Automated verification of a parametric real-time program: the ABR conformance protocol

Béatrice Bérard and Laurent Fribourg *

LSV – Ecole Normale Supérieure de Cachan & CNRS
61 av. Pdt. Wilson - 94235 Cachan - France
email: {berard,fribourg}@lsv.ens-cachan.fr

Abstract. The ABR conformance protocol is a real-time program developed at France Telecom, that controls dataflow rates on ATM networks. A crucial part of this protocol is the dynamical computation of the expected rate of data cell emission. We present here a modelization of the corresponding program, using parametric timed automata. In this framework, a fundamental property of the service provided by the protocol to the user is expressed as a reachability problem. The tool HyTech is then used for computing the set of reachable states of the model, and automatically proving the property. This case study gives additional evidence of the importance of the model of parametric timed automata and the practical usefulness of symbolic analysis tools.

1 Introduction

Over the last few years, an extensive amount of research has been devoted to the formal verification of real-time concurrent systems. Among the various approaches to the analysis of timed models, one of the most successful is based on timed automata. Since its first introduction in [3], this model was extended with many different features, leading to the general notion of hybrid automata [1,2,15]. Although hybrid automata have an infinite number of states, the fixpoint computation of reachable states often terminates in practice, thus allowing the verification of "safety" properties. This explains the increasing success of the development of tools for the analysis of real-time systems [5,8,12], as well as the numerous industrial case studies which have already been presented. In this paper, we propose an automated verification of correctness for the *Available Bit Rate* (ABR) conformance protocol, developed by France Telecom at CNET (Centre National d'Etudes des Télécommunications, Lannion, France) in the context of network communications with *Asynchronous Transfer Mode* (ATM).

The ABR conformance protocol. ATM is a flexible packet-switching network architecture, where several communications can be multiplexed over a

* Supported by Action *FORMA* (Programme DSP-STTC/CNRS/MENRT)

same physical link, thus providing better performances than traditional circuit-switching networks. Different types of ATM connections are possible at the same time, according to the dataflow rate asked (and paid) for by the service user [9]. A contract with ABR connection makes it possible for a source to emit at any time with a rate depending on the load of the network: according to the available bandwidth, the ABR protocol dynamically computes the highest possible dataflow rate and sends this information, via so called *Resource Management* (RM) cells, to the user, who has to adapt his transfer rate of *data* (D) cells.

The service provider has to control the conformance of emission with respect to the currently allowed rate, and filter out D cells emitted at an excessive rate. This is achieved by a program located at an interface between the user and the network, which receives RM cells on their way to the user as well as D cells from the user to the network (see Figure 1). This program has two parts: the easy task

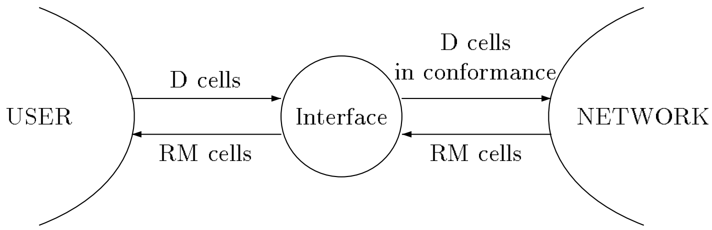

Fig. 1. Schematic view of cells traffic

is to compare the emission rate of D cells with the rate value currently allowed, while the difficult problem is to dynamically compute (or update) the rate values expected for future D cells. The program must take into account the delays introduced by the transit of cells from the interface to the user and back, but the exact value of this delay is not known: only lower and upper bounds, denoted a and b, are given. A simple algorithm called $\mathcal{I}$ computes, from a sequence of rate values carried by previously arrived RM cells, the *ideal (expected) rate* E_t for emission of D cells which will arrive at future time t. However, since the value of t is not known in advance, an implementation of $\mathcal{I}$ would require to store a huge number of rate values. A more realistic algorithm, called $\mathcal{B}'$, due to C. Rabadan, has been adopted by CNET. It stores only two RM cell rates, and dynamically updates an estimated value A of ideal rate E_t.

Correctness of program $\mathcal{B}'$. Before being accepted as an international standard (norm ITU I-371.1), this protocol had to be proved correct: it was necessary to ensure that the flow control of D cells by comparison with A rather than E_t is never disadvantageous to the user. This means that when some D cell arrives at time t, A is an *upper* approximation of E_t. In other words, $A \geq E_t$ when current time reaches t. This property U was proved by hand by Monin and Klay, using a classical method of invariants [14]. However, since this proof was quite difficult to obtain, CNET felt the need for using formal methods and tools to verify $\mathcal{B}'$ in a more mechanical way, as well as future versions of $\mathcal{B}'$ currently under development.

This paper presents a modelization of algorithms $\mathcal{I}$ and $\mathcal{B}'$ as *parametric timed automata* [4], and an automated proof of property U (viewed as a reachability problem) via tool HyTech [12].

Plan of the paper. Section 2 presents the model of parametric timed automata. Section 3 describes algorithms $\mathcal{I}$ and $\mathcal{B}'$, and correctness property U within this framework. Section 4 gives the experimental results obtained with HyTech and a comparison with previous work. Section 5 concludes with final remarks.

2 Parametric timed automata

We use here a model of parametric timed automata, called p-automata for short, which are extensions of timed automata [3] with parameters. A minor difference with the classical parametric model of Alur-Henzinger-Vardi [4] is that we have only one clock variable S and several "discrete" variables $w_1, ..., w_n$ while, in [4], 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 $S - w_i$ (see [10]). Alternatively, our parametric automata can simply be viewed as particular cases of *linear hybrid automata* [1,2,15].

P-automata. In addition to a finite set of locations, p-automata have a finite set P of *parameters*, a finite set W of *discrete variables* and a *universal clock* S. These are all real-valued variables which differ only in the way they evolve when time increases. Parameter values are fixed by an initial constraint and never evolve later on. Discrete variables 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 (without reset).

Formally, a *parametric term* is an expression of the form $w + \sum_{i=1}^{k} p_i + c$, $S + \sum_{i=1}^{k} p_i + c$ or $\sum_{i=1}^{k} p_i + c$, where $w \in W$, $p_i \in P$ and $c \in \mathbb{N}$. (As usual, by convention, a term without parameter corresponds to the case where $k = 0$.) An *atomic constraint* is an expression $term_1 \# term_2$, where $term_1, term_2$ are parametric terms and $\# \in \{<, \leq, =, \geq, >\}$. A *constraint* is a conjunction of atomic constraints. The formulas used in p-automata are *location invariants, guards* and *update relations*. A location invariant is a conjunction of atomic constraints. A guard is a conjunction of atomic constraints with possibly the special expression *asap*. An update relation is a conjunction of formulas of the form $w' \# term$ where w' belongs to a primed copy of W, *term* is a parametric term and $\# \in \{<, \leq, =, \geq, >\}$. As usual $w' = w$ is implicit if w' does not appear

in the update relation.
A *p-automaton* $\mathcal{A}$ is a tuple $\langle L, \ell_{init}, P, W, S, I, \Sigma, T\rangle$, where
- L is a finite set of locations, with *initial location* $\ell_{init} \in L$,
- P and W are respectively the sets of parameters and discrete variables, S is the universal clock,
- I is a mapping that labels each location ℓ in L with some location invariant, simply written I_ℓ instead of $I(\ell)$ in the following,
- Σ is a finite set of *labels* partitioned into *synchronization* labels and *internal* labels,
- T is a set of *action transitions* of the form $\langle \ell, \varphi, \sigma, \theta, \ell'\rangle$, where ℓ and ℓ' belong to L, φ is a guard, $\sigma \in \Sigma$ is a label and θ an update relation. The transition is *urgent* if its guard φ contains the symbol *asap*.

Semantics of p-automata. We briefly and informally recall the semantics of timed automata (see [4] for details), described in terms of transition systems. For a p-automaton $\mathcal{A}$, the (global) state space of the transition system is the set $Q_{\mathcal{A}} = L \times \mathbb{R}^P \times \mathbb{R}^W \times \mathbb{R}$ of tuples (ℓ, γ, v, s), where ℓ is a location of $\mathcal{A}$, $\gamma : P \mapsto \mathbb{R}$ is a *parameter valuation*, $v : W \mapsto \mathbb{R}$ is a data valuation and s is a real value of the clock S. A *region* is a subset of states of the form $\{(\ell, \gamma, v, s) \mid \varphi \text{ holds for } (\gamma, v, s)\}$, for some location ℓ and some constraint φ, written $\ell \times \varphi$.

The set Q_{init} of *initial states* is the region $\ell_{init} \times \varphi_{init}$, for some constraint φ_{init}: the automaton starts in its initial location, with some given initial constraint. (From this point on, the parameter values are not modified.)

A state $q = (\ell, \gamma, v, s)$ is *urgent* if there exists some action transition e, with source location ℓ and a guard of the form $\varphi \wedge asap$, such that φ holds for (γ, v, s): some urgent transition is enabled. From a non urgent state $q = (\ell, \gamma, v, s)$, the automaton can spend some time $\varepsilon \geq 0$ in a location ℓ, providing the invariant I_ℓ remains true. This *delay move* results in state $q' = (\ell, \gamma, v, s + \varepsilon)$ (nothing else is changed during this time). Since location invariants are convex formulas, if I_ℓ is satisfied for s and $s + \varepsilon$, then it is also satisfied for any α, $0 \leq \alpha \leq \varepsilon$.

From a state $q = (\ell, \gamma, v, s)$, the automaton can also apply some action transition $\langle \ell, \varphi, \sigma, \theta, \ell'\rangle$, providing guard φ is true for the current valuations (γ, v, s). In an instantaneous *action move*, the valuation of discrete variables is modified from v to v' according to update relation θ and the automaton switches to target location ℓ', resulting in state $q' = (\ell', \gamma, v', s)$.

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. Similarly, the notions of predecessor and set $Pre^*(Q)$ can be defined.

Synchronized product of p-automata. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two p-automata with a common universal clock S. The *synchronized product* (or parallel composition, see e.g. [12]) $\mathcal{A}_1 \times \mathcal{A}_2$ is a p-automaton with S as universal clock and the union of sets of parameters (resp. discrete variables) of $\mathcal{A}_1$ and $\mathcal{A}_2$ as sets 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. Constraints associated with locations (invariants, initial constraint) are obtained by the conjunction of the components 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). For simplicity we suppose here that synchronized transitions are non urgent.

Parametric verification. For a given automaton, $Post^*(Q_{init})$ represents the set of reachable states. For p-automata, we have the following *closure* property: if Q is a finite union of regions, also called *zone*, then the successor of Q is also a zone. Hence, the output of the computation of $Post^*(Q_{init})$ (if it terminates) is a zone. Consider now some property U, such that the set of states violating U can be characterized by a zone $Q_{\neg U}$. Proving that U holds for the system reduces to prove the emptiness of zone $Post^*(Q_{init}) \cap Q_{\neg U}$. Alternatively it suffices to prove: $Pre^*(Q_{\neg U}) \cap Q_{init} = \emptyset$. Note that we are interested here in proving that property U holds for *all* the valuations of parameters satisfying the initial constraint. The problem is known to be undecidable in general [4]: there is no guarantee of termination for the computation of $Post^*$ (or Pre^*).

3 Description and modelization of the system

Recall that algorithms $\mathcal{I}$ and $\mathcal{B}'$ use rate values carried by RM cells to dynamically compute the rate expected by the network for the conformance test of future D cells. In order to verify the correctness of $\mathcal{B}'$ with respect to $\mathcal{I}$, we introduce a snapshot action taking place at an arbitrary time `t`, which will be a parameter of the model. For our purpose of verification, it is enough to consider the snapshot as a final action of the system.

We first give p-automata as models for the environment and algorithms $\mathcal{I}$ and $\mathcal{B}'$. Then, in the complete 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 will often use `S` instead of s.

3.1 A model of environment and observation

The p-automaton $\mathcal{A}_{env}$ modeling environment (see Figure 2) involves 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 value (`R' > 0`). The *snapshot* action has `S=t` as a guard, and location *Wait* is assigned invariant `S` $\leq$ `t` in order to "force" the switch to location *EndE*.

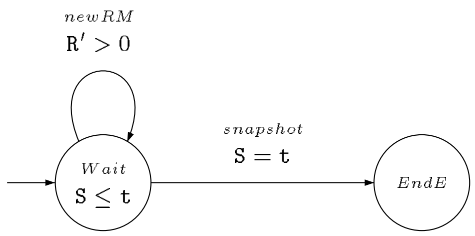

Fig. 2. Automaton $\mathcal{A}_{env}$ modeling arrivals of RM cells and snapshot

3.2 Algorithm $\mathcal{I}$

Definition of Ideal Rate. As already mentioned, transmissions are not instantaneous and parameters a and b represent respectively a lower and an upper bound of the delay. Recall that s is the current time and t the date of the snapshot. An RM cell received from the network is relevant to the computation of the "ideal rate" only if it has been received before s and (1) either it is the last received before or at time $t-b$, or (2) it arrived inside the time interval $]t-b, t-a]$. The *ideal rate* $E_t(s)$, estimated at current time s for time t, is the highest value among these RM cells. In other words, if $n \geq 0$ and $r_0, r_1, \ldots, r_n$ are the successive arrival times (before s) of RM cells, such that $r_0 \leq t-b < r_1 \leq r_2 \leq \cdots \leq r_n \leq t-a$, and if $R_0, R_1, \ldots, R_n$ are the corresponding rate values, then the expected rate is $E_t(s) = Max\{R_i, 0 \leq i \leq n\}$. The case where $n = 0$ is obtained when no new RM cell arrived between $t-b$ and $t-a$. Note that in [14], RM cell arrival times $r_1, r_2, \ldots, r_n$ are additionally assumed to form a *strictly* increasing sequence (see section 4.2).

Incremental algorithm $\mathcal{I}$. The following algorithm $\mathcal{I}$ proceeds in an incremental way, by updating a variable `E` at each reception of an RM cell, until current time s becomes equal to t. It is easy to see that, at this time, the value of `E` is equal to the ideal rate $E_t(s)$ defined above. More precisely, algorithm $\mathcal{I}$ involves variable `R` and parameter `t` (in common with $\mathcal{A}_{env}$) and, in addition:
- the two parameters `a` and `b` (representing the lower and upper bounds of the transit time from the interface to the user and back),
- the specific variable `E` (which will be equal to the ideal rate $E_t(s)$ when the value of the universal clock `S` reaches 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-b` and `t-a`:

1. If $\mathtt{S} \leq \mathtt{t-b}$ (case $n = 0$ above), `E` is updated to the new value of `R`:
   `[I1] if t >= S+b then E'= R`
2. If $\mathtt{t-b} < \mathtt{S} \leq \mathtt{t-a}$, the new ideal rate becomes `E'=`Max`(E,R)` (from the definition and the associativity of Max). To avoid using function Max, this computation is split into two subcases:

```
[I2a] if S+a <= t < S+b and E < R then E'= R
[I2b] if S+a <= t < S+b and E >= R then E'= E
```

3. If `S` $>$ `t-a`, the rate `E` is left unchanged:
```
[I3] if t < S+a then E'= E
```

Algorithm $\mathcal{I}$ terminates when the snapshot takes place (`S=t`).

Remark. A program of conformance control based on $\mathcal{I}$ would need to store at each instant s all the rate values of the RM cells received during interval $]s-b, s]$, which may be in huge number on an ATM network with large bandwidth.

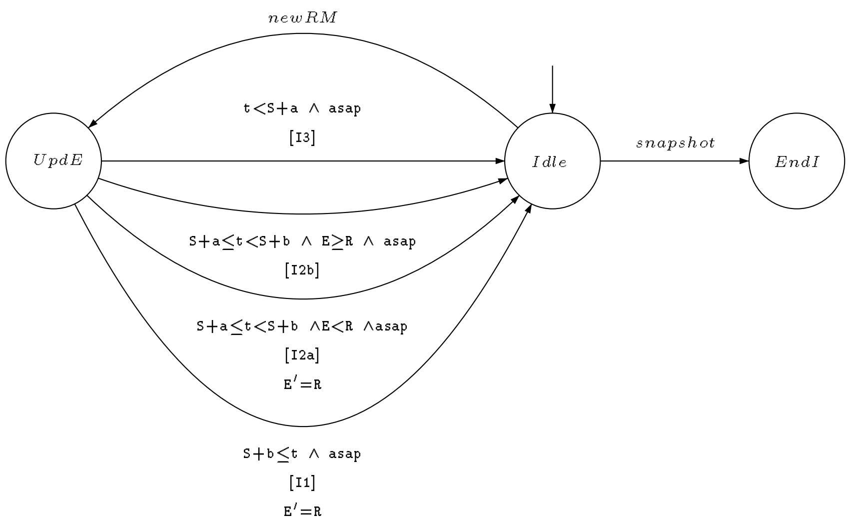

Fig. 3. Automaton $\mathcal{A}_{\mathcal{I}}$

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 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-b` and `t-a`, 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*.

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

Like $\mathcal{I}$, algorithm $\mathcal{B}'$ involves parameters `a` and `b` (but not `t`), and variable `R`. In addition, it has six specific variables:
- `tfi` and `tla`, which play the role of `fi`-rst and `la`-st deadline respectively,
- `ACR`, for "Approximate Current Rate" , which corresponds to $A(s)$,
- `FR`, for "First Rate", which is the value taken by `ACR` when current time `S` reaches `tfi`,
- `LR`, for "Last Rate", which is the value taken by `ACR` when current time `S` reaches `tla`. It stores the rate value `R` carried by the last received RM cell.
- `Emx` is just a convenient additional variable, intended to be equal to Max`(FR, LR)`.

Initially, `S=tfi=tla`, and the other variables are all equal. Algorithm $\mathcal{B}'$ reacts to two types of events: "receiving an RM cell" and "reaching `tfi`".

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+a` and `S+b` with respect to `tfi` and `tla`, and those of `R` with respect to `Emx` and `ACR`. Among the eight cases (from `[1]` to `[8]`), we omit operations `[1]` to `[5]` for lack of space, but they are similar to `[6]`:

```
[6] if S < tfi and Emx > R and R >= LR then
          LR' = R, FR' = Emx.
[7] if S >= tfi and  ACR <= R then
          LR' = R, FR' = R, Emx'= R, tfi'= S+a, tla'= S+a.
[8] if S >= tfi and  ACR > R then
          LR' = R, FR' = R, Emx' = R, tfi' = S+b, tla' = S+b.
```

Reaching `tfi`. When the current time `S` becomes equal to `tfi`, the approximate current rate `ACR` is updated to `FR` while `FR` is updated to `LR`. Moreover, `tfi` is updated to `tla`. There are two cases depending on whether `tfi` was previously equal to `tla` (operation `[9a]`) or not (operation `[9b]`). In the first case, current time `S` will go beyond `tfi` (= `tla`), while in the second case, `S` will stay beneath the updated value `tla` of `tfi`. We have:

```
[9a] if tfi = tla then
          ACR' = FR, FR' = LR, Emx' = LR.
[9b] if tfi < tla then
          ACR' = FR, tfi' = tla, FR' = LR, Emx' = LR.
```

When the events "reaching `tfi`" (`S=tfi`) and "receiving an RM cell" simultaneously occur, operation `[9a]` (case `tfi=tla`) or `[9b]` (case `tfi<tla`) 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 `[9a]` or `[9b]` must be performed before termination of $\mathcal{B}'$.

Automaton $\mathcal{A}_{\mathcal{B}'}$. Algorithm $\mathcal{B}'$ is modeled as p-automaton $\mathcal{A}_{\mathcal{B}'}$, 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.

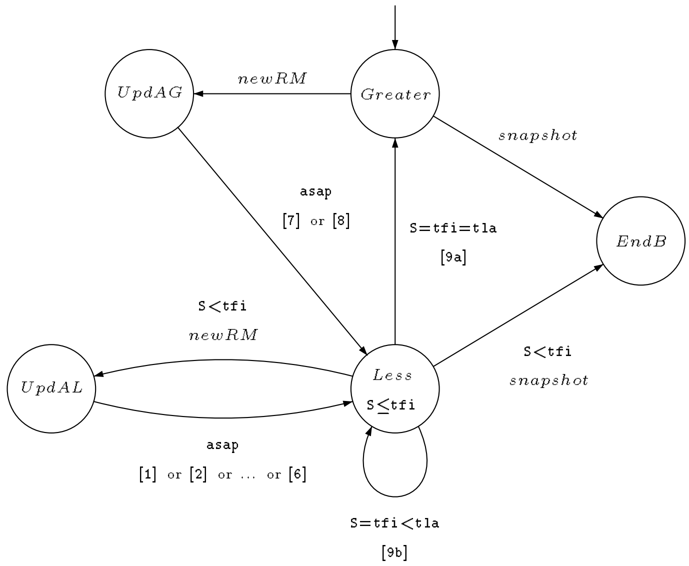

Fig. 4. Approximation automaton $\mathcal{A}_{\mathcal{B}'}$

Event "reaching tfi" (`S=tfi`) is simulated by introducing two locations *Less* and *Greater* in $\mathcal{A}_{\mathcal{B}'}$. Initially $\mathcal{A}_{\mathcal{B}'}$ is in *Greater*, with constraint: `S=tfi=tla` $\wedge$ `ACR=FR=LR=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).

3.4 Synchronized product and property U

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

$Q_{init} = (Wait, Idle, Greater) \times \varphi_{init}$,

where φ_{init} is the constraint `S=tfi=tla` $\wedge$ `R=E=ACR=FR=LR=Emx` $\wedge$ `0<a<b`.

We then have to check that $Post^*(Q_{init})$ does not contain any state where the property U is violated. Recall that property U expresses in terms of the ideal rate $E_t(s)$ computed by $\mathcal{I}$, and the approximate value $A(s)$ computed by $\mathcal{B}'$, by: For all t, when s reaches t, $A(s) \geq E_t(s)$. In our model $\mathcal{T}$, `E` corresponds to $E_t(s)$, `ACR` to $A(s)$ and snapshot (at `S=t`) makes the automaton switch to its final state, hence property U translates as:

when $\mathcal{T}$ is in location $(EndE, EndI, EndB)$, `ACR` $\geq$ `E`.

The set of states where U does not hold is therefore the region

$Q_{\neg U} = (EndE, EndI, EndB) \times$ `ACR<E`.

As explained in Section 2, we have to check $Post^*(Q_{init}) \cap Q_{\neg U} = \emptyset$ or, alternatively, $Pre^*(Q_{\neg U}) \cap Q_{init} = \emptyset$.

4 Verification of correctness

4.1 Verification with HyTech

Automata $\mathcal{A}_{env}$, $\mathcal{A}_{\mathcal{I}}$ and $\mathcal{A}_{\mathcal{B}'}$ can be directly implemented into HyTech [12], which automatically computes the synchronization product $\mathcal{T}$. 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 487 sec. on a SUN station ULTRA-1 with 64 Megabytes of RAM memory. Alternatively, the backward computation of $Pre^*(Q_{\neg U})$ requires 15 iteration steps and its intersection with Q_{init} is checked to be empty in 90 sec. The automated proof of correctness of $\mathcal{B}'$ is thus achieved. Recall that these automata (with 3 parameters a, b and t) belong to an undecidable class [4], so termination was not guaranteed a priori.

4.2 Comparison with previous work

Verification at CNET. Ideal rate algorithm $\mathcal{I}$ and correctness property U (`S=t`$\Rightarrow$ `E`$\leq$ `ACR`) have been formalized by J.-F. Monin and F. Klay at CNET. In [14], they give the first manual proof of U, using the classical method of invariants. They first split U into a conjunction of two properties:

U_1 : `tfi` $\leq$ `S` $\leq$ `t` $\Rightarrow$ `E` $\leq$ `ACR` and U_2 : `S` $\leq$ `t` $<$ `tfi` $\Rightarrow$ `E` $\leq$ `ACR`.

The proof of $U_1 \wedge U_2$ is then done in two steps. First, $U_1 \wedge U_2$ is in turn strengthened into $V \equiv U_1 \wedge U_2 \wedge U_3 \wedge \cdots \wedge U_{10}$, where $U_3, \ldots, U_{10}$ are nontrivial auxiliary properties of $\mathcal{B}'$. Second, V is proved to be an invariant (true initially and remaining true after each event). The invariance proof for V has been mechanically checked with the proof assistant COQ [13]. The auxiliary properties $U_3, \ldots, U_{10}$

can be seen as "lemmas" necessary to achieve the proof of $U_1 \wedge U_2$ by (fixpoint) induction.

With respect to our approach, property V can be seen as a fixpoint of *Post* and, as such, is an overall approximation of $Post^*$ (since $Post^*$ is the *least* fixpoint). The main advantage of our approach, is that no auxiliary property ("lemma") such as $U_3, \cdots, U_{10}$ has to be manually discovered: U is mechanically verified in its original form. Note that one of the U_i (`tfi= tla` $\Rightarrow$ `FR=LR`) is not true in our model but should be replaced by `tfi= tla` $\Rightarrow$ `FR` $\geq$ `LR`. This is a consequence of the slightly more general hypothesis $r_1 \leq r_2 \leq \cdots \leq r_n$ instead of $r_1 < r_2 < \cdots < r_n$. Another advantage here is that $Post^*$ characterizes *all* the properties of the system, and not only U. Therefore $Post^*$ can be immediately reused for proving any other property P of the system by testing that $Post^*$ does not contain any state violating P. Finally our modelization is likely to be reusable for modeling and verifying enhanced versions of $\mathcal{B}'$, which are currently under development at CNET.

Verification with GAP. In [10], we achieved a first mechanical proof of U by encoding the successor relation of the system as a logic program with arithmetical constraints, and computing a fixed-point of the program through the bottom-up evaluation procedure of Revesz [16]. The encoding required an approximation of the successor relation, so that only an upper approximation of $Post^*$ was generated. Nevertheless this approximation was sufficient to prove U, because it did not contain any state violating U.

With respect to that approach, we used here HyTech [12], a sophisticated and widely spread analysis tool for hybrid systems [2], rather than GAP, a specific prototype implementation of Revesz's procedure [11]. Therefore our results are now easily reproducible. Besides, with respect to GAP, we reach an exact fixed-point rather than an approximation, and the execution time is much (about 10 times) faster. On the other hand, termination of fixpoint computation was guaranteed with GAP by Revesz's decidability result.

5 Final Remarks

Our modelization is a direct translation without any simplification of the real algorithm $\mathcal{B}'$ described in the international norm ITU I-371. We automatically proved the basic correctness property U of algorithm $\mathcal{B}'$ using HyTech [12] (the full HyTech code is given in [6]). The proof is parametric in the sense that U holds for all values of the two parameters a and b (with $0 < a < b$) involved by $\mathcal{B}'$. A third parameter t was used for specifying property U itself. Such a proof is *a priori* impossible to do with other analysis tools of real-time systems such as UPPAAL [5] or KRONOS [8] due to this use of parameters. Our analysis contributes to improve the comprehension of the correctness proof for the ABR conformance protocol, in particular in relaxing some unnecessary assumptions. It paves the way for the verification of enhanced versions of $\mathcal{B}'$ currently under development at CNET. This case study gives additional evidence of the importance of (variants of) *parametric timed automata* [4] as a means for modeling and analysing real industrial applications. Other successful verifications of parametric concurrent systems using HyTech can be found in [7].

References

1. 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.
2. 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.
3. R. Alur and D. Dill. "Automata for Modeling Real-Time Systems". *Proc. 17th ICALP*, LNCS 443, 1990, pp. 322–335.
4. 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.
5. 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.
6. B. Bérard and L. Fribourg. "Automated verification of a parametric real-time program: the ABR conformance protocol". *Technical Report LSV-98-12*, CNRS & Ecole Normale Supérieure de Cachan, Dec. 1998 (http://www.lsv.ens-cachan.fr/Publis/).
7. B. Bérard and L. Fribourg. "Reachability Analysis of (Timed) Petri Nets Using Real Arithmetic". *Technical Report LSV-99-3*, CNRS & Ecole Normale Supérieure de Cachan, March 1999 (http://www.lsv.ens-cachan.fr/Publis/).
8. C. Daws, A. Olivero, S. Tripakis and S. Yovine. "The Tool KRONOS". *Hybrid Systems III*, LNCS 1066, 1996, pp. 208-219.
9. P. Felix et al.. "Compréhension de l'étude de cas ABR". *Internal Note*, LaBRI, University of Bordeaux, France, 1997.
10. 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/)
11. L. Fribourg and J. Richardson. "Symbolic Verification with Gap-Order Constraints". *Proc. 6th Intl. Workshop on Logic Program Synthesis and Transformation (LOPSTR)*, LNCS 1207, 1996, pp. 20–37.
12. T. Henzinger, P.-H. Ho and H. Wong-Toi. "A User Guide to HYTECH". *Proc. TACAS'95*, LNCS 1019, 1995, pp. 41–71.
13. J.F. Monin. "Proving a real time algorithm for ATM in Coq". *Types for Proofs and Programs*, LNCS 1512, 1998, pp. 277–293.
14. J.-F. Monin and F. Klay. "Formal specification and correction of I.371.1 algorithm for ABR conformance". *Internal Report NT DTL/MSV/003*, CNET, 1997.
15. 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.
16. P.Z. Revesz. "A Closed-Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints", *Theoretical Computer Science*, 1993, vol. 116, pp. 117-149.