}} s \cdots$.

Proof. Immediate. □

Examples of refinement based parts are given in Fig. 5.

In the next section, we extend the partitioned verification approach to a class of PLTL properties (which includes $\mathcal{C}_{mod}$) that are not always verifiable by parts (not with every partitioning). They are verifiable by parts for a given system with regard to this refinement based partitioning.

5. Partitioned verification on refined systems

The sufficient condition of verification by parts presented in the previous section focuses on a limited class of PLTL properties. Indeed, several usual properties do not satisfy it and are not *a priori* verifiable by parts. In particular, this is the case for response properties with fairness assumptions. The previous condition does not consider the way the system was split into parts. But we intend to use the refinement based partitioning in order to split the system. By using the specificity of this kind of partitioning, we are able to extend the class of properties verifiable by parts. In this way, properties which are not *a priori* verifiable by parts could be so in the particular context of a system and its refinement. In this section, we present sufficient conditions to determine if a property is verifiable by parts, regarding to a partitioning based on the refinement process.

5.1. Preliminary definitions

First, let us give some preliminary definitions that are necessary to define the sufficient conditions we propose. In the sequel, we consider a Büchi automaton $\mathcal{B} \stackrel{\text{def}}{=} \langle q_0, Q, SP_V, T_\mathcal{B}, \mathcal{F}_\mathcal{B}\rangle$.

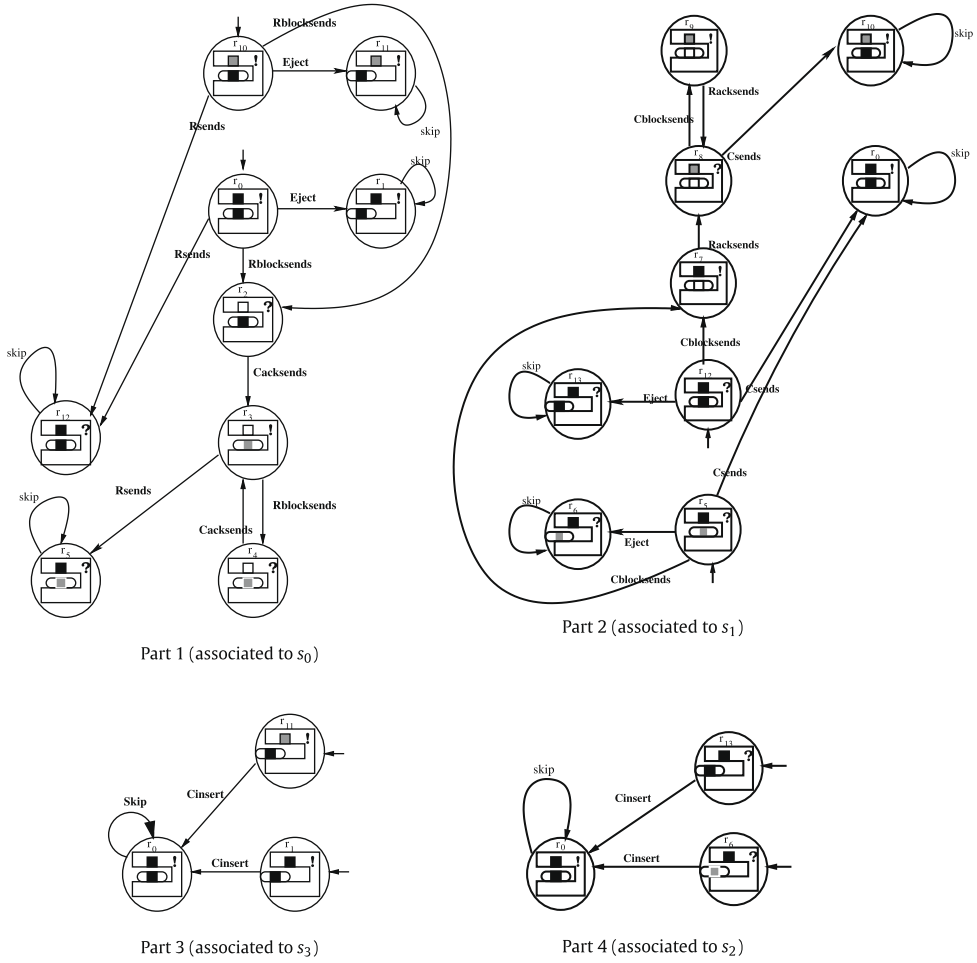

Fig. 5. The four parts obtained by splitting the global refined transition system of protocol $T = 1$.

- The *starting* states of $\mathcal{B}$ are the accepting states and the states reached by one or more transitions from these accepting states. The set $Q_{s_\mathcal{B}}$ of starting states is defined as

$$Q_{s_\mathcal{B}} \stackrel{\text{def}}{=} \{q \mid q \in Post^* (\mathcal{F}_\mathcal{B})\}$$

where $Post^* (\mathcal{F}_\mathcal{B})$ is the set of all the successors of the states in $\mathcal{F}_\mathcal{B}$, reachable with zero or a finite number of transitions.

- The *inhospitable* states are all the non-accepting states which can be reached from a *starting* state. The set $Q_{h_\mathcal{B}}$ of inhospitable states is defined as

$$Q_{h_\mathcal{B}} \stackrel{\text{def}}{=} Q_{s_\mathcal{B}} \setminus \mathcal{F}_\mathcal{B}.$$

- Let $\Delta_{a_\mathcal{B}} \subseteq T_\mathcal{B}$ be the set of transitions such that the target state is an accepting state and the source state is a starting state

$$\Delta_{a_\mathcal{B}} \stackrel{def}{=} \{q \xrightarrow{p} q' \mid q \xrightarrow{p} q' \in T_\mathcal{B} \wedge q \in Q_{s_\mathcal{B}} \wedge q' \in \mathcal{F}_\mathcal{B}\}.$$

- Let $\Delta_{h_\mathcal{B}} \subseteq T_\mathcal{B}$, be the set of transitions such that the target state is an inhospitable state and the source state is a starting state

$$\Delta_{h_\mathcal{B}} \stackrel{def}{=} \{q \xrightarrow{p} q' \mid q \xrightarrow{p} q' \in T_\mathcal{B} \wedge q \in Q_{s_\mathcal{B}} \wedge q' \in Q_{h_\mathcal{B}}\}.$$

- We define $\texttt{Prefix}_{\mathcal{B}}$ as the set of prefixes (with at least two transitions) of runs of $\mathcal{B}$ which start when the initial state is left and which end in the first accepting state encountered.

$$\texttt{Prefix}_{\mathcal{B}} \stackrel{\text{def}}{=} \left\{ q_0 \stackrel{p_0}{\rightarrow} q_1 \cdots q_{n-1} \stackrel{p_{n-1}}{\rightarrow} q_n \;\mid\; q_1 \neq q_0 \wedge q_n \in \mathcal{F}_{\mathcal{B}} \wedge n \geq 2 \wedge \forall i \cdot (1 \leq i < n \Rightarrow q_i \notin \mathcal{F}_{\mathcal{B}}) \right\}.$$

- We define $\texttt{LastTransInPrefix}_{\mathcal{B}}$ as the set of transitions occurring in the runs of $\texttt{Prefix}_{\mathcal{B}}$ such that they leave a non-initial state and lead to an accepting state

$$\texttt{LastTransInPrefix}_{\mathcal{B}} \stackrel{\text{def}}{=} \{ q \stackrel{p}{\rightarrow} q' \;\mid\; q \neq q_0 \wedge q_0 \stackrel{p_0}{\rightarrow} q_1 \cdots q \stackrel{p}{\rightarrow} q' \in \texttt{Prefix}_{\mathcal{B}} \}.$$

5.2. Conditions of verifiability by parts with a refinement-based partitioning

We now present some conditions which, when they hold, ensure that an extended class of PLTL properties (in comparison to the initial method) is verifiable by parts, when considering the particular partitioning based on the refinement process.

The conditions are expressed with respect to two transition systems (such that one refines the other) and to a PLTL property, for which we consider the Büchi automaton of its negation. When both these conditions hold, and the automaton is in a class[6] that we have called $\mathcal{C}$, then the property is verifiable by parts with the refinement based partitioning of the refined transition system, as stated by Theorem 28.

5.2.1. The $\mathcal{C}$ class of Büchi automata

A Büchi automaton belongs to the $\mathcal{C}$ class if the two following criteria hold:

- the initial state is not accepting and there is a loop labelled by $\texttt{True}$ on it (see clause 1),
- for each transition $q \stackrel{p}{\rightarrow} q'$ leading to an accepting state q', there is a cycle reachable from q' that contains an accepting state. Moreover, the label p implies both
  - the label of every transition leading to the cycle,
  - the label of every transition in the cycle (see clause 5).

Definition 24 ($\mathcal{C}$ *class*). A Büchi automaton $\mathcal{B}$ belongs to the $\mathcal{C}$ class if the following two clauses hold:

$$q_0 \stackrel{\texttt{True}}{\rightarrow} q_0 \in T_{\mathcal{B}} \wedge q_0 \notin \mathcal{F}_{\mathcal{B}}, \tag{4}$$

$$q \stackrel{p}{\rightarrow} q' \in T_{\mathcal{B}} \wedge q' \in \mathcal{F}_{\mathcal{B}} \Rightarrow \exists \pi \cdot (\pi = q' \stackrel{p_0}{\rightarrow} q_1 \stackrel{p_1}{\rightarrow} q_2 \cdots \wedge \mathit{Inf}_s(\pi) \cap \mathcal{F}_{\mathcal{B}} \neq \emptyset \wedge \forall i \cdot (i \geq 0 \Rightarrow (p \Rightarrow p_i))). \tag{5}$$

Example 25. To illustrate the definition of the $\mathcal{C}$ class expressed in Definition 24, let us consider the example of the Büchi automaton $\mathcal{B}$ in Fig. 6. This automaton accepts the negation of the PLTL property $\Box(\Box\Diamond p \Rightarrow \Diamond q) \Rightarrow \Box(r \Rightarrow \Diamond s)$, which is a liveness property under a fairness assumption. For this automaton, we have (the labels of the transitions are omitted for readability reasons):

$$\begin{aligned} \texttt{Prefix}_{\mathcal{B}} = \{ & 0 \rightarrow 1 \rightarrow 2, 0 \rightarrow 1 \rightarrow 1 \rightarrow 2, 0 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 2, \cdots, \\ & 0 \rightarrow 3 \rightarrow 2, 0 \rightarrow 3 \rightarrow 3 \rightarrow 2, 0 \rightarrow 3 \rightarrow 3 \rightarrow 3 \rightarrow 2, \cdots, \\ & 0 \rightarrow 3 \rightarrow 4, 0 \rightarrow 3 \rightarrow 3 \rightarrow 4, 0 \rightarrow 3 \rightarrow 3 \rightarrow 3 \rightarrow 4, \cdots \} \end{aligned}$$

$\texttt{LastTransInPrefix}_{\mathcal{B}} = \{1 \rightarrow 2, 3 \rightarrow 2, 3 \rightarrow 4\}$.
$Q_{s_{\mathcal{B}}} = \{2, 3, 4\}$, $Q_{h_{\mathcal{B}}} = \{3\}$.
$\Delta_{a_{\mathcal{B}}} = \{2 \rightarrow 2, 4 \rightarrow 2, 4 \rightarrow 4, 3 \rightarrow 4\}$.
$\Delta_{h_{\mathcal{B}}} = \{3 \rightarrow 3, 4 \rightarrow 3\}$.

This automaton $\mathcal{B}$ belongs to the $\mathcal{C}$ class since the clauses 4, 5 expressed in Definition 24 hold.

- There is a loop labelled by $\texttt{True}$ on the initial state (which is state 0), so the clause 4 holds.
- There are two accepting states in this automaton: states 2 and 4.
  - State 2. Five transitions lead to this state: $0 \stackrel{\neg p \wedge r \wedge \neg s}{\rightarrow} 2$, $1 \stackrel{\neg p \wedge r \wedge \neg s}{\rightarrow} 2$, $2 \stackrel{\neg p \wedge \neg s}{\rightarrow} 2$, $3 \stackrel{\neg p \wedge \neg s}{\rightarrow} 2$ and $4 \stackrel{\neg p \wedge \neg s}{\rightarrow} 2$. By considering cycle $2 \stackrel{\neg p \wedge \neg s}{\rightarrow} 2$, the clause 5 holds.
  - State 4. Two transitions lead to this state: $3 \stackrel{q \wedge \neg s}{\rightarrow} 4$ and $4 \stackrel{q \wedge \neg s}{\rightarrow} 4$. By considering cycle $4 \stackrel{q \wedge \neg s}{\rightarrow} 4$, the clause 5 holds.

[6] The class in itself is not a sufficient condition for the verifiability by parts in general.

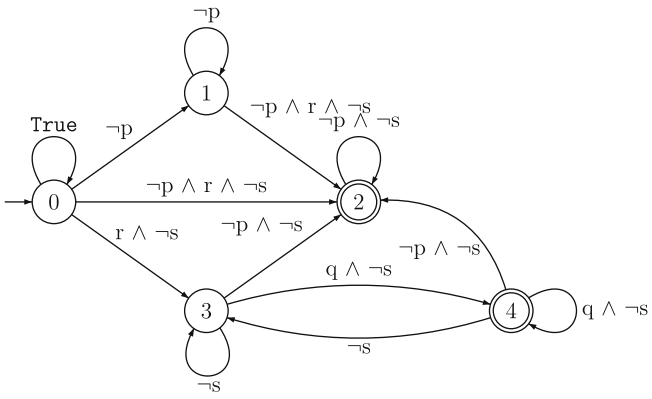

Fig. 6. A Büchi automaton belonging to the $\mathcal{C}$ class.

5.2.2. Sufficient conditions

The intuition to express the conditions of verifiability by parts is based on the definition of a property verifiable by parts, expressing that: if a property is false on the global system, then there is at least one part on which it is false. If a property φ is false on the global system, then there is at least one execution (of the global system) on which it does not hold. That is, this execution is accepted by some run of $\mathcal{B}_{\neg\varphi}$ (the Büchi automaton that recognizes the ω-language of $\neg\varphi$). We focus on the first occurrence of an accepting state in this run. Let us call q_f this first occurrence of the accepting state in the run. Reaching q_f in the run occurs when synchronizing with a particular transition in the execution. The source state s of this transition necessarily belongs to a part. Note that if $q \xrightarrow{p} q_f$ is the transition that reaches q_f in the run, then $s \models p$. Remember that with a refinement-based partitioning, according to Proposition 22, every execution of a part is made of a finite sequence of occurrences of new actions (τ), ended by the occurrence of a former action (see Fig. 1), after which there is an infinite sequence of `Skip`. We indicate two conditions ensuring that there is an execution of a part which is accepted by the Büchi automaton:

(1) The accepting state q_f can be reached from inside a part. That is, by synchronization with a prefix of an execution of the part, leading to the state s whose label allows the synchronization to reach q_f (condition c_1).
(2) It is possible to recognize the violation inside the part, from the state s which allows to reach q_f. That is, it must always be possible to reach an accepting state when exiting the part (condition c_2).

Condition c_1. The condition c_1 concerns the prefixes of the runs (of the Büchi automaton $\mathcal{B}_{\neg\varphi}$) leading to an accepting state, but which do not contain any other accepting states (i.e. the set $\texttt{Prefix}_{\mathcal{B}_{\neg\varphi}}$). It expresses that, for each prefix, the first occurrence of an accepting state is reachable by synchronization with the prefix of an execution of the part. This is true when:

- Either the accepting state, reachable by some transition which do not leave the initial state, can also be directly reached from the initial state.
- Or if the prefix of the run can synchronize with some part of execution going to s, then this part of execution is a sequence of new actions, possibly ended by the occurrence of an old one (thus, the part of execution is contained in one part).

Let us now define formally this condition. In the definition, the notation $\texttt{Factor}_n$, of a set of executions Σ, represents the set of all subsequences of length n (in terms of number of states) of the executions in Σ.

Definition 26 (*Condition* c_1)**.** Let $\text{TS}_1 \stackrel{\text{def}}{=} \langle S_{0_1}, S_1, A_1, T_1, \mathcal{L}_1\rangle$ and $\text{TS}_2 \stackrel{\text{def}}{=} \langle S_{0_2}, S_2, A_2, T_2, \mathcal{L}_2\rangle$ be two transition systems, respectively, interpreted over sets of variables V_1 and V_2, such that $\text{TS}_2 \sqsubseteq \text{TS}_1$. Let $\mathcal{B} \stackrel{\text{def}}{=} \langle q_0, Q, SP_V, T_{\mathcal{B}}, \mathcal{F}_{\mathcal{B}}\rangle$ be a Büchi automaton. The condition c_1 is defined as follows:

$$
\begin{aligned}
&\forall s,q,p,q' \cdot (s \in S_2 \wedge q \xrightarrow{p} q' \in \texttt{LastTransInPrefix}_{\mathcal{B}} \wedge s \models p \Rightarrow \\
&\quad \exists p' \cdot (q_0 \xrightarrow{p'} q' \in T_{\mathcal{B}} \wedge p \Rightarrow p') \ \vee \\
&\quad \forall \sigma \cdot (\sigma = s_1 \xrightarrow{a_1} s_2 \cdots s_{n-1} \xrightarrow{a_{n-1}} s \in \texttt{Factor}_n(\Sigma(TS)) \ \wedge \\
&\quad \exists \rho \cdot (\rho = q_1 \xrightarrow{p_1} q_2 \cdots q_{n-1} \xrightarrow{p_{n-1}} q \xrightarrow{p} q' \in \texttt{Prefix}_{\mathcal{B}} \ \wedge \forall i \cdot (1 \le i \le n-1 \Rightarrow s_i \models p_i)) \Rightarrow \\
&\quad \forall i \cdot (1 \le i < n-1 \Rightarrow a_i \in A_2 \backslash A_1))).
\end{aligned}
$$

Note that in the second part of c_1 (i.e. the second part of the disjunction), the execution σ contains one transition less than the run ρ. This is because, since $s \models p$, any transition from s can synchronize with $q \xrightarrow{p} q'$, and lead to the accepting state. Thus in c_1, we do not focus on one such transition in particular, since we are only interested in being able to reach the accepting state q'.

CONDITION c_2. We now express condition c_2, that guarantees that once an accepting state is reached, the violation is recognized inside the part. We call *exit states*, the states which are target of a transition labelled by an old action (i.e. an action of the abstract transition system). The condition expresses that, from every starting state of the automaton, if the transitions from the exit states cannot synchronize with a transition leading to an accepting state, then they cannot synchronize with a transition going to an inhospitable state.

Definition 27 (*Condition c_2*)**.** Let $\mathrm{TS}_1 \stackrel{\text{def}}{=} \langle S_{0_1}, S_1, A_1, T_1, \mathcal{L}_1\rangle$ and $\mathrm{TS}_2 \stackrel{\text{def}}{=} \langle S_{0_2}, S_2, A_2, T_2, \mathcal{L}_2\rangle$ be two transition systems, respectively, interpreted over sets of variables V_1 and V_2, such that $\mathrm{TS}_2 \sqsubseteq \mathrm{TS}_1$. Let $\mathcal{B} \stackrel{\text{def}}{=} \langle q_0, Q, SP_V, T_{\mathcal{B}}, \mathcal{F}_{\mathcal{B}}\rangle$ be a Büchi automaton. The condition c_2 is defined as follows:

$$\forall s, a, s', q \cdot (s \xrightarrow{a} s' \in T_2 \wedge a \in A_1 \wedge q \in Q_{s_{\mathcal{B}}} \wedge \nexists q', p \cdot (q \xrightarrow{p} q' \in \Delta_{a_{\mathcal{B}}} \wedge s' \models p) \Rightarrow \nexists q'' \cdot (q \xrightarrow{p'} q'', p' \in \Delta_{h_{\mathcal{B}}} \wedge s' \models p')).$$

5.2.3. Verifiability by parts

These conditions lead to the following theorem of verifiability by parts, in the case of a refinement-based partitioning, for properties whose negation can be represented by a Büchi automaton in $\mathcal{C}$.

Theorem 28. *Let* $\mathrm{TS}_1 \stackrel{def}{=} \langle S_{0_1}, S_1, A_1, T_1, \mathcal{L}_1\rangle$ *and* $\mathrm{TS}_2 \stackrel{def}{=} \langle S_{0_2}, S_2, A_2, T_2, \mathcal{L}_2\rangle$ *be two transition systems, respectively, interpreted over sets of variables V_1 and V_2, such that* $\mathrm{TS}_2 \sqsubseteq \mathrm{TS}_1$. *Suppose that* TS_2 *is split into a set of refinement-based parts* $\mathbb{M}$ (*see Def. 21*). *Let $\mathcal{B}$ be a Büchi automaton in the $\mathcal{C}$ class that recognizes the ω-language of a PLTL property $\neg\varphi$.*
If c_1 and c_2 are valid then φ is verifiable by parts on $\mathbb{M}$.

Proof. The complete proof can be found in Appendix A. □

Comparison with our previous works. The following propositions show that the method described in this section to guarantee the verifiability by parts subsumes the previous one (presented in Section 4).

Proposition 29. *The $\mathcal{C}_{mod}$ class is included in the $\mathcal{C}$ class.*

Proof. We want to prove that if a Büchi automaton verifies the clauses 1, 2, 3 of the $\mathcal{C}_{mod}$ class, then it also verifies the clauses 4, 5 of the $\mathcal{C}$ class.

- From the clause 1 of the $\mathcal{C}_{mod}$ class, we know that there is a loop labelled by `True` on the initial state.
- From the clauses 2 and 3 of $\mathcal{C}_{mod}$, we know that each accepting state has a successor which is also an accepting state. As the number of accepting states (in $\mathcal{F}_{\mathcal{B}}$) is finite, it is always possible to reach a cycle containing an accepting state from an accepting state (more precisely, the cycle only contains accepting states and is reached by accepting states). With the clause 3 of $\mathcal{C}_{mod}$, and as the cycle only contains accepting states and is reached by accepting states, we can conclude that the clause 5 holds. □

Proposition 30. *For an automaton in the $\mathcal{C}_{mod}$ class, whatever the transition systems considered, conditions c_1 and c_2 hold.*

Proof

- Condition c_1: from Proposition 19, we know that any run of a Büchi automaton $\mathcal{B}$ in $\mathcal{C}_{mod}$ visits at most one non-accepting state after leaving the initial state and before reaching an accepting state. Thus, for those automata, the prefixes of runs in $\mathtt{Prefix}_{\mathcal{B}}$ are of length two. The second part of the condition (i.e. the second part of the disjunction) forbids the parts of executions which synchronizes with those prefixes of runs to contain transitions labelled by old actions, except for the last transition. Recall that these parts of executions have one transition less than the prefixes of runs. Thus, the parts of executions only contains one transition. These executions thus immediately satisfy the second part of the disjunction, and thus the condition itself.
- Condition c_2: in any run of an automaton in $\mathcal{C}_{mod}$, all the successors of an accepting state are also accepting states (see Proposition 19). Thus, the set of starting states of such an automaton is the set of its accepting states. Moreover, as the inhospitable states are the non-accepting states that can be reached from a starting state, there are no inhospitable states for an automaton in $\mathcal{C}_{mod}$, and so condition c_2 holds. □

5.3. Application to the protocol $T = 1$

We use the refinement-based partitioning to split the refined transition system of the protocol $T = 1$. This partitioning leads to the creation of four parts, as illustrated in Fig. 5 in Section 4. Using our method, we want, in particular, to verify two properties indicating that each message is composed of a finite number of blocks, i.e. each time a device sends a block, then it will eventually send a last block. These two properties are expressed formally with the liveness properties P_1 and P_2:

- $P_1 \stackrel{def}{=} \Box(CardF_2 = bl \Rightarrow \Diamond(CardF_2 = lb))$,
- $P_2 \stackrel{def}{=} \Box(ReaderF_2 = bl \Rightarrow \Diamond(ReaderF_2 = lb))$.

These two properties must be, respectively, verified under the following fairness assumptions f_1 and f_2. They ensure that the card and the reader will not send blocks (bl) forever.

- $f_1 \stackrel{def}{=} \Box(\Box\Diamond(SenderF_2 = card \wedge CardF_2 = bl) \Rightarrow \Diamond CardF_2 \neq bl)$,
- $f_2 \stackrel{def}{=} \Box(\Box\Diamond(SenderF_2 = reader \wedge ReaderF_2 = bl) \Rightarrow \Diamond ReaderF_2 \neq bl)$.

Thus, the two properties considered are $f_1 \Rightarrow P_1$ and $f_2 \Rightarrow P_2$. They can be represented by a Büchi automaton in the $\mathcal{C}$ class that is in the shape of the one in Fig. 6. The experimentation demonstrates that both properties are verifiable by parts and verified by parts. Indeed, the two conditions c_1 and c_2 hold for these properties with the refinement based parts, and the verification on the four parts of the protocol (using the model-checker SPIN [15]) was successful.

Notice that the verification of the conditions c_1 and c_2 is decidable and can be performed by an algorithm that works on automata issued from the Büchi automata of the properties, and of the transition systems of the parts. The condition c_1 is to be verified on the finite sets S_2 (the states of the refined system, which can be obtained by parts) and $\texttt{LastTransInPrefix}_{\mathcal{B}}$, and on the infinite set $\texttt{Prefix}_{\mathcal{B}}$, which is regular. The condition c_2 is to be verified on the finite sets $\Delta_{a_{\mathcal{B}}}$, $\Delta_{h_{\mathcal{B}}}$ and T_2 (the transitions of the refined system, which can be obtained by parts).

6. Conclusion and future work

In this paper, we remind the reader of the main results of an out-of-core model-checking technique that has been presented in [22,16]. This technique is called verification by parts. The transition system of a system is partitioned into a set of parts, and every part is verified independently from the others. We call verifiable by parts the properties that can be verified in this way, and we present an interesting class of PLTL properties that are verifiable by parts for any partitioning. We propose a partitioning of the transition system that is based on the refinement process.

Our contribution in this paper is to present sufficient conditions called c_1 and c_2 according to which a PLTL property is verifiable by parts with regard to this partitioning and a refined system. This allows more PLTL properties to be verified by parts, such as liveness properties expressed with fairness assumptions. In particular, the properties concerned are the ones which negation can be represented by a Büchi automaton belonging to a class called $\mathcal{C}$. The sufficient conditions c_1 and c_2 are expressed as predicates that link the Büchi automaton of the negation of the property to the refined transition system.

We have presented in [5] a different (though similar) approach for the verification of properties under fairness assumptions in a partitioned way. Only the class $\mathcal{C}_{mod}$ is presented in [5], and fairness is handled by adding explicit fairness constraints to the transition systems. This allows properties in the shape of $f \implies \varphi$ (where φ is a property in $\mathcal{C}_{mod}$ and f is a fairness assumption) to become verifiable by parts, even if $f \implies \varphi$ as a whole is not in $\mathcal{C}_{mod}$.

We adopt a different approach in the present paper. Indeed, fairness constraints need not be added to the transition systems we consider, and we can deal for the verification by parts with properties in the shape of $f \implies \varphi$ without the need for φ to be in $\mathcal{C}_{mod}$. Also, the method in the present paper is not restricted to properties under fairness assumptions. It applies to any property in class $\mathcal{C}$ expressed on a system for which conditions c_1 and c_2 hold. Moreover, the partitioning is not the same. In [5], it is necessary that some fair transitions be added to the parts as considered in the present paper, for the method to apply. Thus the parts are usually "bigger" in [5]. However the two methods cannot be compared from their generality, as properties of [5] are verifiable by parts for any partitioning, whereas in the present paper we address more properties, but for which verifiability by parts depends on the refinement based partitioning.

Notice that the results in the present paper still apply if the fairness assumptions are added explicitly to the transition system, as was the case in [5].

Another interesting approach to the model-checking blow up problem when dealing with fairness assumptions is that of [18]. The authors suggest dealing with the fairness assumptions at the algorithmic level instead of adding them to specification. For that, they express fairness requirements as Street automata acceptance conditions, and they propose a symbolic model-checking algorithm that checks for the emptiness of the language defined by Street automata. Thus, the verification of a formula $f \Rightarrow \varphi$ is reduced to the verification of φ. Our partitioned verification approach is compatible with the symbolic model-checking approach of [18]. For this, we need to translate our fair transition systems into Street automata. It is always possible as our fairness assumptions are expressible in PLTL. Once the global system is partitioned, the symbolic model-checking can be applied on each of the parts.

Our approach has similarities with methods based on the assume-guarantee paradigm [14,2,19]. We simply assume that the property holds on all parts but the being currently verified. And if it also holds on the current part, then we guarantee that it holds for the whole system. Our partitioning is not based on a parallel composition of interacting components. It is consecutive to the conception of a system through successive refinements. It can be applied inside a component, in addition to a technique that guarantees that the parallel composition of all the components still satisfies the property.

Also, our approach has to be compared with [25], where a partitioning of the state space is proposed for the verification of a property. Our approach is orthogonal since the partitioning in [25] is guided by the property. Our partitioning is guided by the refinement of a transition system.

Our research team is currently working on an implementation of the partitioned model-checking technique, so that we can evaluate its performance on industrial-size applications. Some of the tools have already been implemented such as a partitioner for a transition system, parsers and interfaces with propositional calculus provers in order to determine if a given Büchi automaton is in $\mathcal{C}_{mod}$, or in $\mathcal{C}$.

Appendix A. Proof of Theorem 28

Proof. Consider a transition system TS_2, which refines a transition system TS_1. A property φ is verifiable by parts on TS_2, split according to the refinement process in a set of parts $\mathbb{M}$, iff when $\neg\varphi$ holds on an execution of TS_2, then a part M in $\mathbb{M}$ contains an execution on which $\neg\varphi$ holds.
Thus, assume that there is an execution $\sigma \in \Sigma(TS_2)$ on which $\neg\varphi$ holds:

$$\sigma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots s_{i-1} \xrightarrow{a_{i-1}} s_i \xrightarrow{a_i} s_{i+1} \cdots$$

Consider now a Büchi automaton $\mathcal{B}_{\neg\varphi}$ which represents the negation of φ. As σ satisfies $\neg\varphi$, there exists a run π in $\mathcal{B}_{\neg\varphi}$ which accepts σ:

$$\pi = q_0 \xrightarrow{p_0} q_1 \xrightarrow{p_1} q_2 \xrightarrow{p_2} \cdots q_{i-1} \xrightarrow{p_{i-1}} q_i \xrightarrow{p_i} q_{i+1} \cdots$$

such that $\forall k \geq 0, s_k \models p_k$ and $Inf_s(\pi) \cap \mathcal{F}_{\mathcal{B}} \neq \emptyset$.

Thus, π synchronizes with σ and contains an infinity of accepting states. We focus on the first occurrence of an accepting state in π. Let q_i be this accepting state. Notice that the first accepting state met in π (i.e. the state q_i) cannot be the initial state, since $\mathcal{B}_{\neg\varphi}$ is in the class $\mathcal{C}$. As $s_{i-1} \models p_{i-1}$, the transition $s_{i-1} \xrightarrow{a_{i-1}} s_i$ synchronizes with the transition $q_{i-1} \xrightarrow{p_{i-1}} q_i$. The state s_{i-1} necessarily belongs to a part M. We want to prove that there exists an execution in this part M, which contains s_{i-1} and on which $\neg\varphi$ holds. Note that s_{i-1} can also belong to two different parts: as an initial state in one part, and as an *exit* state in the other. In this case, we consider the part where s_{i-1} is an exit state. Consider an execution σ' in M

$$\sigma' = s'_0 \xrightarrow{a'_0} s'_1 \xrightarrow{a'_1} \cdots s_{j-1} \xrightarrow{a'_{j-1}} s'_j \xrightarrow{a'_j} s'_{j+1} \cdots$$

where s'_0 is an initial state of M and $s_{j-1} = s_{i-1}$ (but the indices i and j can be different). We prove that there exists a run π' of $\mathcal{B}_{\neg\varphi}$ which accepts σ':

$$\pi' = q_0 \xrightarrow{p'_0} q'_1 \xrightarrow{p'_1} \cdots q'_{j-1} \xrightarrow{p'_{j-1}} q_j \xrightarrow{p'_j} q'_{j+1} \cdots$$

where $q_j = q_i$ (as for σ', the indices i and j can be different). The state q_j still represents the first occurrence of an accepting state in π'. We split the proof in two parts:

- (Part 1) The prefix of π' up to the state q_j synchronizes with the prefix of σ' up to state s'_j. That is, $q_0 \xrightarrow{p'_0} q'_1 \xrightarrow{p'_1} \cdots q'_{j-1} \xrightarrow{p'_{j-1}} q_j$ synchronizes with $s'_0 \xrightarrow{a'_0} s'_1 \xrightarrow{a'_1} \cdots s_{j-1} \xrightarrow{a'_{j-1}} s'_j$.
- (Part 2) The suffix of π' from the state q_j synchronizes with the suffix of σ' from the state s'_j. Moreover, the suffix of π' from q_j contains an infinity of accepting states. That is, $q_j \xrightarrow{p'_j} q'_{j+1} \cdots$ synchronizes with $s'_j \xrightarrow{a'_j} s'_{j+1} \cdots$, and $\forall k > j, \exists l > k$ such that $q'_l \in \mathcal{F}_{\mathcal{B}}$ (recall that q_j is also an accepting state).

Proof of Part 1: the prefix of π' up to the state q_j synchronizes with the prefix of σ' up to the state s'_j

To prove this first clause, let us go back to the transition $q_{i-1} \xrightarrow{p_i} q_i$ of the run π. We know that q_i (which is q_j in π') is an accepting state and that $s_{i-1} \models p_{i-1}$ (s_{i-1} is s_{j-1} in σ'). We handle the two following cases: in π, either (1) q_i is directly reached from the initial state, or (2) it is reached from some other non-initial state.

(1) case $q_{i-1} = q_0$. By the clause 4 of the class $\mathcal{C}$, there is a loop *true* on the initial state of $\mathcal{B}_{\neg\varphi}$. Thus, in this case, the prefix of π' which synchronizes with σ' is immediately the following:

$$\pi' = q_0 \overset{true}{\rightarrow} q_0 \overset{true}{\rightarrow} \cdots q_0 \overset{p_{j-1}}{\rightarrow} q_j$$

where q_{j-1} is the same state as q_{i-1} in π and p_{j-1} is equal to the label p_{i-1} in π.

(2) case $q_{i-1} \neq q_0$. Now, in the run π, we consider that the first accepting state q_i is not directly reached from the initial state in one step. We use the condition c_1 to prove this case. In π, the transition $q_{i-1} \overset{p_{i-1}}{\rightarrow} q_i$ belongs to the set `lastTransInPrefix` since $q_{i-1} \neq q_0$ and $q_i \in \mathcal{F}_{\mathcal{B}}$. The condition c_1 states that

(i) *either there is a transition* $q_0 \overset{p}{\rightarrow} q_i$ *such that* $p_{i-1} \Rightarrow p$. As $s_{i-1} \models p_{i-1}$, then $s_{i-1} \models p$. In σ', we have $s_{j-1} \models p$, and therefore the run π' which accepts σ' is the following:

$$\pi' = q_0 \overset{true}{\rightarrow} q_0 \overset{true}{\rightarrow} \cdots q_0 \overset{p}{\rightarrow} q_j$$

(ii) *or if there exists a part of an execution up to the state* s_{i-1} *which synchronizes with a part of a run leaving the initial state* q_0 *of the Büchi automaton and leading to the transition* $q_{i-1} \overset{p_{i-1}}{\rightarrow} q_i$, *then this part of execution up to the state* s_{i-2} *only contains transitions labelled with new events*. This part of execution exists: it is the part of σ which synchronizes with the part of π from the last occurrence of q_0 to q_i. Call q_k this last occurrence of q_0 in π, we have for π up to the state q_i:

$$\pi = q_0 \overset{p_0}{\rightarrow} q_0 \overset{p_1}{\rightarrow} q_0 \cdots q_k \overset{p_k}{\rightarrow} q_{k+1} \cdots q_{i-1} \overset{p_{i-1}}{\rightarrow} q_i \cdots$$

and the corresponding σ up to state s_i:

$$\sigma = s_0 \overset{a_0}{\rightarrow} s_1 \overset{a_1}{\rightarrow} s_2 \cdots s_k \overset{a_k}{\rightarrow} s_{k+1} \cdots s_{i-1} \overset{a_{i-1}}{\rightarrow} s_i \cdots$$

By the condition c_1, we know that each a_l, $k \leq l < a_{i-2}$, is a new action (i.e. labelled in $A_2 \backslash A_1$). Thus, the part of σ from s_k to s_i is entirely contained in one part. Indeed, recall Proposition 22 saying that, with a refinement-based partitioning, the executions of a part are composed of a finite sequence of new actions, ended by an old one (labelled in A_1), and a `Skip` loop. Thus, a_k to a_{i-3} are new actions, a_{i-2} can be either a new action or an old one, and a_{i-1} can be either a new action, or an old one, or a `Skip` loop. Thus, the execution σ' is the part of σ from s_k to s_i, preceded by a possible sequence of consecutive transitions that goes to s_k in σ and that are labelled by new actions. The transitions can synchronize with the loop *true* on the initial state of the Büchi automaton.

Proof of Part 2: the suffix of π' from the state q_j synchronizes with the suffix σ' from the state s'_j, and this suffix of π' contains an infinity of accepting states

In the first part of the proof, we proved that the state s_{j-1} in σ' satisfies the label p'_{j-1} of some transition in π' leading to the accepting state q_j (whatever this transition is, according to the first part of the proof). Thus, as $s_{j-1} \models p'_{j-1}$, then any transition from s_{j-1} can synchronize with the transition $q'_{j-1} \overset{p'_{j-1}}{\rightarrow} q_j$. In particular, this is the case for the transition $s_{i-1} \overset{a_{i-1}}{\rightarrow} s_i$ in the global execution σ. Moreover, recall that the suffix of σ from the state s_i can synchronize with the suffix of the run π from the state q_i, and that this suffix of π contains an infinity of accepting states. Thus, consider that σ' has the same suffix from s_{j-1} as σ from the state s_{i-1}. Call σ'_{j-1} the suffix of σ' from s_{j-1}:

$$\sigma'_{j-1} = s_{j-1} \overset{a_{j-1}}{\rightarrow} s_j \overset{a_j}{\rightarrow} s_{j+1} \overset{a_{j+1}}{\rightarrow} \cdots$$

where $s_{j-1} = s_{i-1}, a_{j-1} = a_{i-1}, s_j = s_i, \cdots$. We consider the two following cases:

(1) either σ'_{j-1} is entirely contained in the part M which contains s_{j-1},
(2) or it is cut on some exit state s_x of the part. That is, only the prefix from s_{j-1} to s_x is contained in M, and is then extended with `Skip` transitions:

$$\sigma'_{j-1} = s_{j-1} \overset{a_{j-1}}{\rightarrow} s_j \overset{a_j}{\rightarrow} s_{j+1} \overset{a_{j+1}}{\rightarrow} \cdots s_{x-1} \overset{a_{x-1}}{\rightarrow} s_x \overset{\texttt{Skip}}{\rightarrow} s'_{x+1} \overset{\texttt{Skip}}{\rightarrow} \cdots$$

where $x \geq j$ and $\forall k > x, s'_k = s_x$.

Note that we go back to the states of indices $j-1$. Indeed, in the first part of the proof, we only proved that $s_{j-1} \models p'_{j-1}$, for some transition labelled by p'_{j-1} leading to the accepting state q_j. But we did not specify which transition from s_{j-1} synchronizes with the transition $q'_{j-1} \overset{p'_{j-1}}{\rightarrow} q_j$.

(1) The suffix σ'_{j-1} is entirely contained in M. In this case, the proof is immediate. The suffix σ'_{j-1} synchronizes with the following suffix π'_{j-1}:

$$\pi'_{j-1} = q'_{j-1} \xrightarrow{p'_{j-1}} q_j \xrightarrow{p_j} q_{j+1} \xrightarrow{p_{j+1}} \cdots$$

such that the suffix from q_j is the same as the suffix of π from its state q_i. As this suffix π_i contains an infinity of accepting states, then π'_j also contains an infinity of accepting states.

(2) The suffix σ'_{j-1} is cut on some exit state s_x of M, and is extended with `Skip` transitions from s_x. We consider two cases: either $s_x = s_{j-1}$ or $s_x \neq s_{j-1}$:

(i) if $s_x = s_{j-1}$: in this case, the only transition available from s_x is a `Skip` transition. Thus, we have

$$\sigma'_{j-1} = s_{j-1} \xrightarrow{\texttt{Skip}} s'_j \xrightarrow{\texttt{Skip}} s'_{j+1} \cdots.$$

such that $\forall k \geq j, s'_k = s_{j-1}$.
By the clause 5 of the class $\mathcal{C}$, we know that there exists a suffix π'_{j-1} of π' from the state q'_{j-1}:

$$\pi'_j = q'_{j-1} \xrightarrow{p'_{j-1}} q_j \xrightarrow{p'_j} q'_{j+1} \xrightarrow{p'_{j+1}} q'_{j+2} \cdots$$

such that $p'_{j-1} \Rightarrow p'_k$, $\forall k \geq j$ and that $Inf_s(\pi'_j) \cap \mathcal{F}_\mathcal{B} \neq \emptyset$. As $s_{j-1} \models p'_{j-1}$ (whatever p'_{j-1} is, according to the first part of the proof), then $\forall k \geq j, s_{j-1} \models p'_k$ and thus $s'_k \models p'_k$.

(ii) if $s_x \neq s_{j-1}$: by construction of the parts, $s_{x-1} \xrightarrow{a_{x-1}} s_x$ is labelled by an old action in A_1. Recall that σ'_{j-1} has the following form:

$$\sigma'_{j-1} = s_{j-1} \xrightarrow{a_{j-1}} s_j \xrightarrow{a_j} s_{j+1} \xrightarrow{a_{j+1}} \cdots s_{x-1} \xrightarrow{a_{x-1}} s_x \xrightarrow{\texttt{Skip}} s'_{x+1} \xrightarrow{\texttt{Skip}} \cdots$$

We can prove that there exists a suffix π'_{j-1}

$$\pi'_{j-1} = q'_{j-1} \xrightarrow{p'_{j-1}} q_j \xrightarrow{p_j} q_{j+1} \xrightarrow{p_{j+1}} \cdots q_{x-1} \xrightarrow{p_{x-1}} q_x \xrightarrow{p'_x} q'_{x+1} \xrightarrow{p'_{x+1}} \cdots$$

which synchronizes with σ'_{j-1} and contains an infinity of accepting states. For this purpose, we use the assumption that condition c_2 holds. First recall that there exists a transition $q_x \xrightarrow{p_x} q_{x+1}$ in the Büchi automaton such that $s_x \models p_x$. Thus, each $s'_k \models p_x$, for $k > x$. As $x \geq j$ and q_j is an accepting state, each state q_k, $k \geq j$, is a starting state and each state q_k, $k > j$, is either an inhospitable or an accepting state. Thus, $q_x \in Q_{s_\mathcal{B}}$ and $q_{x+1} \in Q_{h_\mathcal{B}} \cup \mathcal{F}_\mathcal{B}$:

- If $q_{x+1} \in \mathcal{F}_\mathcal{B}$: according to the clause 5 of the class $\mathcal{C}$, there is a run from q_{x+1} which contains an infinity of accepting states, such that p_k implies the labels of each transition in this run. As each s'_k satisfies p_k, for $k > x$, then each s'_k satisfies the labels of the transitions of this run. The suffix σ'_{j-1} synchronizes with the following accepting suffix π'_{j-1}:

  $$\pi'_{j-1} = q'_{j-1} \xrightarrow{p'_{j-1}} q_j \xrightarrow{p_j} q_{j+1} \xrightarrow{p_{j+1}} \cdots q_{x-1} \xrightarrow{p_{x-1}} q_x \xrightarrow{p_x} q_{x+1} \xrightarrow{p'_{x+1}} \cdots$$

  such that $p_x \Rightarrow p'_k$, for $k > x$.

- If $q_{x+1} \in Q_{h_\mathcal{B}}$: we know that the transition $s_{x-1} \xrightarrow{a_{x-1}} s_x$ is labelled by an old action in A_1, and that $s_x \models p_x$. By condition c_2, we know that there exists a transition $q_x \xrightarrow{p'_x} q'_{x+1}$ such that $s_x \models p'_x$ and $q'_{x+1} \in \mathcal{F}_\mathcal{B}$. At this point, the previous case applies. The suffix σ'_{j-1} synchronizes with an accepting suffix π'_{j-1}:

  $$\pi'_{j-1} = q'_{j-1} \xrightarrow{p'_{j-1}} q_j \xrightarrow{p_j} q_{j+1} \xrightarrow{p_{j+1}} \cdots q_{x-1} \xrightarrow{p_{x-1}} q_x \xrightarrow{p'_x} q'_{x+1} \xrightarrow{p'_{x+1}} \cdots$$

  such that $p'_x \Rightarrow p'_k$, for $k > x$. □

References

[1] F. Bellegarde, J. Julliand, O. Kouchnarenko, Ready-simulation is not ready to express a modular refinement relation, FASE'2000, LNCS, vol. 1783, Springer-Verlag, 2000, pp. 266–283.
[2] S. Berezin, S. Campos, E.M. Clarke, Compositional reasoning in model checking, COMPOS'97 – Revised Lectures, LNCS, vol. 1536, Springer-Verlag, London, UK, 1998, pp. 81–102.
[3] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, Y. Zhu, Symbolic model checking using SAT procedures instead of BDDs, DAC'99, Design Automation Conference, ACM Press, 1999, pp. 317–320.
[4] J.R. Burch, E.M. Clarke, K.L. McMillan, L.J. Dill, D.L. Hwang, Symbolic model checking: 10^{20} states and beyond, 5th IEEE Symposium on Logic in Computer Science, IEEE, 1990, pp. 428–439.
[5] S. Chouali, J. Julliand, P.-A. Masson, F. Bellegarde, PLTL partitioned model-checking for reactive systems under fairness assumptions, ACM Transactions on Embedded Computing Systems (TECS) 4 (2) (2005) 267–301.
[6] E.M. Clarke, E.A. Emerson, P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specification, in: ACM Transactions on Programming Languages and Systems, vol. 2, 1986, pp. 244–263.
[7] E.M. Clarke, O. Grumberg, D.E. Long, Model checking and abstraction, ACM Transactions on Programming Languages and Systems (TOPLAS) 16 (5) (1994) 1512–1542.
[8] E.M. Clarke Jr., O. Grumberg, D.A. Peled, Model Checking, MIT Press, 1999..
[9] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, 4th ACM Symposium on Principles of Programming Languages (POPL'77), ACM Press, Los Angeles, California, USA, 1977, pp. 238–252.
[10] C. Darlot, J. Julliand, O. Kouchnarenko, Refinement preserves PLTL properties, in: Third International Conference of B and Z Users, ZB'03, LNCS, vol. 2651, Turku, Finland, September 2003, pp. 408–420.
[11] Comité Européen de Normalisation, EN 27816-3 european standard – identification cards, integrated circuit(s) card(s) with contacts, electronic signal and transmission protocols, September 1992.
[12] J. Dingel, T. Filkorn, Model checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving, Computer Aided Verification, CAV'95, Lecture Notes in Computer Science, vol. 939, Springer-Verlag, Liège, Belgique, 1995, pp. 54–69.
[13] R. Gorrieri, A. Rensink, Action refinement, in: J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, Elsevier, 2001, pp. 1047–1147. (Chapter 16).
[14] O. Grumberg, D.E. Long, Model checking and modular verification, TOPLAS, ACM Transactions on Programming Languages and Systems 16 (3) (1994) 843–871.
[15] G.J. Holzmann, The model checker SPIN, IEEE Transactions on Software Engineering, Special Issue on Formal Methods in Software Practice 23 (5) (1997) 279–295.
[16] J. Julliand, P.-A. Masson, H. Mountassir, Vérification par model-checking modulaire des propriétés dynamiques introduites en B, Technique et Science Informatiques 20 (7) (2001) 927–957.
[17] S. Katz, D.A. Peled, An efficient verification method for parallel and distributed programs, Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS, vol. 354, Springer-Verlag, 1988, pp. 489–507.
[18] Y. Kesten, A. Pnueli, L. Raviv, Algorithmic verification of linear temporal logic specifications, Proceedings of the 25th International Colloquium on Automata, Languages, and Programming (ICALP 1998), LNCS, vol. 1443, Springer-Verlag, 1998, pp. 1–16.
[19] O. Kupferman, M.Y. Vardi, Modular model checking, COMPOS'97 – Revised Lectures, LNCS, vol. 1536, Springer-Verlag, 1998, pp. 381–401.
[20] O. Lichtenstein, A. Pnueli, Checking that finite state concurrent programs satisfy their linear specification, in: Proceedings of the 12th ACM Symposium on Principles of Programming Languages (POPL'85), New Orleans, USA, 1985, pp. 97–107.
[21] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems: Specification, Springer-Verlag, 1992, ISBN 0-387-97664-7.
[22] P.-A. Masson, H. Mountassir, J. Julliand, Modular verification for a class of PLTL properties, Proceedings of the 2nd International Conference on Integrated Formal Methods (IFM 2000), LNCS, vol. 1945, Springer-Verlag, Dagstuhl Castle, Germany, 2000, pp. 398–419.
[23] Ken McMillan, Symbolic Model-Checking, Kluwer Academic Publishers, 1993..
[24] J.-P. Queille, J. Sifakis, Specification and verification of concurrent systems in CESAR, Proceedings of the International Symposium on Programming, LNCS, vol. 137, Springer-Verlag, Turin, Italy, 1982, pp. 337–351.
[25] R. Sebastiani, E. Singerman, S. Tonetta, M.Y. Vardi, GSTE is partitioned model-checking, Technical Report DIT-04-032, University of Trento, Department of Information and Communication Technology, May 2004.
[26] Silvan Toledo, A survey of out-of-core algorithms in numerical linear algebra, External Memory Algorithms and Visualization, 1999.
[27] M.Y. Vardi, P. Wolper, An automata theoretic approach to automatic program verification, in: Proceedings of the First IEEE Symposium on Logic in Computer Science (LICS'86), Cambridge, USA, June 1986, pp. 332–344.
[28] M.Y. Vardi, P. Wolper, Reasoning about infinite computations, Information and Computation 115 (1) (1994) 1–37.
[29] P. Wolper, P. Godefroid, Partial-order methods for temporal verification, Proceedings of the International Conference on Concurrency Theory (CONCUR'93), LNCS, vol. 715, Springer-Verlag, Hildesheim, 1993, pp. 233–246.