A Compared Study of Two Correctness Proofs for the Standardized Algorithm of ABR Conformance [*]

Béatrice Bérard[1], Laurent Fribourg[1], Francis Klay[2] and Jean-François Monin[2]

[1] LSV, CNRS UMR 8643, ENS de Cachan, 61 av. du Prés. Wilson,
94235 Cachan cedex, France
{berard,fribourg}@lsv.ens-cachan.fr
[2] France Telecom CNET DTL/MSV, Technopole Anticipa
2 av. Pierre Marzin, 22307 Lannion, France
{Francis.Klay,JeanFrancois.Monin}@cnet.francetelecom.fr

Abstract. The ABR conformance protocol is a real-time program that controls dataflow rates on ATM networks. A crucial part of this protocol is the dynamical computation of the expected rate of data cells. We present here a modelling of the corresponding program with its environment, using the notion of (parametric) timed automata. A fundamental property of the service provided by the protocol to the user is expressed in this framework and proved by two different methods. The first proof relies on inductive invariants, and was originally verified using theorem-proving assistant Coq. The second proof is based on reachability analysis, and was obtained using model-checker HyTech. We explain and compare these two proofs in the unified framework of timed automata.

1 Introduction

Over the last few years, an extensive amount of research has been devoted to the formal verification of real-time concurrent systems. Basically, formal proof methods belong to two different fields: theorem proving and model-checking. For all these methods, a first crucial phase is to build a formal description of the system under study. With theorem proving, the description consists of a set of formulas, and the verification is done using logical inference rules. With model-checking, the description is a graph and the verification is performed using systematic search in the graph. The theorem-proving methods apply to more general problems, but often need human interaction while model-checking methods are more mechanical, but apply to a restricted class of systems. These methods thus appear as complementary ones, and several authors advocated for the need to combine them together [17, 21, 24]. This is now an exciting and ambitious trend of research, but still a very challenging issue. We believe that a preliminary useful step towards this objective is to evaluate comparatively the respective merits

[*] Partially supported by Action FORMA (French Programme DSP-STTC/CNRS/MENRT) and RNRT Project *Calife*

and shortcomings of such methods, not only at a general abstract level, but on difficult practical examples. From such a comparison, one may hopefully draw some general lessons for combining methods in the most appropriate way. Besides the comparison may be interesting *per se*, as it may contribute to a better understanding of the specific treated problem. We illustrate here the latter point by performing a compared analysis of two correctness proofs obtained separately for a sophisticated real-life protocol.

More precisely, we propose a comparison between two proofs of the *Available Bit Rate* (ABR) conformance algorithm, a protocol designed at France Telecom [15] in the context of network communications with *Asynchronous Transfer Mode* (ATM). The first proof [19, 18] was obtained in the theorem proving framework using Floyd-Hoare method of inductive invariants and the proof assistant Coq [7]. The second proof [8] was based on the method of reachability analysis, and used the model-checking tool HyTech [14]. In order to compare these methods more easily, we formulate them in the unified framework of "p-automata" [8], a variant of parametric timed automata [5]. The choice of such a framework (motivation, differences with other models, specific problems) is discussed at the end of the paper (section 7).

Context and motivation of the case study. ATM is a flexible packet-switching network architecture, where several communications can be multiplexed over a same physical link, thus providing better performances than traditional circuit-switching networks. Several types of ATM connexions, called *ATM Transfer Capabilities* (ATC), are possible at the same time, according to the dataflow rate asked (and paid) for by the service user. Each mode may be seen as a generic contract between the user and the network. On one side, the network must guarantee the negociated *quality of service* (QoS), defined by a number of characteristics like maximum cell loss or transfer delay. On the other side, data packets (cells) sent by a user must conform to the negociated traffic parameters. Among other ATCs, *Deterministic Bit Rate* connexions operate with a constant rate, while *Statistical Bit Rate* connexions may use a high rate, but only for "short" time intervals.

In some of the most recently defined ATCs, like *Available Bit Rate* (ABR), the allowed cell rate (Acr) may vary during the same session, depending on the current congestion state of the network. Such ATCs are designed for irregular sources, that need high cell rates from time to time, but may reduce their cell rate when the network is busy. A servo-mechanism is then proposed in order to let the user know whether he can send data or not. This mechanism has to be well defined, in order to have a clear traffic contract between user and network.

The conformance of cells sent by the user is checked using an algorithm called GCRA (generic control of cell rate algorithm). In this way, the network is protected against user misbehaviors and keeps enough resources for delivering the required QoS to well behaved users. In fact, a new ATC cannot be accepted (as an international standard) without an efficient conformance control algorithm, and some evidence that this algorithm has the intended behavior.

In this paper, we study the particular case of ABR, for which a simple "ideal" algorithm of conformance control can be given. This algorithm is very inefficient, in terms of memory space, and only approximation algorithms can be implemented in practice. The correctness proof for these approximation algorithms consists in showing that their outputs are never smaller than the outputs computed by the ideal algorithm. More precisely, we focus on an algorithm, called $\mathcal{B}'$, due to Christophe Rabadan at France-Telecom, now part of the I.371.1 standard [15]. We describe this algorithm and prove its correctness with respect to the ideal algorithm, using the two methods mentioned above.

The plan of the paper is as follows: section 2 gives an informal description of the problem of ABR conformance control and section 3 presents an incremental algorithm used henceforth as a specification. Section 4 describes a general modelling framework, called "p-automata", and expresses the two different proof methods in this context; the description of ABR algorithms as p-automata is given in section 5. Verification with the two proof methods is done in section 6. A discussion on constraints linked to the modelling with p-automata follows in section 7, then a comparison between the two methods is given in section 8. We conclude in section 9.

2 Overview of ABR

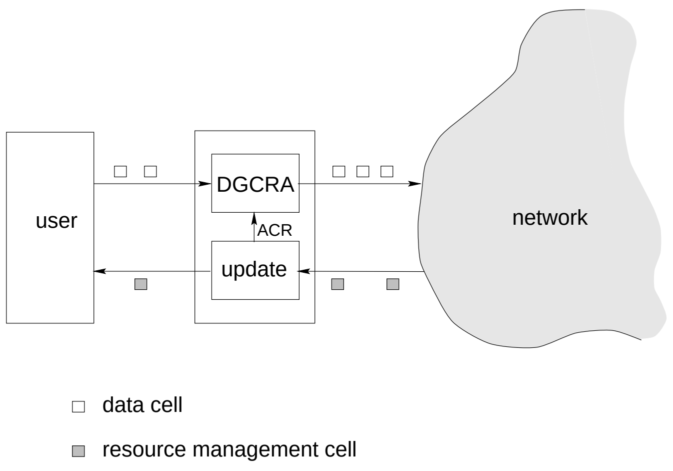


Fig. 1. conformance control

An abstract view of the ABR protocol is given in Figure 1. The conformance control algorithm for ABR has two parts. The first one is quite simple and is not addressed here. It consists of an algorithm called DGCRA (dynamic GCRA), which is an adaptation of the public algorithm for checking conformance of cells: it just checks that the rate of data cells emitted by the user is not higher than a value which is approximately Acr, the allowed cell rate. Excess cells may be discarded by DGCRA. In the case of ABR, the rate Acr depends on time: its current value has to be known each time a new data cell comes from the user. Thus, the complexity lies in the second part: the computation of $\mathrm{Acr}(t)$ ("update" in Figure 1), where t represents an arbitrary time in the future, at which some data cell may arrive. The value of $\mathrm{Acr}(t)$ depends on successive values carried by resource management (RM) cells transmitted from the network to the user [1]. Each such value corresponds to some rate Acr, that should be reached as soon as possible.

2.1 Definition of ideal rate $\mathrm{Acr}(t)$

We consider the sequence of values $(\mathrm{R}_0, \dots, \mathrm{R}_n)$ carried by RM cells, ordered by their arrival time ($r_i \leq r_{i+1}$ for any i). By a slight abuse of notation, the cell carrying R_i will be called itself R_i. The value $\mathrm{Acr}(t)$ depends only on cells R_i whose arrival time r_i occur before t. Intuitively, $\mathrm{Acr}(t)$ should be the last value R_i received at time t, i.e. R_n with $n = \mathrm{last}(t)$, where $\mathrm{last}(u) = max\{i \mid r_i \leq u\}$. Unfortunately, because of electric propagation time and various transmission mechanisms, the user is aware of this expected value only after a delay. Taking the user's reaction time τ observed by the control device into consideration, that is, the overall round trip time between the control device and the user, $\mathrm{Acr}(t)$ should then be R_n with $n = \mathrm{last}(t-\tau)$. But τ may vary in turn. ITU-T considers that a lower bound τ_3 and an upper bound τ_2 for τ are established during the negociation phase of each ABR connection. Hence, a cell arriving from the user at time t on DGCRA may legitimately have been emitted using any rate R_i such that i is between $\mathrm{last}(t-\tau_2)$ and $\mathrm{last}(t-\tau_3)$. Any rate less than or equal to any of these values, or, equivalently less than or equal to the maximum of them, should then be allowed. Therefore, $\mathrm{Acr}(t)$ is taken as the maximum of these R_i. Formally, if $m \leq n$ and $r_m, r_{m+1}, \dots, r_n$ are the successive arrival times of RM cells, such that :

$$r_m \leq t - \tau_2 < r_{m+1} \leq r_{m+2} \leq \cdots \leq r_n \leq t - \tau_3,$$

(in other words, $m = \mathrm{last}(t-\tau_2)$ and $n = \mathrm{last}(t-\tau_3)$; note that m and n depend on t) and if $\mathrm{R}_m, \mathrm{R}_{m+1}, \dots, \mathrm{R}_n$ are the corresponding rate values, then the expected rate is

$$\mathrm{Acr}(t) = max\{\mathrm{R}_i, m \leq i \leq n\}.$$

The case where $n = m$ is obtained when no new RM cell has arrived between $t - \tau_2$ and $t - \tau_3$.

A program of conformance control based on this specification would need to compute, at each instant s, the maximum of the rate values of all the RM

[1] Actually, RM cells are sent by the user, but only their transmission from the network to the user is relevant here; details are available in [22]

cells received during interval $]s - \tau_2, s]$, which may be several hundreds on an ATM network with large bandwidth. ITU-T committee considered that an exact computation of $\mathrm{Acr}(t)$ along these lines is not feasible at reasonable cost with current technologies.

2.2 Algorithm $\mathcal{B}'$: efficient computation of an approximation of $\mathrm{Acr}(t)$

A more realistic algorithm, called $\mathcal{B}'$, due to C. Rabadan, has been proposed by France-Telecom and adopted in I.371.1. It requires the storage of no more than two RM cell rates at a time, and dynamically computes an approximation value `A` of $\mathrm{Acr}(t)$. Two auxiliary variables, `Efi` and `Ela`, are used in the program for storing these two RM cell rates and two dates, `tfi` and `tla`, are associated with `Efi` and `Ela` respectively. When the current time s reaches `tfi`, `A` is updated with value `Efi`, and only one RM cell rate is kept in memory (`Efi:=Ela, tfi:=tla`). When a new RM cell arrives ($s = r_k$), auxiliary variables `Efi`, `Ela`, `tfi` and `tla` are updated according to several cases, depending on the position of r_k w.r.t. `tfi` and `tla`, and R_k w.r.t. `Efi` and `Ela`. The full description of $\mathcal{B}'$ is given in section 5.3 (cf. pseudocode in appendix A).

Correctness of $\mathcal{B}'$. Before being accepted as an international standard (norm ITU I-371.1), algorithm $\mathcal{B}'$ had to be proved correct with respect to $\mathrm{Acr}(t)$: it was necessary to ensure that the flow control of data cells by comparison with `A` rather than $\mathrm{Acr}(t)$ is never disadvantageous to the user. This means that when some data cell arrives at time t, `A` is an *upper* approximation of $\mathrm{Acr}(t)$.

3 Incremental computation of the rate $\mathrm{Acr}(t)$

As explained above, the correctness of algorithm $\mathcal{B}'$ mainly relies on a comparison between the output value `A` of $\mathcal{B}'$ and the ideal rate $\mathrm{Acr}(t)$. The initial specification of $\mathrm{Acr}(t)$ given in section 2.1, turns out to be inadequate for automated (and even manual) manipulation. Therefore, it is convenient to express the computation of $\mathrm{Acr}(t)$ under an algorithmic form as close as possible to $\mathcal{B}'$, where, in particular, updates are performed upon reception of RM cells. Monin and Klay were the first to formulate such an incremental computation using a higher-order functional point of view [19]. We recall their algorithm, then adopt a slightly different view, which is more suited to the modelling framework described subsequently.

3.1 A higher-order functional view: algorithm $\mathcal{F}$

We now consider an algorithm, called $\mathcal{F}$, that stores, at current time s, an estimation `E` of Acr, and updates it at each arrival of an RM cell. More precisely, when receiving a new cell R_k at time r_k, algorithm $\mathcal{F}$ computes the new function `E`$'$ from the former function `E`, depending on the situation of r_k with respect to the argument t of `E`:

	$t < r_k + \tau_3$	$r_k + \tau_3 \leq t < r_k + \tau_2$	$r_k + \tau_2 \leq t$
$\mathtt{E}'(t)$	$\mathtt{E}(t)$	$max(\mathtt{E}(t), \mathrm{R}_k)$	R_k

.

It can be shown that the value $\mathtt{E}(t)$ at current time s is equal to the ideal rate $\mathrm{Acr}(t)$, as defined in section 2.1, for each t such that $t \leq s + \tau_3$. A formal proof of this statement can be found in [19], although a much shorter one can be deduced from the justification in section 3.2 below.

Algorithm $\mathcal{F}$ can thus be seen as a *higher order* functional program which computes the (first-order) functions $\mathtt{E}$. It might be implemented using the general notion of "closures". However, the functions $\mathtt{E}$ are constant over time intervals $[a, b[$, where a and b are of the form $r_i + \tau_j$ and such that there is no value of this form between a and b. Hence, they can be encoded using well chosen finite lists of pairs (t, e), where t is a time and e is the rate $\mathtt{E}(t)$. Such lists can be seen as schedulers for the expected rate: when current time s becomes equal to t, the expected rate becomes e. The length of these schedulers may be several hundreds on networks with large bandwidth (entailing a high frequency of events R_k), even if only the relevant pairs (t, e) are kept. In contrast, algorithm $\mathcal{B}'$ can be considered as using a scheduler of length at most two (containing pairs (tfi,Efi), (tla,Ela)).

3.2 A parametric view: algorithm $\mathcal{I}_{\mathtt{t}}$

Another way of looking at $\mathcal{F}$ is to consider t as a *parameter* (written $\mathtt{t}$) whose value is fixed but unknown, and represents a target observation time. Function $\mathcal{F}$ becomes an "ideal algorithm" $\mathcal{I}_{\mathtt{t}}$, which updates a value $\mathtt{E}(t)$, henceforth written $\mathtt{E}_{\mathtt{t}}$, upon reception of RM cells as follows:

	$r_k \leq \mathtt{t} - \tau_2$	$\mathtt{t} - \tau_2 < r_k \leq \mathtt{t} - \tau_3$	$\mathtt{t} - \tau_3 < r_k$
$\mathtt{E}'_{\mathtt{t}}$	R_k	$max(\mathtt{E}_{\mathtt{t}}, \mathrm{R}_k)$	$\mathtt{E}_{\mathtt{t}}$

,

It is now almost immediate that the value $\mathtt{E}_{\mathtt{t}}$ computed by $\mathcal{I}_{\mathtt{t}}$, is equal to the ideal rate $\mathrm{Acr}(\mathtt{t})$, as defined in section 2.1, when $s = \mathtt{t}$. Indeed, as s increases, k takes the values

$$0, \ldots, m, m+1, \ldots, n, n+1, \ldots,$$

with $m = \mathrm{last}(\mathtt{t} - \tau_2)$ and $n = \mathrm{last}(\mathtt{t} - \tau_3)$, and $\mathtt{E}_{\mathtt{t}}$ takes accordingly the values

$$\mathrm{R}_0, \ldots, \mathrm{R}_m, max(\mathrm{R}_m, \mathrm{R}_{m+1}), \ldots, max(\mathrm{R}_m, \ldots, \mathrm{R}_n), max(\mathrm{R}_m, \ldots, \mathrm{R}_n), \ldots$$

In particular, when $s = \mathtt{t}$, we have $\mathtt{E}_{\mathtt{t}} = max(\mathrm{R}_m, \ldots \mathrm{R}_n) = \mathrm{Acr}(\mathtt{t})$.

The correctness property of $\mathcal{B}'$ with respect to $\mathrm{Acr}(\mathtt{t})$ can now be reformulated as follows, where $\mathtt{A}$ is the output value of $\mathcal{B}'$ and $\mathtt{E}_{\mathtt{t}}$ the value computed by algorithm $I_{\mathtt{t}}$:

$$\text{when } s = \mathtt{t}, \quad \mathtt{A} \geq \mathtt{E}_{\mathtt{t}}$$

This property is referred to as $U_{\mathtt{t}}$ and should be proved for all values of parameter $\mathtt{t}$. Henceforth, the parameter $\mathtt{t}$ is left implicit and we simply write U instead of $U_{\mathtt{t}}$. Accordingly, we write $\mathtt{E}$ instead of $\mathtt{E}_{\mathtt{t}}$ and $\mathcal{I}$ instead of $\mathcal{I}_{\mathtt{t}}$. From this point on, algorithm $\mathcal{I}$ plays the rôle of *specification*.

4 Modelling Framework and Proof Methods

It should be clear from the informal definition of the control conformance algorithm given above, that the ability to reason about real time is essential. The expressions $\mathtt{A}$ and $\mathtt{E}$, involved in property U, denote quantities that evolve as time goes, and should be considered as functions of the current time s. In order to express and prove property U, we need a formal framework. The model of p-automata which was chosen in this paper, turns out to be sufficient for our purposes of formal description and verification of the considered system. We describe this model hereafter as well as two proof methods for verifying properties in this context.

4.1 p-automata

The model of parametric timed automata, called p-automata for short, is an extension of timed automata [4] with parameters. A minor difference with the classical parametric model of Alur-Henzinger-Vardi [5] is that we have only one clock variable $\mathtt{s}$ and several "discrete" variables $w_1, ..., w_n$ while, in [5], there are several clocks and no discrete variable. One can retrieve (a close variant of) Alur-Henzinger-Vardi parametric timed automata by changing our discrete variable w_i into $\mathtt{s}-w_i$ (see [12]). Alternatively, p-automata can be viewed as particular cases of *linear hybrid automata* [2, 3, 20], and our presentation is inspired from [14]. The main elements of a p-automaton are a finite set L of *locations*, transitions between these locations and a family of real-valued variables. In the figures, as usual, locations are represented as circles and transitions as labeled arrows.

Variables and constraints. The variables of a p-automaton are: a tuple $\overline{p}$ of *parameters*, a tuple $\overline{w}$ of *discrete variables* and a *universal clock* $\mathtt{s}$. These real-valued variables differ only in the way they evolve when time increases. Parameter values are fixed by an initial constraint and never evolve later on. Discrete variable values do not evolve either, but they may be changed through instantaneous updates. A universal clock is a variable whose value increases uniformly with time.

A *(parametric) term* is an expression of the form $w+\Sigma_i \alpha_i p_i+\beta$, $\mathtt{s}+\Sigma_i \alpha_i p_i+\beta$ or $\Sigma_i \alpha_i p_i + \beta$, where w is a discrete variable, p_i is a parameter and α_i, β are constants in $\mathbb{Z}$. With the usual convention, an empty set of indices corresponds to a term without parameter. A *convex constraint* is a conjunction of (strict or large) inequalities between terms. A *p-constraint* is a disjunction of convex constraints. An *update relation* is a conjunction of inequalities between a variable and a term. It is written $w' \odot$ *term*, where w' is a primed copy of a discrete variable w, $\odot\ \in \{<, \leq, =, \geq, >\}$ and *term* is a parametric term. As usual, $x' = x$ is implicit if x' does not appear in the update relation.

Locations and transitions. With each location $\ell \in L$ is associated a convex constraint called *location invariant*. Intuitively, the automaton control may reside in location ℓ only while its invariant value is *true*, so invariants are used to enforce progress of the executions. When omitted, the default invariant of a location is the constant *true*.

A transition in a p-automaton is of the form: $\langle \ell, \varphi, a, \theta, \ell' \rangle$, where a is the label of the transition, ℓ the origin location, ℓ' the target location, φ a *guard* and θ an *update relation*. The guard is a convex constraint. Guards may additionnaly contain the special expression *asap*. In our model, a location ℓ is called *urgent* if all transitions with origin ℓ contain *asap* in the guard (no time is allowed to pass in such a location). Otherwise, it is called *stable*. A sequence of transitions is called *complete* if it is of the form $\langle \ell, \varphi_1, a_1, \theta_1, \ell_1 \rangle \langle \ell_1, \varphi_2, a_2, \theta_2, \ell_2 \rangle \ldots$ $\langle \ell_n, \varphi_{n+1}, a_{n+1}, \theta_{n+1}, \ell' \rangle$, where ℓ and ℓ' are stable locations and all intermediate locations ℓ_i are urgent locations. With the usual convention, the case $n = 0$ corresponds to a single transition between stable locations.

Executions. The executions of a p-automaton are described in terms of a transition system. A (symbolic) state q is defined by a formula $(\lambda = \ell)\ \wedge\ \psi(\mathtt{s}, \overline{p}, \overline{w})$, where λ is a variable ranging over the set of locations, and ψ is a p-constraint. We are primarily interested in states for which ψ implies the location invariant I_ℓ. Such states are called *admissible states*. A *p-zone*, represented by a formula $\Pi(\lambda, \mathtt{s}, \overline{p}, \overline{w})$, is a finite disjunction of states. Alternatively it can be regarded as a finite set of states. The *initial state* is $q_{init} : (\lambda = \ell_{init}) \wedge \psi_{init}(\mathtt{s}, \overline{p}, \overline{w})$, for some p-constraint ψ_{init} and initial location ℓ_{init}. Initial location is assumed to be stable. Since parameter values are fixed from the initial state, we often omit the tuple $\overline{p}$ of parameters from the formulas. For a p-automaton, two kinds of moves called *action moves* and *delay moves* are possible from an admissible state $q :\ (\lambda = \ell)\ \wedge\ \psi$.

- An *action move* uses a transition of the form $\langle \ell, \varphi, a, \theta, \ell' \rangle$, reaching an admissible state $q' :\ (\lambda = \ell') \wedge \psi'$, provided ψ implies guard φ and $\psi'(\mathtt{s}, \overline{w}')$ is equivalent to $\exists \overline{w}\ \ \psi(\mathtt{s}, \overline{w}) \wedge \theta(\overline{w}, \overline{w}')$. Informally, discrete variables are modified according to update relation θ and the automaton switches to target location ℓ'. This notion of action move generalizes in a natural way to the notion of action move through a complete sequence of transitions. Note that action moves are instantaneous: the value s of the clock does not evolve.
- A *delay move* corresponds to spending time in a location ℓ. This is possible if ℓ is stable and if the invariant I_ℓ remains true. The resulting admissible state is $q' :\ (\lambda = \ell) \wedge \psi(\mathtt{s'}, \overline{w})$ with $\mathtt{s'} \geq \mathtt{s}$ (nothing else is changed during this time).

A *successor* of a state q is a state obtained either by a delay or an action move. For a subset Q of states, $Post^*(Q)$ is the set of iterated successors of the states in Q. It can be easily proved that the class of p-zones is *closed* under the *Post* operation. As a consequence, $Post^*(Q)$ is a p-zone if Q is a p-zone and computation of $Post^*$ terminates. The notion of *predecessor* is defined in a similar way, using operator *Pre*.

Synchronization. From two or more p-automata representing components of a system, it is possible to build a new p-automaton by a synchronized product. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two p-automata with a common universal clock $\mathtt{s}$. The *synchronized product* (or parallel composition, see e.g. [14]) $\mathcal{A}_1 \times \mathcal{A}_2$ is a p-automaton with $\mathtt{s}$ as universal clock and the union of sets of parameters (resp. discrete variables) of $\mathcal{A}_1$ and $\mathcal{A}_2$ as set of parameters (resp. discrete variables). Locations of the product are pairs (ℓ_1, ℓ_2) of locations from $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. A pair (ℓ_1, ℓ_2) is stable if ℓ_1 *and* ℓ_2 are stable. The p-constraints associated with locations (invariants, initial p-constraint) are obtained by the conjunction of the components p-constraints. The automata move independently, except when transitions from $\mathcal{A}_1$ and $\mathcal{A}_2$ have a common synchronization label. In this case, both automata perform a synchronous action move, the associated guard (resp. update relation) being the conjunction of both guards (resp. update relations).

4.2 Proof methods

We now present two proof methods for proving a property $\Pi(\lambda, \mathtt{s}, \overline{w})$ in the framework of p-automata. This property is assumed to involve only stable locations, and to hold for *all* parameter valuations satisfying the initial p-constraint of the modeled system. The first method, based on Floyd-Hoare method of assertions, consists in proving that Π is an inductive invariant of the model (see, e.g., [27]). The second one, based on model-checking techniques, consists in characterizing the set of all the reachable states of the system, and checking that no element violates Π.

Inductive invariants. To prove Π by inductive invariance, one has to prove that Π holds initially, and is preserved through any move of the system: either an action or a delay move. Formally, we have to prove:

- $\psi_{init}(\mathtt{s}, \overline{w}) \wedge I_{\ell_{init}}(\mathtt{s}, \overline{w}) \Rightarrow \Pi(\ell_{init}, \mathtt{s}, \overline{w})$
- For any transition $\langle \ell, \varphi, a, \theta, \ell' \rangle$ between two stable locations ℓ and ℓ':
  $\varphi(\mathtt{s}, \overline{w}) \wedge \theta(\overline{w}, \overline{w}') \wedge I_\ell(\mathtt{s}, \overline{w}) \wedge I_{\ell'}(\mathtt{s}, \overline{w}') \wedge \Pi(\ell, \mathtt{s}, \overline{w}) \Rightarrow \Pi(\ell', \mathtt{s}, \overline{w}')$.
  A similar formula must also be proved for complete sequences of transitions.
- For any stable location ℓ:
  $I_\ell(\mathtt{s}, \overline{w}) \wedge I_\ell(\mathtt{s}', \overline{w}) \wedge \mathtt{s} \leq \mathtt{s}' \wedge \Pi(\ell, \mathtt{s}, \overline{w}) \Rightarrow \Pi(\ell, \mathtt{s}', \overline{w})$.

Reachability analysis. Since $Post^*(q_{init})$ represents the set of reachable states of a p-automaton, property Π holds for the system if and only if $Post^*(q_{init})$ is contained in the set Q_Π of states satisfying Π. Equivalently, one can prove the emptiness for the zone $Post^*(q_{init}) \cap Q_{\neg\Pi}$, where $Q_{\neg\Pi}$ is the set of states violating Π. Also note that the same property can be expressed using Pre^* by $q_{init} \cap Pre^*(Q_{\neg\Pi}) = \emptyset$.

5 Description of the system

Algorithms $\mathcal{I}$ and $\mathcal{B}'$ will be naturally represented by p-automata. However, they are reactive programs: they react when some external events occur (viz., upon reception of an RM cell) or when current time s reaches some value (e.g., `tfi`). Thus, in order to formally prove correctness property U, we need to model as a third p-automaton, an appropriate environment viewed as an event generator. Finally, in the full system obtained as a synchronized product of the three automata, we explain how to check the correctness property. All these p-automata share a universal clock `s`, the value of which is the current time s. Without loss of understanding (context will make it clear), we often use `s` instead of s.

5.1 A model of environment and observation

As mentioned above, the p-automaton $\mathcal{A}_{env}$ modeling environment (see Figure 2) generates external events such as receptions of RM cells. It also generates a "snapshot" action taking place at time `t`. Note that for our purpose of verification of U, it is enough to consider the snapshot as a final action of the system. The variables involved are the parameter `t` (snapshot time) and a discrete variable `R` representing the rate value carried by the last received RM cell. In the initial location *Wait*, a loop with label *newRM* simulates the reception of a new RM cell: the rate `R` is updated to a non deterministic positive value (written `R' > 0`, as in HyTech [14]). The *snapshot* action has `s=t` as a guard, and location *Wait* is assigned invariant $\mathtt{s} \leq \mathtt{t}$ in order to "force" the switch to location *EndE*.

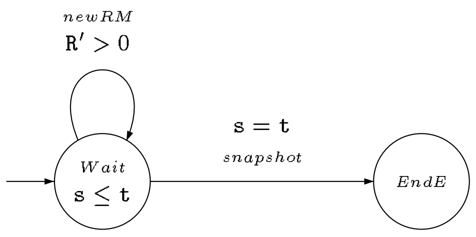


Fig. 2. Automaton $\mathcal{A}_{env}$ modeling arrivals of RM cells and snapshot

5.2 Algorithm $\mathcal{I}$

Algorithm $\mathcal{I}$ computes `E` in an incremental way as shown in the table of section 3.2. Variable `E` is updated at each reception of an RM cell, until current

time s becomes equal to `t`. More precisely, algorithm $\mathcal{I}$ involves variable `R` and parameter `t` (in common with $\mathcal{A}_{env}$) and, in addition:
- the two parameters τ_3 and τ_2 (representing the lower and upper bounds of the transit time from the interface to the user and back),
- the "output" variable `E` (which equates with the ideal rate Acr(t) when $s = t$).
Initially, `E` and `R` are equal. Algorithm $\mathcal{I}$ reacts to each arrival of a new RM cell with rate value `R` by updating `E`. There are three cases, according to the position of its arrival time `s` with respect to `t`-τ_2 and `t`-τ_3 (see section 3.2):

1. If `s` $\leq$ `t`-τ_2, `E` is updated to the new value `R`:
   `[I1] if t >= s+`τ_2 `then E'= R`
2. If `t`-τ_2 $<$ `s` $\leq$ `t`-τ_3, the new rate becomes `E'=`*max*`(E,R)`. To avoid using function *max*, this computation is split into two subcases:
   `[I2a] if s+`τ_3 `<= t < s+`τ_2 `and E < R then E'= R`
   `[I2b] if s+`τ_3 `<= t < s+`τ_2 `and E >= R then E'= E`
3. If `s` $>$ `t`-τ_3, the rate `E` is left unchanged:
   `[I3] if t < s+`τ_3 `then E'= E`

Algorithm $\mathcal{I}$ terminates when the snapshot takes place (`s=t`). In the following, we will sometimes write the updated output value `E'` under the "functional" form $\mathcal{I}$(`s`, `R`, `E`).

Automaton $\mathcal{A}_{\mathcal{I}}$. Algorithm $\mathcal{I}$ is naturally modeled as p-automaton $\mathcal{A}_{\mathcal{I}}$ (see Figure 3). Initial location is *Idle*, with initial p-constraint `E = R`. The reception of an RM cell is modeled as a transition *newRM* from location *Idle* to location *UpdE*. This transition is followed by an urgent (*asap*) transition from *UpdE* back to *Idle*, which updates `E` depending on the position of `s` w.r.t. `t`-τ_2 and `t`-τ_3, as explained above. Without loss of understanding, transitions from *UpdE* to *Idle* are labeled `[I1], [I2a], [I2b], [I3]` as the corresponding operations. Observation of the value `E` corresponds to the transition *snapshot* from *Idle* to final location *EndI*.

5.3 Algorithm $\mathcal{B}'$: computation of an approximation

We now give a full description of algorithm $\mathcal{B}'$ (cf. pseudo-code in appendix). Like $\mathcal{I}$, algorithm $\mathcal{B}'$ involves parameters τ_3 and τ_2 and variable `R`. However, note that `t` is not a parameter for $\mathcal{B}'$. It computes `A` (intended to be an approximation of `E`) using five specific auxiliary variables:
- `tfi` and `tla`, which play the role of `fi`-rst and `la`-st deadline respectively,
- `Efi`, which is the value taken by `A` when current time `s` reaches `tfi`,
- `Ela`, which stores the rate value R carried by the last received RM cell.
- `Emx`, a convenient additional variable, representing the maximum of `Efi`, `Ela`.

Initially, `s=tfi=tla`, and the other variables are all equal. Algorithm $\mathcal{B}'$ reacts to two types of events: "receiving an RM cell" (which is an event in common with $\mathcal{I}$), and "reaching `tfi`" (which is an event specific to $\mathcal{B}'$).

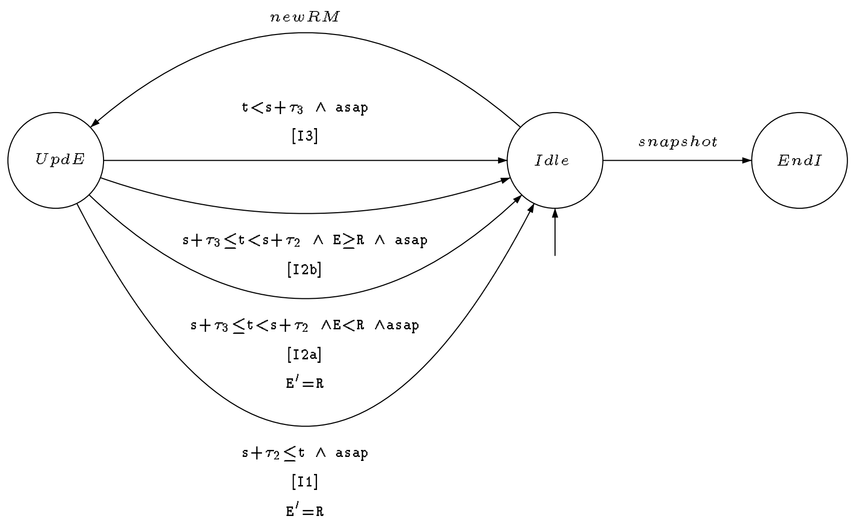


Fig. 3. Automaton $\mathcal{A}_{\mathcal{I}}$

Receiving an RM cell. When, at current time `s`, a new RM cell with value `R` arrives, the variables are updated according to the relative positions of `s+`τ_3 and `s+`τ_2 with respect to `tfi` and `tla`, and those of `R` with respect to `Emx` and `A`. There are eight cases from `[1]` to `[8]` (with two subcases for `[1]`):

```
[1a] if s < tfi and Emx <= R and tfi < s+$\tau_3$ and s+$\tau_3$ < tla then
       Emx' = R; Ela' = R; tla' = s+$\tau_3$
[1b] if s < tfi and Emx <= R and tfi < s+$\tau_3$ and tfi= tla then
       Emx' = R; Ela' = R; tla' = s+$\tau_3$
[2] if s < tfi and Emx <= R and tfi < tla and tla <= s+$\tau_3$ then
       Emx' = R; Ela' = R
[3] if s < tfi and Emx <= R and tfi >= s+$\tau_3$ and A <= R then
       Emx' = R; Ela' = R; Efi' = R; tfi' = s+$\tau_3$; tla' = s+$\tau_3$
[4] if s < tfi and Emx <= R and tfi >= s+$\tau_3$ and A > R then
       Emx' = R; Ela' = R; Efi' = R; tla' = tfi
[5] if s < tfi and Emx > R and R < Ela then
       Ela' = R; Efi' = Emx; tla' = s+$\tau_2$
[6] if s < tfi and Emx > R and R >= Ela then
       Ela' = R, Efi' = Emx
[7] if s >= tfi and A <= R then
       Ela' = R, Efi' = R, Emx'= R, tfi'= s+$\tau_3$, tla'= s+$\tau_3$
[8] if s >= tfi and A > R then
       Ela' = R, Efi' = R, Emx' = R, tfi' = s+$\tau_2$, tla' = s+$\tau_2$
```

Reaching `tfi`. When the current time `s` becomes equal to `tfi`, the approximate current rate `A` is updated to `Efi` while `Efi` is updated to `Ela` and `tfi`

is updated to `tla` (operation `[9]`):
`[9] A' = Efi, tfi' = tla, Efi' = Ela, Emx' = Ela`

When the events "reaching `tfi`" (`s=tfi`) and "receiving an RM cell" simultaneously occur, operation `[9]` must be performed before operation `[1], ..., [8]` (accounting for the RM cell reception).

Like $\mathcal{I}$, algorithm $\mathcal{B}'$ terminates at snapshot time (`s=t`). If the snapshot occurs simultaneously with reaching `tfi`, operation `[9]` must be performed before termination of $\mathcal{B}'$.

Note that the ordering of `s`, `tfi` and `tla` just after operation `[9]` depends on the respective positions of `tfi` and `tla` at the moment of performing `[9]`. In case (`s=`)`tfi=tla`, one still has `s=tfi=tla` just after performing `[9]`, then `s` becomes greater than `tfi=tla` as time increases (until an RM cell occurs or `s=t`). In case (`s=`)`tfi<tla` when performing `[9]`, one has `s<tfi=tla` immediately after.

Automaton $\mathcal{A}_{\mathcal{B}'}$. In order to implement the higher priority of operation `[9]` over the other operations in case of simultaneous events, it is convenient to distinguish the case where `s` is greater than `tfi` from the case where `s`$\leq$`tfi`. To that goal, we introduce two locations *Greater* and *Less*. Operation `[9]` always occurs at location *Less*, but the target location depends on whether `tfi=tla` (subcase `[9a]`) or `tfi<tla` (subcase `[9b]`).

The p-automaton $\mathcal{A}_{\mathcal{B}'}$ is represented in Figure 4 with only the most significant guards and no update information. Like before, the same labels are used for automaton transitions and corresponding program operations.

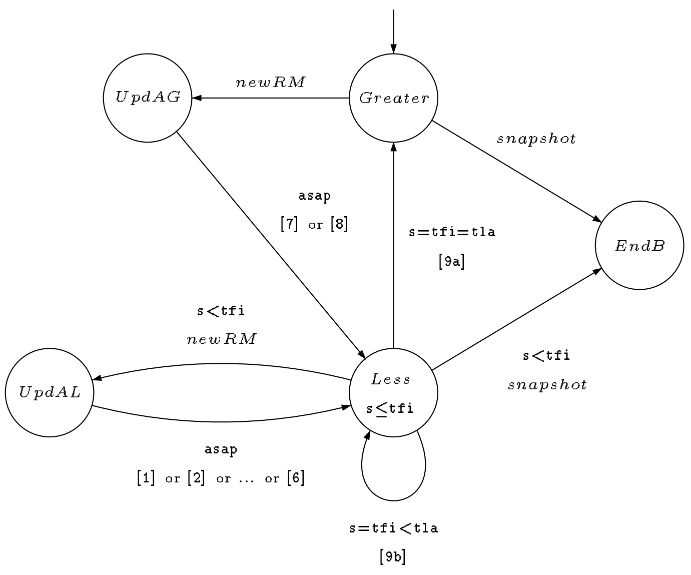


Fig. 4. Approximation automaton $\mathcal{A}_{\mathcal{B}'}$

Initially $\mathcal{A}_{\mathcal{B}'}$ is in *Greater*, with p-constraint: `s=tfi=tla` $\wedge$ `A=Efi=Ela=Emx=R`. Location *Less* has `s`$\leq$`tfi` as an invariant, in order to force execution of transition `[9b]` (if `tfi<tla`) or `[9a]` (if `tfi=tla`) when `s` reaches `tfi`. From *Less*, transition `[9b]` goes back to *Less* (since, after update, `s<tfi=tla`) while transition `[9a]` switches to *Greater* (since `s`$\geq$`tfi=tla` as time increases). The reception of an RM cell corresponds to a transition *newRM*. There are two cases depending on whether the source location is *Less* or *Greater*. From *Less* (resp. *Greater*), transition *newRM* goes to location *UpdAL* (resp. *UpdAG*). This transition is followed by an urgent transition from *UpdAL* (resp. *UpdAG*) back to *Less*, which updates the discrete variables according to operations `[1],...,[6]` (resp. `[7],[8]`), as explained above. Note that transition *newRM* from *Less* to *UpdAL* has an additional guard `s<tfi` in order to prevent an execution of *newRM* before `[9a]` or `[9b]` when `s=tfi` (which is forbidden when "reaching `tfi`" and *newRM* occur simultaneously).

Like before, observation is modeled as a transition *snapshot* from location *Less* or *Greater* to *EndB*. Also note that transition *snapshot* from *Less* to *EndB* has guard `s<tfi` in order to prevent its execution before `[9a]` or `[9b]` when `s=tfi` (which is forbidden when "reaching `tfi`" and the snapshot occur simultaneously).

5.4 Synchronized product and property *U*

The full system is obtained by the product automaton $\mathcal{T} = \mathcal{A}_{env} \times \mathcal{A}_{\mathcal{I}} \times \mathcal{A}_{\mathcal{B}'}$ of the three p-automata above, synchronized by the labels *newRM* and *snapshot*. The action moves occur when the current time reaches `tfi` or `t` or upon reception of an RM cell (*newRM*). In this last case, return to a stable location is obtained by a complete sequence of transitions: *newRM* followed by transitions `[I1],[I2a],[I2b],[I3]` in $\mathcal{A}_{\mathcal{I}}$ and `[1],...,[6]` or `[7],[8]` in $\mathcal{A}_{\mathcal{B}'}$.

Recall that property U expresses in terms of the ideal rate `E` computed by $\mathcal{I}$, and the approximate value `A` computed by $\mathcal{B}'$, as: when $\mathtt{s} = \mathtt{t}$, $\mathtt{A} \geq \mathtt{E}$. In our model $\mathcal{T}$, snapshot action occurs as soon as `s=t`, and makes the automaton switch to its final location $\ell_\infty = (EndE, EndI, EndB)$. Henceforth we write ℓ_+ and ℓ_- respectively for locations $(Wait, Idle, Greater)$ and $(Wait, Idle, Less)$. Actually, the property $\mathtt{A} \geq \mathtt{E}$ does not hold in all locations of $\mathcal{T}$ when `s=t`. This is due to the necessary completion of all the actions in case of simultaneous events. Thus, at location ℓ_-, when `s=t=tfi`, one may have $\mathtt{A} < \mathtt{E}$ just before treatment of `[9]`. However in location $\ell_\infty = (EndE, EndI, EndB)$, all the appropriate actions are completed. Property U states therefore as follows:

$((\lambda = \ell_\infty) \wedge (\mathtt{s} = \mathtt{t})) \Rightarrow \mathtt{A} \geq \mathtt{E}$.

Since location ℓ_∞ is reached when `s=t`, and no action then occurs, an equivalent statement of U is:

$\lambda = \ell_\infty \Rightarrow \mathtt{A} \geq \mathtt{E}$.

6 Verification of correctness

6.1 Verification with inductive invariants

In order to prove $U : \lambda = \ell_\infty \Rightarrow \mathtt{A} \geq \mathtt{E}$, we are going to prove that $\mathrm{Inv} \equiv U \wedge \bigwedge_{i=1}^{10} \mathrm{Aux}_i$ is an inductive invariant of the system, where Aux_i are auxiliary properties of the system. Some of these auxiliary properties (viz., $\mathrm{Aux}_3, \mathrm{Aux}_4, \mathrm{Aux}_5$) involve an additional variable $\mathtt{r}$, which represents the reception date of the last RM cell. (Such variables, that record some history of system execution without affecting it, are called "history" variables [1, 27].) In our model, this can be easily implemented by introducing a discrete variable $\mathtt{r}$ in the environment automaton, and updating it with current time value $\mathtt{s}$, whenever event *newRM* occurs. Initially: $\mathtt{s} = \mathtt{r} + \tau_3$. Enriched automaton $\mathcal{A}_{env}$ is represented in Figure 5.

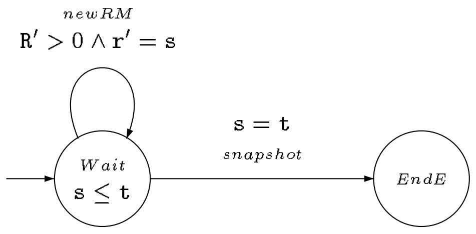


Fig. 5. Enriched automaton $\mathcal{A}_{env}$ modeling arrivals of RM cells and snapshot

More precisely, let Aux_i $(1 \leq i \leq 10)$ be:

$(\lambda = \ell_+ \wedge \mathtt{tfi} \leq \mathtt{s} \leq \mathtt{t}) \Rightarrow \mathtt{A} \geq \mathtt{E}$ $\quad(\mathrm{Aux}_1)$
$(\lambda = \ell_- \wedge \mathtt{s} \leq \mathtt{t} < \mathtt{tfi}) \Rightarrow \mathtt{A} \geq \mathtt{E}$ $\quad(\mathrm{Aux}_2)$
$(\lambda = \ell_- \vee \lambda = \ell_+) \Rightarrow \mathtt{tfi} \leq \mathtt{tla} \leq \mathtt{r} + \tau_2$ $\quad(\mathrm{Aux}_3)$
$((\lambda = \ell_- \vee \lambda = \ell_+) \wedge \mathtt{A} < \mathtt{Efi}) \Rightarrow \mathtt{tfi} \leq \mathtt{r} + \tau_3$ $\quad(\mathrm{Aux}_4)$
$((\lambda = \ell_- \vee \lambda = \ell_+) \wedge \mathtt{Efi} < \mathtt{Ela}) \Rightarrow \mathtt{tla} \leq \mathtt{r} + \tau_3$ $\quad(\mathrm{Aux}_5)$
$(\lambda = \ell_- \vee \lambda = \ell_+) \Rightarrow \mathtt{Emx} = max(\mathtt{Efi}, \mathtt{Ela})$ $\quad(\mathrm{Aux}_6)$
$(\lambda = \ell_- \vee \lambda = \ell_+) \Rightarrow \mathtt{Ela} = \mathtt{R}$ $\quad(\mathrm{Aux}_7)$
$((\lambda = \ell_- \vee \lambda = \ell_+) \wedge \mathtt{tfi} = \mathtt{tla}) \Rightarrow \mathtt{Efi} \geq \mathtt{Ela}$ $\quad(\mathrm{Aux}_8)$
$((\lambda = \ell_- \vee \lambda = \ell_+) \wedge \mathtt{tfi} \leq \mathtt{t} < \mathtt{tla}) \Rightarrow \mathtt{E} \leq \mathtt{Efi}$ $\quad(\mathrm{Aux}_9)$
$((\lambda = \ell_- \vee \lambda = \ell_+) \wedge \mathtt{tla} \leq \mathtt{t}) \Rightarrow \mathtt{E} \leq \mathtt{Ela}$ $\quad(\mathrm{Aux}_{10})$

Property $\mathrm{Inv} \equiv U \wedge \bigwedge_{i=1}^{10} \mathrm{Aux}_i$ is proved by inductive invariance, i.e. by showing that it holds initially and is preserved through any transition corresponding to either a complete action move or a delay move. The stable locations are ℓ_+, ℓ_-, ℓ_∞. The action moves starting from and leading to one of these locations are those associated with the reception of an RM cell, the reaching of `tfi`, or snapshot. In the case of RM cell reception, there are several subcases depending on the complete sequence of actions in $\mathcal{A}_{\mathcal{B}'}$ (*newRM* followed by [1a], [1b], [2], ... , or [8]}) and, subsidiarily, on the sequences in $\mathcal{A}_{\mathcal{I}}$ (*newRM* followed by [I1], ... , or [I3]}). We now give in details the statements involved in the proof. Variables of Inv are explicitly mentionned by writing $\mathrm{Inv}(\lambda, \mathtt{s}, \mathtt{r}, \overline{w})$, where $\overline{w}$ is the vector (E, Efi, Ela, Emx, A, R, tfi, tla). Note that, provided an encoding of locations $\ell_+, \ell_-, \ell_\infty$ is given as integers (e.g., $0, 1, 2$), all these statements are linear arithmetic formula over reals (involving variables $\lambda, \mathtt{s}, \mathtt{r}, \overline{w}$) and can be proved just by arithmetic reasoning. This can be done automatically with an arithmetic theorem prover. Such a proof was actually done using Coq [18], then was reformulated in the present context, after encoding p-automata in Coq. Let us recall that, in Coq, the user states definitions and theorems, then he proves the latters by means of *scripts* made of *tactics*. Scripts are not proofs, but *produce* proofs, which are data (terms) to be checked by the kernel of the proof assistant. Some tactics used for ABR were described in [18]. The script written for the ABR is about 3500 lines long and required about 4 man-months of work. A crucial part of the human work consisted in identifying the relevant invariants. Around two hundred subproofs were then automatically produced and checked. The whole proof check takes 5 minutes on a PC 486 (33 MHz) under Linux.

In order to give a flavour of the proof structure, we give now some typical statements to be proved in the case of action and delay moves.

Action moves. As an example, consider the reception of an RM cell where the subsequent action of $\mathcal{A}_{\mathcal{B}'}$ is [1b]. This corresponds to a complete sequence of transitions from location ℓ_- to itself. We have to prove:

$$\begin{aligned}
&(\mathtt{r}' = \mathtt{s} < \mathtt{tfi} = \mathtt{tla} < \mathtt{s} + \tau_3 = \mathtt{tla}' \wedge \mathtt{s} \leq \mathtt{t} \wedge\ \mathtt{R}' > 0 \\
&\wedge \mathtt{Emx} \leq \mathtt{Emx}' = \mathtt{Ela}' = \mathtt{R}' \ \wedge\ \mathtt{E}' = \mathcal{I}(\mathtt{s}, \mathtt{R}', \mathtt{E}) \\
&\wedge\ \mathrm{Inv}(\ell_-, \mathtt{s}, \mathtt{r}, \overline{w})) \Rightarrow \mathrm{Inv}(\ell_-, \mathtt{s}, \mathtt{r}', \overline{w}')
\end{aligned}$$

The conclusion $\mathrm{Inv}(\ell_-, \mathtt{s}, \mathtt{r}', \overline{w}')$ is a conjunction of the form $\bigwedge_{i=2}^{10} \mathrm{Aux}_i'$ with:

$$\mathtt{s} \leq \mathtt{t} < \mathtt{tfi} \Rightarrow \mathtt{A} \geq \mathtt{E}' \qquad (\mathrm{Aux}_2')$$
$$\mathtt{tfi} \leq \mathtt{tla}' \leq \mathtt{r}' + \tau_2 \qquad (\mathrm{Aux}_3')$$
$$\mathtt{A} < \mathtt{Efi} \Rightarrow \mathtt{tfi} \leq \mathtt{r}' + \tau_3 \qquad (\mathrm{Aux}_4')$$
$$\mathtt{Efi} < \mathtt{Ela}' \Rightarrow \mathtt{tla}' \leq \mathtt{r}' + \tau_3 \qquad (\mathrm{Aux}_5')$$
$$\mathtt{Emx}' = \mathit{max}(\mathtt{Efi}, \mathtt{Ela}') \qquad (\mathrm{Aux}_6')$$
$$\mathtt{Ela}' = \mathtt{R}' \qquad (\mathrm{Aux}_7')$$
$$\mathtt{tfi} = \mathtt{tla}' \Rightarrow \mathtt{Efi} \geq \mathtt{Ela}' \qquad (\mathrm{Aux}_8')$$
$$\mathtt{tfi} \leq \mathtt{t} < \mathtt{tla}' \Rightarrow \mathtt{E}' \leq \mathtt{Efi} \qquad (\mathrm{Aux}_9')$$
$$\mathtt{tla}' \leq \mathtt{t} \Rightarrow \mathtt{E}' \leq \mathtt{Ela}' \qquad (\mathrm{Aux}_{10}')$$

For example, let us show how to prove Aux_9', i.e. $\mathtt{E}' \leq \mathtt{Efi}$ assuming $\mathtt{tfi} \leq \mathtt{t} < \mathtt{tla}'$. By Aux_{10}, we have $\mathtt{E} \leq \mathtt{Ela}$ (since $\mathtt{tla} \leq \mathtt{t}$). By Aux_8, we have $\mathtt{Ela} \leq \mathtt{Efi}$ (since tfi=tla). Hence, by transitivity: $\mathtt{E} \leq \mathtt{Efi}$. On the other hand, E'=E by [I3] (since $\mathtt{t} < \mathtt{s} + \tau_3 (= \mathtt{tla}')$). Hence $\mathtt{E}' \leq \mathtt{Efi}$. All the other cases are proved similarly to this one, by case analysis, use of transitivity of $\leq, <, =$, and regularity of $+$ over these relations.

Delay moves. They take place at stable locations ℓ_+, ℓ_- and ℓ_∞. The corresponding properties to be proved are respectively:

$\texttt{tfi} \le \texttt{s} \le \texttt{s}' \le \texttt{t} \wedge \mathrm{Inv}(\ell_+, \texttt{s}, \texttt{r}, \overline{w}) \Rightarrow \mathrm{Inv}(\ell_+, \texttt{s}', \texttt{r}, \overline{w})$.

$\texttt{s} \le \texttt{s}' \le \texttt{tfi} \wedge \texttt{s}' \le \texttt{t} \wedge \mathrm{Inv}(\ell_-, \texttt{s}, \texttt{r}, \overline{w}) \Rightarrow \mathrm{Inv}(\ell_-, \texttt{s}', \texttt{r}, \overline{w})$.

$\texttt{s} \le \texttt{s}' \wedge \mathrm{Inv}(\ell_\infty, \texttt{s}, \texttt{r}, \overline{w}) \Rightarrow \mathrm{Inv}(\ell_\infty, \texttt{s}', \texttt{r}, \overline{w})$.

These formula easily reduce to:

$\texttt{tfi} \le \texttt{s} \le \texttt{s}' \le \texttt{t} \wedge \mathrm{Inv}(\ell_+, \texttt{s}, \texttt{r}, \overline{w}) \Rightarrow \texttt{A} \ge \texttt{E}$.

$\texttt{s} \le \texttt{s}' \le \texttt{tfi} \wedge \texttt{s}' \le \texttt{t} \wedge \mathrm{Inv}(\ell_-, \texttt{s}, \texttt{r}, \overline{w}) \Rightarrow \texttt{A} \ge \texttt{E}$.

$\texttt{s} \le \texttt{s}' \wedge \mathrm{Inv}(\ell_\infty, \texttt{s}, \texttt{r}, \overline{w}) \Rightarrow \texttt{A} \ge \texttt{E}$.

The first (resp. second, third) implication is true because its conclusion $\texttt{A} \ge \texttt{E}$ follows from the hypothesis using the Aux_1-conjunct (resp. Aux_2-conjunct, U-conjunct) of Inv.

6.2 Verification by reachability analysis

In order to mechanically prove property U, we have to compute $Post^*$ for the product automaton $\mathcal{T}$, starting from its initial state

$q_{init} : (\lambda = \ell_+) \wedge \psi_{init}$,

where ψ_{init} is the p-constraint $\texttt{s=tfi=tla} \wedge \texttt{R=E=A=Efi=Ela=Emx} \wedge \texttt{0<}\tau_3\texttt{<}\tau_2$. We then have to check that $Post^*(q_{init})$ does not contain any state where the property U is violated. Recall that property U can be stated as:

$\lambda = \ell_\infty \Rightarrow \texttt{A} \ge \texttt{E}$.

The state where U does not hold is then

$q_{\neg U} : (\lambda = \ell_\infty) \wedge \texttt{A} < \texttt{E}$

Automata $\mathcal{A}_{env}$, $\mathcal{A}_{\mathcal{I}}$ and $\mathcal{A}_{\mathcal{B}'}$ can be directly implemented into HyTech [14], which automatically computes the synchronized product $\mathcal{T}$. The modelling of the protocol and property as p-automata and the encoding in HyTech required about 3 man-months of work. The forward computation of $Post^*(q_{init})$ requires 23 iteration steps and its intersection with $q_{\neg U}$ is checked to be empty. (This takes 8 minutes on a SUN station ULTRA-1 with 64 Megabytes of RAM memory. See Appendix B for a display of the generated p-zone at ℓ_∞.) This achieves an automated proof of correctness of $\mathcal{B}'$. Such a proof first appears along these lines in [8]. Note that HyTech can provide as well a proof by *backward* reasoning (using Pre^* instead of $Post^*$).

7 Discussion on p-automata

Tools based on timed automata have been successfully used in the recent past for verifying or correcting real-life protocols (e.g., the Philips Audio Control protocol [16] and the Bang & Olufsen Audio/Video protocol [13]). Experiences with such tools are very promising. This observation led us to use here p-automata, a close variant of timed automata. The differences between p-automata and classical timed automata are two-fold. A first minor difference is that p-automata use a form of "updatable" time variables instead of traditional clocks (but see [9] for a proof of equivalence between the two classes). Second p-automata incorporate *parameters*, which are essential in our case study. The choice of HYTECH, as an associated tool, is natural in this context. We now explain some new features that appear in the proof of the protocol using p-automata, and some difficulties encountered in the process of building the specification.

7.1 Towards p-automata

We first point out some significant differences between the proof by invariants as stated by Monin and Klay [19, 18] and the corresponding proof presented here (see section 6.1).

Representation of time. In the work reported in [19, 18], the formalization of time aspects, though influenced by timed automata, was performed using *ad hoc* devices, appealing to the reader's natural understanding. This problem is settled here, thanks to p-automata, which include a built-in notion of clock and rely on a well-understood and widely accepted notion of time. The use of p-automata, although it introduces an additional level of encoding, thus makes the effort of specification easier with this respect. Note however that the encoding of p-automata in COQ does not specify *a priori* the granularity of time evolution, and allows for either a continuous or discrete underlying model of time. It is based on a minimal underlying theory of arithmetic (mainly the transitivity of relations $<, \leq$ and the regularity of $+$ over them). This is not the case in the theory of timed automata and the associated tools like HYTECH (or others [6, 11]) where the time domain is assumed to be continuous ($\mathbb{Q}$ or $\mathbb{R}$), and a sophisticated package for manipulating linear constraints over *real* arithmetic is used. We will come back to this difference (see section 8.2).

Higher-order vs. first-order specifications. As recalled in section 3, Monin and Klay [19] introduced an *incremental* way of computing the ideal rate $\mathrm{Acr}(t)$ (higher-order algorithm $\mathcal{F}$). We then recast their algorithm $\mathcal{F}$ under a parametric form $\mathcal{I}$. The latter view is probably less natural, but fits better into the first order framework of p-automata.

Reformulating the proof by invariance. We also rewrote the proof by invariance of [19] in the context of p-automata, in order to assess the proofs in a uniform framework. Expressing the auxiliary properties needed in this proof required to (re)introduce history variable $\mathtt{r}$ accounting for the reception time of the last RM cell. Moreover, these properties only concerned *stable* locations of the system: expressions of the form $\lambda = \ell$ had to be included, and only complete sequences of actions were considered. Note that the auxiliary property Aux_8: $\mathtt{tfi} = \mathtt{tla} \Rightarrow \mathtt{Efi} \geq \mathtt{Ela}$, is different from its counterpart Aux_8^0: $\mathtt{tfi} = \mathtt{tla} \Rightarrow \mathtt{Efi} = \mathtt{Ela}$, as found in [19]. Actually, Aux_8^0 is *false* in our model. We will subsequently explain this discrepancy (see section 8.2).

7.2 Specific modelling problems with p-automata

In the process of constructing p-automata while keeping with the specification, we had to face some problems, which are listed below.

Modelling the environment as a p-automaton. In addition to the automata corresponding naturally to $\mathcal{I}$ and $\mathcal{B}'$, a third automaton was introduced to model the environment, thus providing a clear separation between external and internal events.

Introducing urgent locations. The class of update relations in p-automata (derived from HyTech) does not allow for *simultaneous* updates. For instance, choosing (at random) a new identical value of `R` and `E` (with an instruction like `E'=R'>0`) is forbidden. In order to implement such an update relation, an *urgent* intermediate location, such as *UpdE* depicted in figure 6, had to be introduced.

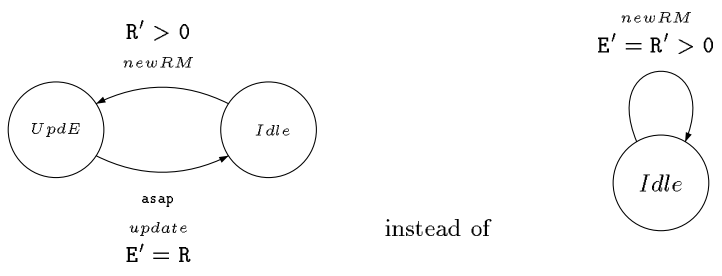


Fig. 6. Reception and update with two locations instead of one

Introducing two stable locations. In order to implement the higher priority of operation "reaching `tfi`" (when occuring simultaneously with other actions of the system), we were led to create *two* stable locations (*Greater* and *Less*) in the p-automaton representing algorithm $\mathcal{B}'$. Note that we overlooked this priority requirement in a preliminary implementation embedding only one stable location, which entailed a violation of property U. (This was detected subsequently by running HyTech.)

8 Proof Comparison

We now assess the respective merits and shortcomings of the proof methods by invariance and reachability analysis within the unified framework of p-automata, regarding the ABR conformance problem. We also explain how to cross-fertilize the results of the two methods.

8.1 Automated proof vs. readable proof

It is well-known that a proof in a model-checker is more automatic, but that more insight in the algorithm is gained by doing the proof with a theorem-prover. Let us confirm this general opinion in our particular case study.

As already noticed, the reachability proof was done in a fully automatic manner (via HyTech). This is an outstanding advantage over the proof by inductive invariance (which required the human discovery of several nontrivial auxiliary properties) and justifies *a posteriori* our effort of translating the problem into the formalism of p-automata. In particular, it becomes easier to validate other ABR conformance protocols as soon as they are formalized themselves in terms of p-automata. This is actually what was done recently in the framework of RNRT project *Calife*: different variants of $\mathcal{B}'$ were easily checked (or invalidated) with HyTech by reachability analysis along the lines described above. It was not possible to do the same with the inductive invariance approach because several of the original auxiliary properties became false while others had to be discovered.

Nevertheless, several qualifications must be done about this positive side of model-checking. Let us first stress that the proof obtained by reachability analysis, merely consists of a long list of constraints (see appendix B) that represents the whole set of reachable symbolic states. This information is hardly exploitable by a human: in particular the essential fact that such a list is *complete* (i.e. "covers" all the reachable states) is impossible to grasp by hand. In contrast, the invariance proofs as checked by a theorem prover are more human-oriented. It is instructive to inspect the case analysis that was automatically performed, and allows the reader to be convinced of the property accurateness (or *a contrario* of some flaws). Besides, the auxiliary properties are very important *per se*, and bring important information about algorithm $\mathcal{B}'$ itself. (Some of these properties are indeed part of norm ITU I.371.1, and must be henceforth fulfilled by any new ABR algorithm candidate to normalization.)

We explain now how one can go beyond the limitations of each method, by using both of them in a fruitful cross-fertilizing way.

8.2 Cross-fertilizing proofs

Checking the output produced by Hytech. A proof produced by HyTech, i.e. the (finite) list $Post^*$ of all the symbolic states reached from the initial one, can be seen as a *fixed-point* associated with the set of transitions of the product automaton. Therefore, one can verify that such a list is "complete" (covers all the reachable states) by checking that it is *invariant* through action and delay moves. This can be done, using the Coq system, exactly as explained in section 6.1. This gives of course an increased confidence in the model-checking proof. In addition it may give new insight about the conditions on the environment that were assumed to perform the proof: as noticed in section 7.1, in Coq, we use a very flexible model for time, assuming only that time increases (but nothing about its continuity). In fact, the correctness of algorithm $\mathcal{B}'$ holds even for a discrete time modelling. Such a feature cannot not be derived from the proof of HyTech, since it uses *a priori* an assumption of continuous time evolution.

Checking the invariants used in Coq. In the other way around, one can check the correctness of the auxiliary properties $\mathrm{Aux}_1, \cdots \mathrm{Aux}_{10}$ of COQ proof, simply by asking HYTECH if all the reachable states satisfy them (i.e. *Post** $\subseteq$ Aux_i, for all $i = 1, \cdots, 10$). The answer is always "yes", which gives us another proof of the Aux_is. Recall however that Aux_8 differs from its counterpart Aux_8^0 in [19]: $\mathtt{tfi} = \mathtt{tla} \Rightarrow \mathtt{Efi} = \mathtt{Ela}$. Actually Aux_8^0 is false in the p-automata model and true in the original model of Monin-Klay (see section 7.1). This discrepancy originates from the different ways two consecutive RM cells follow each other in the two models. In our p-automata model, two consecutive RM cells may arrive simultaneously while this is precluded in the model of Monin-Klay as reception times of RM cells must form a *strictly* increasing sequence. The model presented here is then more general than the original model of Monin-Klay, as it relaxes some assumption concerning the sequence of RM cells. As a by-product, this provides us with a better understanding of the conditions under which $\mathcal{B}'$ behaves correctly.

Finally note that the model of p-automata is flexible enough to incorporate the assumption of strictly increasing sequences of RM cells, as used in [19]: it suffices to use explicitly the additional variable $\mathtt{r}$ mentioned in section 7.1 (date of last RM cell reception), and add guard $\mathtt{s} > \mathtt{r}$ to the *newRM* transition in the environment automaton of figure 5. With such a modification, property Aux_8^0 also becomes true in the model with p-automata.

8.3 Further experiments and foreseen limits

We thus claim that checking properties proved by one tool, using the other one, is very fruitful. As examplified above, it may reveal possible discrepancies, which lead in turn to discover implicit modelling assumptions. It may also of course detect real flaws, which originate from the protocol or its modelling (although it has not been the case here). In any case this proof confrontation helps the verification work, and increases the confidence of the human in mechanical proofs. One can now wonder how general are the remarks we made on this case study, given the fact that we focused on one problem (the correctness of algorithm $\mathcal{B}'$), and used two specific tools as a theorem-prover (COQ) and a model-checker (HYTECH).

Regarding the tools, we believe that our experience with COQ and HYTECH is not specific, but can be reproduced with equivalent tools as well. We have concrete indications in this sense. Actually, in the framework of project *Calife*, Pierre Castéran and Davy Rouillard, from University of Bordeaux, have performed a proof similar to the proof in COQ, using the model of p-automata and theorem prover ISABELLE [10, 25]. Concerning HYTECH, we do not know any other model-checking tool allowing for parameters but, as mentioned in appendix C, we did some successful experiments with GAP [12], a tool based on constraint logic programming, which works as a fixed-point engine very much as HYTECH generates *Post**.

Concerning the studied problem, the success of the proof by model-checking comes from the fact that the computation of the *Post** computation with HYTECH had terminated. This can be considered a "lucky" event, since analysis of such a parametric algorithm is known to be undecidable [5]. This means that computation of *Post** does not always terminate for all p-automata. (This observation leads us to propose, in Appendix C, an "approximate" version of $\mathcal{B}'$, belonging to a subclass for which *Post** is guaranteed to terminate.) Is such a termination property preserved when considering ABR conformance algorithms other than $\mathcal{B}'$? The answer is ambivalent. On the one hand, as already mentioned, our model-checking experience on $\mathcal{B}'$ was successfully reused on other (relatively close) algorithms of ABR conformance in the framework of project *Calife*. On the other hand, we failed to mechanically check an algorithm of ABR conformance of a different kind: the generic algorithm of Rabadan-Klay (see e.g. [26]). This algorithm involves an unbounded list of N scheduled dates (instead of 2, as in $\mathcal{B}'$), and cannot be modeled with p-automata due to the use of a list data type. Even for the restricted version where N is bound to a small value, e.g: 3, in which case we get a natural model with p-automata, HYTECH runs out of memory and fails to generate *Post**.

The latter experiment recalls us some inherent limits of model-checking: if the algorithm uses not only a finite set of numeric variables, but also unbounded data structures (such as lists), then the verification process has to rely essentially on classical methods of theorem-proving; this is also true when the program can be modeled as a p-automaton, but the space of reachable symbolic states is too big to be computed by existing model-checkers.

9 Conclusion

As a recapitulation, we believe that many useful informations about real-time programs can be obtained without resorting to new integrated tools, when it is possible to make a joint use of well-established theorem prover and model checker. In our case, we gained much insight about the algorithm $\mathcal{B}'$, and important confidence in the proofs of correctness produced by COQ and HYTECH, basically by using the unified framework of p-automata, and cross-fertilizing the two proofs. In particular we saw that algorithm $\mathcal{B}'$ is robust in the sense that several underlying assumptions can be relaxed: the nature of time can be discrete (instead of continuous); the (measured) time interval between two received RM cells can be null. Moreover, the basic p-automata model underlying $\mathcal{B}'$ was successfully reused for proving the correctness of some variants. To our knowledge, it is the first time that such a compared study between theorem proving and model checking has been performed on the same industrial problem. We hope that this work paves the way for further experiences on real-life examples. In the framework of project *Calife*, we are currently developing a two-step methodology for verifying the quality of new services provided by telecommunication networks, which exploits the synergy between the two proof methods: the first step, based on model-checking, yields a p-automaton model endowed with a collection of invariants it satisfies; in the second step, the p-automaton is recasted under an algorithmic form better suited to the end-user, and verification is done via a generic proof assistant, with the help of invariants.

References

1. M. Abadi and L. Lamport. "The existence of refinement mappings". *TCS 82:2*, 1991, pp. 253–284.
2. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis and S. Yovine. "The Algorithmic Analysis of Hybrid Systems". *Theoretical Computer Science 138:3*, 1995, pp. 3–34.
3. R. Alur, C. Courcoubetis, T.A. Henzinger and P.-H. Ho. "Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems". *Hybrid Systems I*, LNCS 736, 1993, pp. 209–229.
4. R. Alur and D. Dill. "Automata for Modeling Real-Time Systems". *Proc. 17th ICALP*, LNCS 443, 1990, pp. 322–335.
5. R. Alur, T.A. Henzinger, M. Vardi. "Parametric real-time reasoning". *Proc. 25th Annual ACM Symp. on Theory of Computing (STOC)*, 1993, pp. 592–601.
6. J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson and W. Yi. "UPPAAL – a Tool Suite for Automatic Verification of Real-Time Systems". *Hybrid Systems III*, LNCS 1066, 1996, pp. 232-243.
7. B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliâtre, E. Giménes, H. Herbelin, G. Huet, P. Manoury, C. Munõz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi and B. Werner. "The Coq Proof Assistant User's Guide, version 6.1", INRIA Rocquencourt and CNRS-ENS Lyon, 1996.
8. B. Bérard and L. Fribourg. "Automated verification of a parametric real-time program: the ABR conformance protocol". In *Proc. 11th Int. Conf. Computer Aided Verification (CAV'99)*, LNCS 1633, 1999, pp. 96-107.(see also http://www.lsv.ens-cachan.fr/Publis/).
9. P. Bouyer, C. Dufourd, E. Fleury and A. Petit. "Are Timed Automata Updatable?". In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'00)*, LNCS, 2000.
10. P. Castéran and D. Rouillard. "Reasoning about parametrized automata". *Proc. Real Time Systems 2000.*
11. C. Daws, A. Olivero, S. Tripakis and S. Yovine. "The Tool KRONOS". *Hybrid Systems III*, LNCS 1066, 1996, pp. 208–219.
12. L. Fribourg. "A Closed-Form Evaluation for Extended Timed Automata". *Technical Report LSV-98-2*, CNRS & Ecole Normale Supérieure de Cachan, March 1998. (http://www.lsv.ens-cachan.fr/Publis/)
13. Klaus Havelund, Arne " Skou, Kim G. Larsen and Kristian Lund. Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL". *Proc. 18th IEEE Real-Time Systems Symposium.* San Francisco, California, USA, 1997, pp. 2–13.
14. T. Henzinger, P.-H. Ho and H. Wong-Toi. "A User Guide to HYTECH". *Proc. TACAS'95*, LNCS 1019, 1995, pp. 41–71.
15. ITU-T Recommendation I.371.1. "Traffic control and congestion control in B-ISDN", 1997.
16. Kim G. Larsen, Paul Pettersson and Wang Yi. "Model-Checking for Real-Time Systems". *Proc. 10th International Conference on Fundamentals of Computation Theory*, LNCS 965, 1995, pp. 62–88.

17. Z. Manna. "Beyond model checking". *CAV'94*, LNCS 818, Springer-Verlag, 1994, pp. 220–221.
18. J.F. Monin. "Proving a real time algorithm for ATM in Coq". *Types for Proofs and Programs*, LNCS 1512, 1998, pp. 277–293.
19. J.-F. Monin and F. Klay. Correctness Proof of the Standardized Algorithm for ABR Conformance. In *Formal Methods 99*, LNCS 1708. Springer Verlag, 1999, pp. 662–681.
20. X. Nicollin, A. Olivero, J. Sifakis and S. Yovine. "An Approach to the Description and Analysis of Hybrid Systems". *Hybrid Systems I*, LNCS 736, 1993, pp. 149–178.
21. A. Pnueli and E. Shahar. "A Platform for Combining Deductive with Algorithmic Verification". *CAV'96*, LNCS 1102, Springer-Verlag, 1996, pp. 184–195.
22. Christophe Rabadan. L'ABR et sa conformité. NT DAC/ARP/034, CNET, 1997.
23. P.Z. Revesz. "A Closed-Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints", *Theoretical Computer Science*, 1993, vol. 116, pp. 117-149.
24. S. Rajan, N. Shankar and M.K. Srivas. "An Integration of Model Checking with Automated Proof Checking". *CAV'95*, LNCS 939, Springer-Verlag, 1995, pp. 84–97.
25. D. Rouillard. "Formalisation dans CClair de la preuve de conformité de l'algorithme $\mathcal{B}'$". *Unpublished manuscript*, February 2000, 15 pages.
26. M. Rusinowitch, S. Stratulat and F. Klay. "Mechanical Verification of a Generic Incremental ABR Conformance Algorithm". In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'00)*, LNCS, 2000.
27. A.U. Shankar. "An Introduction to Assertional Reasoning for Concurrent Systems." *ACM Computing Surveys* 25:3, 1993, pp. 225–262.

Appendix A: Pseudocode of $\mathcal{B}'$

- when a new RM cell (with value `R`) arrives:

```
if s < tfi then
   if Emx <= R then
      if tfi < s+tau3 then
         if s+tau3 < tla or tfi=tla then
            [1] Emx:= R; Ela:= R; tla:= s+tau3
         else
            [2] Emx:= R; Ela:= R
      else
         if A <= R then
            [3] Emx:= R; Ela:= R; Efi:= R; tfi:= s+tau3; tla:= s+tau3
         else
            [4] Emx:= R; Ela:= R; Efi:= R; tla:= tfi
   else
      if R < Ela then
         [5] Efi:= Emx; Ela:= R; tla:= s+tau2
      else
         [6] Efi:= Emx; Ela:= R
else
   if A <= R then
      [7] Efi:= R; Ela:= R; Emx:= R; tfi:= s+tau3; tla:= s+tau3
   else
      [8] Efi:= R; Ela:= R; Emx:= R; tfi:= s+tau2; tla:= s+tau2
```

- when current time `s` reaches `tfi`:

```
   [9]  tfi:= tla; A:= Efi; Efi:= Ela; Emx:= Ela
```

If a new RM cell arrives at `tfi`, the action for `s=tfi` should be done before the one for the new RM cell

Appendix B: p-zone at ℓ_∞

The p-zone at location ℓ_∞ is a disjunction of 64 convex constraints, some of which are listed below in the HyTech format ($\texttt{t} \leq \texttt{s} \wedge 0 < \texttt{tau3} < \texttt{tau2}$ being always implicit).

```
    tau3 + tau2 <= tfi   &  tfi + tau2 <= tla + tau3
& tla <= t + tau2   & t < tfi   & A <= Efi  & Ela < Efi   & 0 < Ela
& 0 < E   & E <= A   & 0 < A
|
    tfi + tau3 <= t + tau2   & 0 < Ela   & Efi < A   & E < A
& Ela < Efi   & 0 < E   & tla <= t + tau2   & t < tfi
& t + tau2 < tla + tau3   & 2tau2 <= tla   & tau3 + tau2 <= tfi
|
    E < A   & 2tau2 <= tla   & tfi < tla   & Efi <= E   & 0 < Ela
& Ela < Efi   & tla + tau3 <= t + tau2   & t < tfi
|
    tau3 + tau2 < tfi   & t + tau3 < tfi   & tla <= t + tau2
& t + tau2 < tfi + tau3   & tfi <= tla     & Efi < A   & Ela < Efi
& 0 < Ela   & E <= A   & 0 < E
|
     tau2 < 2tau3   & tau2 < t   & Ela < Efi   & 0 < Ela   & 0 < E
& tfi + tau2 <= tla + tau3   & 2tau3 < tfi   & t < tfi   & E <= A
& A <= Efi   & tla <= t + tau2
|
     tau2 < 2tau3   & tau2 < t   & 0 < Ela   & Efi < A   & Ela < Efi
& tla <= t + tau2   & E <= A   & tfi <= tla   & t + tau2 < tfi + tau3
& tau3 + tau2 < tfi   & 0 < E
|
    Efi = Ela   & tfi = tla   & 2tau2 <= tfi   & 0 < E   & 0 < Efi
& E <= A   & Efi < A   & tfi <= t + tau2   & t + 2tau2 < tfi + 2tau3
|
    Efi = Ela   & tfi = tla   & 2tau2 < tfi   & E <= A   & 0 < Efi
& 0 < E   & Efi < A   & tfi <= t + tau2   & t + tau2 < tfi + tau3
|
    Efi <= Ela   & 0 < Efi   & 2tau3 <= tla   & tfi < tla
& tla <= t + tau3   & t + 2tau3 < tla + tau2   & t + tau2 < tfi + tau3
& E <= A   & 0 < A   & 0 < E
|
    Efi <= Ela   & 0 < Efi   & tfi < tla   & t + tau2 < tfi + tau3
& tau2 <= t   & tla <= t + tau3   & E <= A   & 0 < A   & 0 < E
& 2tau3 <= tla
|
    E < A   & 0 < E   & Efi <= Ela   & 0 < Efi   & tfi < tla
& tau2 <= t   & t < tfi   & tfi + tau3 <= t + tau2   & tla <= t + tau3
& 2tau3 <= tla
|
    2tau2 <= tfi   & tfi <= tla   & tla <= t + tau2
& t + 2tau2 < tfi + 2tau3   & Ela < Efi   & 0 < Ela   & Efi < A
& 0 < E   & E <= A
```

Appendix C: Approximation Model

One drawback with p-automata is that computation $Post^*$ is not guaranteed to terminate. We now explain a way of always guaranteeing termination by replacing guards and update relations of p-automata by overapproximations. These overapproximations are p-constraints of a restricted form, which ensures that the computation of $Post^*$ always terminates. (These constraints are called "gap-order" constraints, and the termination comes from Revesz's result [23].) The price to be paid is that the generated set of states is a superset of what is computed using $Post^*$ with the original automaton. Even for this superset, the inequation $\mathtt{A} \geq \mathtt{E}$ still holds at location ℓ_∞. This provides us with yet another proof of correctness of $\mathcal{B}'$. A proof along these lines first appeared in [12].

In order to describe the approximate computation, it is convenient to reason not with our original model of p-automata, but with an equivalent variant, where delay-moves are replaced with specific action moves.

A variant of p-automata

We now revisit the modelling of the system using a variant of p-automata. In the original model, there were two kinds of transitions: action moves and delay moves. The time was evolving at every (non-urgent) location through delay moves. In the new variant, we model time evolution through a specific *action* move, labeled *tick*, which loops on every (non-urgent) location. (There is no more delay moves.) Time variable $\mathtt{s}$ is not any longer a clock but a discrete variable, which is updated by a *tick* action. In this variant, urgent locations are those for which there is no *tick* action. Accordingly there is no more need for specifying urgent actions (with label *asap*). In the new model, automaton $\mathcal{A}_{env}$ is represented on figure 7. Automata $\mathcal{A}_{\mathcal{I}}$ and $\mathcal{A}_{\mathcal{B}'}$ are obtained in the same way, by adding *tick* transitions to the locations *Idle*, *Less* and *Greater*.

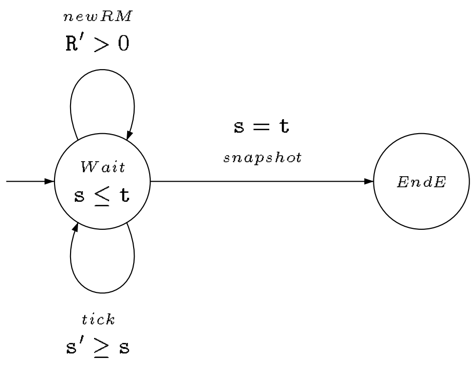
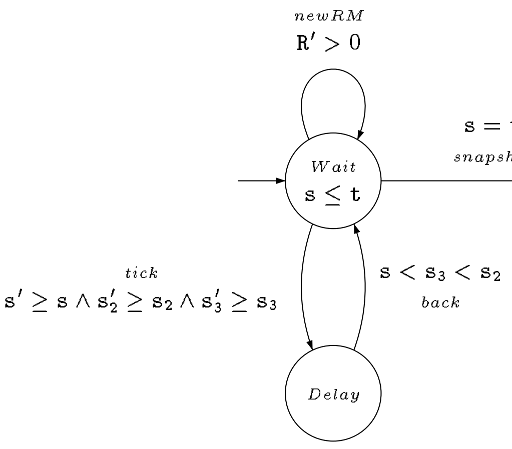


Fig. 7. Revisited automaton $\mathcal{A}_{env}$ modeling arrivals of RM cells and snapshot

Approximation Automata

Approximation automata are obtained by replacing in each revisited automata expressions of the form $\mathtt{s}+\tau_2$ and $\mathtt{s}+\tau_3$ with two new variables $\mathtt{s}_2$ and $\mathtt{s}_3$ respectively. These new variables are updated through the *tick* action, by increasing them together with $\mathtt{s}$. An additional guard $\mathtt{s} < \mathtt{s}_3 < \mathtt{s}_2$ is also introduced to preserve the ordering of these variables (originating from the fact that $0 < \tau_3 < \tau_2$). This leads to a new automaton for the environment represented in Figure 8. Note that the new modeling introduces a loss of precision since the fact that the difference between $\mathtt{s}$ and $\mathtt{s}_3$, as well as the difference between $\mathtt{s}$ and $\mathtt{s}_2$, are not any longer constant but may now vary in time (at each *tick* action).



newRM
$\mathtt{R}' > 0$
$\mathtt{s} = \mathtt{t}$
snapshot
Wait
$\mathtt{s} \leq \mathtt{t}$
EndE
tick
$\mathtt{s}' \geq \mathtt{s} \wedge \mathtt{s}_2' \geq \mathtt{s}_2 \wedge \mathtt{s}_3' \geq \mathtt{s}_3$
$\mathtt{s} < \mathtt{s}_3 < \mathtt{s}_2$
back
Delay

Fig. 8. Approximated automaton $\mathcal{A}_{env}$ modeling arrivals of RM cells and snapshot

Approximated p-automata $\mathcal{A}_{\mathcal{I}}$ and $\mathcal{A}_{\mathcal{B}'}$ are simply obtained by replacing expressions $\mathtt{s}+\tau_2$ and $\mathtt{s}+\tau_3$ with $\mathtt{s}_2$ and $\mathtt{s}_3$ respectively. Approximated p-automaton $\mathcal{A}_{\mathcal{I}}$ is thus represented Figure 9.

Experimental results with the approximated model

The forward computation of *Post** with HyTech now requires 24 iteration steps and its intersection with $q_{\neg U}$ is again checked to be empty. (This takes 371 sec. on a SUN station ULTRA-1 with 64 Megabytes of RAM memory.) The p-zone at location ℓ_∞ is now a disjunction of 23 convex constraints some of which are listed below in the HyTech format (expression $0 < \mathtt{s} = \mathtt{t} < \mathtt{s}_3 < \mathtt{s}_2$ being always implicit).

```
     Efi = E   & Ela = E   & tla = tfi   & A = E  & tfi < s
|
     s3 = tfi   & Efi = Ela  & E = A   & s3 = tla   & 0 < Efi
& E <= Efi
```

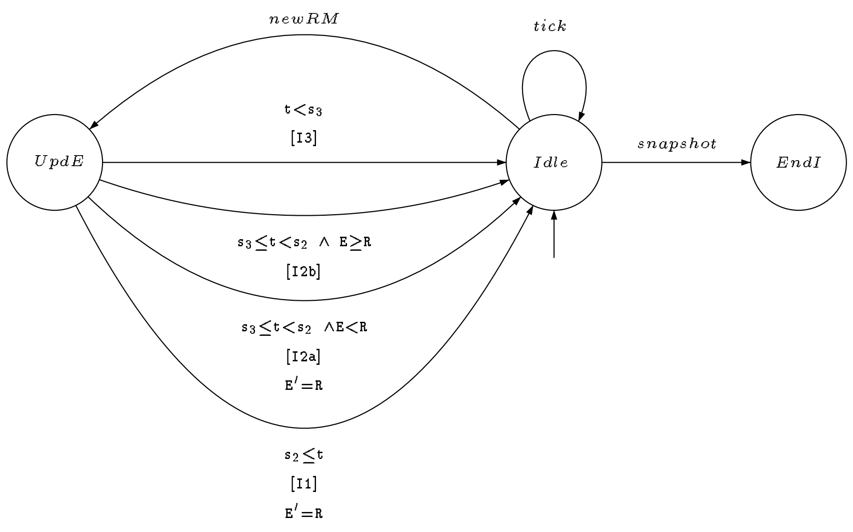


Fig. 9. Approximated automaton $\mathcal{A}_{\mathcal{I}}$

```
|
    Efi = Ela   & E = A   & tfi = tla  & 0 < Efi   & E <= Efi
& tfi < s3   & s < tfi
|
    A = E   & s3 = tfi  & s2 = tla   & E <= Efi   & 0 < Ela
& Ela < Efi
|
    s2 = tla   & A = E  & E <= Efi   & tfi < s3   & 0 < Ela
& Ela < Efi   & s < tfi
|
    A = E  & E <= Efi   & 0 < Ela   & Ela < Efi   & tfi < s3
& tfi < tla   & tla < s2   & s < tfi
|
    s = tla   & E = Ela   & s=tfi   & E = Efi  & 0 < E   & E < A
|
    A = E  & E <= Efi   & Efi <= Ela   & 0 < Efi   & tla <= s3
& tfi < tla   & s < tfi
|
    E = Efi   & E = Ela   & tfi = tla  & 0 < E   & E < A
& 0 < tla   & tla < s
|
    Efi = Ela   & tla = tfi   & s < tfi   & tfi < s2  & Ela < A
& 0 < Ela   & 0 < E   & E <= A
```