Mechanical Verification of an Ideal Incremental *ABR* Conformance Algorithm *

Michaël Rusinowitch[1], Sorin Stratulat[2], and Francis Klay[3]

[1] LORIA-INRIA
[2] LORIA-Université Henri Poincaré
54506 VANDŒUVRE-LES-NANCY CEDEX, FRANCE
email:rusi,stratula@loria.fr
[3] France Telecom CNET DTL/MSV, Technopole Anticipa
2 av. Pierre Marzin, 22307, Lannion, France
email:Francis.Klay@cnet.francetelecom.fr

Abstract. The Available Bit Rate protocol (ABR) for ATM networks is well-adapted to data traffic by providing minimum rate guarantees and low cell loss to the ABR source end system. An ABR conformance algorithm for controlling the source rates through an interface has been defined by ATM Forum and a more efficient version of it has been designed in [13]. We present in this work the first complete mechanical verification of the equivalence between these two algorithms. The proof is involved and has been supported by the PVS theorem-prover. It has required many lemmas, case analysis and induction reasoning for the manipulation of unbounded scheduling lists. Some ABR conformance protocols have been verified in previous works. However these protocols are approximations of the one we consider here. For instance, the algorithms mechanically proved in [10] and [5] consider scheduling lists with only two elements.

Introduction

The Available Bit Rate protocol (ABR) for ATM networks is well-adapted to data traffic by providing minimum rate guarantees and low cell loss to the ABR source end system. The protocol relies on a contract between the operator who ensures a minimum rate and the source who must respect a rate that is dynamically allocated to him, according to the resources available in the networks. Due to its flexibility the ABR service admits elaborated traffic management mechanisms. To avoid congestion the operator should control in real-time that the actual rate consumed by every ABR application is consistent with the allowed rate. Several algorithms for this conformance control have been proposed and discussed in standardization committees.

* Supported by CNET CTI 96 1B 008 and Action de Recherche Coopérative INRIA PRESYSA

E.A. Emerson and A.P. Sistla (Eds.): CAV 2000, LNCS 1855, pp. 344–357, 2000.

It is essential for an operator to give evidence that the conformance control of the service he proposes does not jeopardize the quality of service (QoS) provided by the ATM network. For such a task formal validation through mathematical arguments is required. However conformance control verification often involves complex case analysis or inductions. This has motivated some operators to employ automated verification tools such as proof-assistants or model-checkers to process these proof obligations.

The algorithm Acr that has been defined by ATM Forum is considered to give the optimal conformance control, in that it computes the minimal allowed rate among the other algorithms. An algorithm (B') for computing an approximation of the optimal control has been designed by C. Rabadan from France-Telecom [12]. It is more efficient since the next two rates to be controlled are scheduled and are updated when receiving RM-cells. This incremental algorithm has been generalized in [13] to the scheduling of an arbitrary number of rates in the future. Our goal here is to derive a mechanical proof that this ideal incremental algorithm is indeed equivalent to the reference algorithm Acr. By this we mean that every step in the equivalence proof has been verified mechanically. The theorem prover we use is PVS [11]. Although PVS is interactive and operates under the direct control of the user it is capable of large autonomous deduction steps by appealing to decision procedures for arithmetics, to rewriting and induction.

Related works Some ABR conformance protocols have been automatically verified in previous works. However all these protocols are approximations of Acr. In particular unlike our case they assume a bound on the number of rates to be scheduled. For instance Algorithm B' of C. Rabadan [12] admits scheduling lists with only two elements. It has been proved recently in [10]. This proof is based on the calculus of weakest preconditions [6] (inductive invariants) and has been completely formalized with the COQ proof-assistant [2]. According to [10] the correctness proof of Algorithm B' *has been a key argument in the standardization process of ABR.* Several proof techniques have been experimented in the FORMA project (`http://www-verimag.imag.fr`) for the validation of ABR protocols. But the model checking approaches have been hindered by the numerical parameters of the algorithm. L. Fribourg has also obtained good results with extended timed automata [7]. A successful proof of protocol B' with the parameterized temporized automata of Hytech [8] has been reported in [5].

Layout of the paper We first describe the principle of ABR Conformance in Section 1. Then we introduce the algorithms Acr and Acr1 for controlling the ABR Conformance in Section 2 and 3 respectively. The rest of the paper is devoted to the equivalence proof of these two algorithms. We first introduce some key properties in Section 4, then an overview of the proof in Section 5. Since the PVS proof is too complex to be presented in extenso we give a skeleton that follows closely the mechanical proof. We comment about the mechanical proof in Section 6. For the interested reader, the full PVS specification and proof scripts can be found at `http://www.loria.fr/~stratula/abr`.

1 ABR Conformance Control

ATM (Asynchronous Transfer Mode) technology allows networks to transmit on the same media various applications whose needs are different in term of data-flow rate or quality of services. ATM is a connection-oriented technology since users should declare service requirements and traffic parameters to all intermediates switches when initializing connections. They also may agree to control these parameters on demand. In order to guarantee QoS a traffic contract specifying a traffic mode is negotiated when the connection is set up. Traffic management should ensure that users get their desired QoS although traffic demand is constantly varying. In other words traffic management should ensure that all contracts are met.

In order to solve the critical issue of *congestion control* the effective rate of cells emitted by user applications is controlled by a *conformance algorithm* called GCRA (Generic Control of Cell Rate Algorithm). Among the possible traffic modes, Constant Bit Rate (CBR) and Variable Bit Rate (VBR) were designed mainly for traffic like voice and video. The Available Bit Rate (ABR) service class is especially adapted for standard data traffic, where timing constraints are not tight. Target applications for ABR are email, WWW, file transfer and variable quality video and voice. The principle of ABR is to divide the available bandwidth[1] fairly among active traffic sources so that the network should provide each user with the best rate that is compatible with the current traffic (best-effort service principle). In ABR connections the allowed cell rate (ACR) is determined by the network from load information and may vary during the same connection. The network informs periodically the user about the new rate he can apply by sending back to him *Resource Management (RM)* cells. Hence the source rate is dynamically adjusted according to the available resources of the network by a *feedback control loop* (see Fig. 1). Since the allocated rate varies during a connection with ABR mode, the conformance control is performed by a dynamic GCRA (DGCRA).

Several ABR conformance algorithms have been proposed to the normalization committees. Algorithm Acr [4] can be viewed as a reference for defining the control of the user data-flow in the case of ABR. Each time a data cell arrives from the ABR terminal into the control interface Algorithm Acr computes the rate that should be applied to this cell. The computational cost induced by Acr has been considered too high and there were several proposals to improve it.

For instance it was noticed that the rate change is only determined by the departure of RM-cells leaving the control interface towards the ABR terminal (called *backward RM-cells*) and these RM-cells are much less frequent than data cells. Hence there is much improvement in scheduling rate changes in advance when receiving backward RM-cells in the interface. This is the motivation for

[1] left-over by CBR,VBR, e.g.

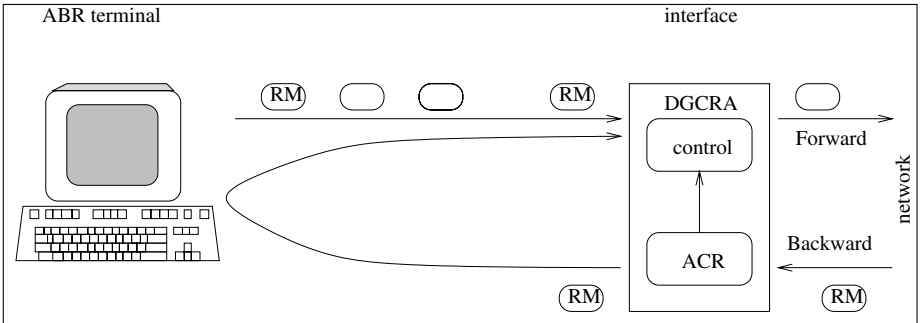

Fig. 1. Time delays between the network and an ABR terminal

the so-called incremental algorithm Acr1 which has been designed to maintain a list of planned rates. This list is updated each time the interface receives a new backward RM-cell. Also controlling the rates with the scheduling list of Acr1 seems to be less expensive than computing the maximum of a list of rates with Acr.

2 Data-Flow Control Definition with ABR (Algorithm Acr)

We shall give the data-flow control definition called here Acr. It has been first introduced in ATM Forum by [4] and since then it has been considered as a reference for the other algorithms.

The principle of the algorithm Acr running in the interface is to manage the list of backward RM-cells received from the network (Fig. 1) in order to determine the rate that has to be controlled at some instant. We shall associate with each RM-cell a couple (t, er) where t is the time when the backward RM-cell leaves the interface (where conformance control is performed) towards the ABR terminal and er is the new rate imposed by the network to the ABR terminal. For our discussion we identify a cell with its associated couple. By convention we call *time* (resp. *rate*) of c the first (resp. second) component of a cell c. The control device receives the new expected rate value before the ABR terminal. Hence due to the transmission delays in the networks, the control device should apply at time t a value received at time $t - \tau$ where τ represents a propagation delay equal to the time taken by an RM-cell to go from the interface to the ABR terminal and back to the interface. However the propagation times in the network may vary according to the traffic load. In order to take into account variations of these delays, the ITU-T has proposed that the rate to be controlled at time t is computed as the maximum among the rates received by the interface within a temporal window limited by $t - \tau_2$ and $t - \tau_3$ and the rate received just before or at $t - \tau_2$. The *window parameters* τ_2, τ_3 satisfy $\tau_2 > \tau_3$. They have been negotiated during the establishment of the traffic contract.

More formally, let $l = [(t_1, er_1), (t_2, er_2), \ldots, (t_n, er_n)]$ be a list of RM-cells. The *first cell* of l is (t_1, er_1). The list l is *time-decreasing* (*t.d.* in short) if $i < j \Rightarrow t_i \geq t_j$, for all $i \in [1..n-1]$ and $j \in [2..n]$. In order to handle limit cases in the definitions below we shall define $t_0 = +\infty$. We denote by $\cdot$ (resp. @) the *cons* (resp. *append*) operator on lists.

The rate to be controlled at time t w.r.t. the t.d. list $l = [(t_1, er_1), (t_2, er_2), \ldots, (t_n, er_n)]$ of backward RM-cells leaving the interface is:

$$Acr(l,t) = \begin{cases} MaxEr(Wind(l,t)) & \text{if } Wind(l,t) \neq \emptyset, \\ 0 & \text{otherwise} \end{cases}$$

where

$$Wind(l,t) = \{(t_i, er_i) \in l |\ (t - \tau_2 < t_i \leq t - \tau_3) \text{ or } (t_i \leq t - \tau_2 < t_{i-1})\}$$
$$MaxEr(s) = max\{er | (t, er) \in s\}$$

For instance, with this rate control policy the end user can benefit at time $t + \tau_3$ from a rate increase received on a backward cell at time t by the interface, that is, as soon as possible. On the other hand a rate decrease will be taken into account only at time $t + \tau_2$, that is, as late as possible. Hence this is a policy in favour of the user and based on worst-case situations. It can be noticed that for a fixed list l, $Acr(l,t)$ is decreasing on t after $t + \tau_3$ since the window is (non strictly) decreasing for the inclusion relation.

3 Incremental Conformance Checking (Algorithm Acr1)

We now introduce an ideal incremental algorithm Acr1 for conformance control. Unlike Acr the algorithm Acr1 computes a list of rates to be controlled in the future. Hence it maintains a list $Prog(l)$ of cells (l is as above) (t_j, er_j) containing a future rate er_j to be controlled together with the time t_j it comes into effect. Similar to l, $Prog(l)$ will be sorted in decreasing order on time. This list is updated when receiving a backward RM-cell. An important gain over Acr is due to the fact that the RM-cells are much less frequent than the data cells. The ratio of RM-cells to all cells recommended by the ATM Forum[2] is 1/32. Let us assume that the list $Prog(l)$ is constructed. We shall prove later that it is time-decreasing as l. Then the rate to be controlled at time t is:

$$Acr1(l,t) = Prog(l)_t$$

where p_t is a function that computes by extrapolation the rate at time t from a list of scheduled rates p. This function simply extracts from p the first rate value that is scheduled at or immediately before the time t. This rate is the one to be controlled at t. Formally:

[2] ATM Forum Traffic Management Specification Version 4.0. ftp://ftp.atmforum.com/pub/approved-specs/af-tm-0056.000.ps

$$p_t = \begin{cases} er_i & \text{if there exists a cell } (t_i, er_i) \in p \text{ such that } t \geq t_i \text{ and} \\ & \quad \text{there is no cell } (t_j, er_j) \in p \text{ with } t \geq t_j,\ i > j \text{ and } i, j \in [1..n] \\ 0 & \text{otherwise} \end{cases}$$

We now describe how the list $Prog(l)$ is updated. A list l' is a *prefix* of a list l if there exists a list l'' such that $l = l'@l''$. Given a list l' and a time t we denote by $l'_{>_\tau t}$ the maximal prefix of l' containing cells with time greater than t. Similarly we define $l'_{\leq_\rho er}$ as the maximal prefix of l' containing cells with rate less or equal than er. This gives the following recursive definition of $Prog(l)$:

$$Prog((t, er) \cdot l) = \begin{cases} \textbf{if} \quad er \geq Prog(l)_{t+\tau_3} \textbf{ then } (t + \tau_3, er) \cdot l' \\ \qquad \text{where } Prog(l) = Prog(l)_{>_\tau t+\tau_3} @ l'. \\ \textbf{else if } Prog(l)_{\leq_\rho er} \text{ is empty } \textbf{then } (t + \tau_2, er) \cdot Prog(l) \\ \qquad \textbf{else } (t', er) \cdot l'' \\ \qquad \text{where } Prog(l) = Prog(l)_{\leq_\rho er} @ l'' \\ \qquad \text{and } Prog(l)_{\leq_\rho er} = L@[(t', er')] \end{cases}$$

$Prog(l)$ is by definition the empty list when l is the empty list.

Our goal in the remaining of the paper is to show that the incremental algorithm Acr1 delivers the same rate values as the reference algorithm Acr, i.e.

$$\forall t\ \forall l \quad Acr1(l, t) = Acr(l, t)$$

4 Two Key Properties

Two properties were used abundantly in the main proof. The first one `time_dec` states that $Prog(l)$ is sorted in decreasing order on its time components. The second one `rate_inc` specifies a prefix of $Prog(l)$ that is sorted in increasing order on its rate component. Both assume that l is time-decreasing (t.d.). We denote by $Timel(l)$ the time of its first cell (0 if l is empty).

Property 1 (time_dec) *Given a t.d. cell list l the list $Prog(l)$ is also t.d..*

Proof. By induction on l. When l is empty $Prog(l)$ is empty hence t.d.. Assume by induction hypothesis that for any t.d. list l_1 of length less or equal than the length of l, $Prog(l_1)$ is t.d.. Let us prove that $Prog((t, er) \cdot l)$ is t.d. when $t \geq Timel(l)$. We perform a case analysis guided by the definition of $Prog$ from Section 3. Note that any sublist of a t.d. list is also t.d..

1. assume that $er \geq Prog(l)_{t+\tau_3}$. Let l' be such that $Prog(l) = Prog(l)_{>_\tau t+\tau_3} @ l'$. Then $Prog((t, er) \cdot l)$ is equal to $(t + \tau_3, er) \cdot l'$ which is t.d. since $t + \tau_3 \geq Timel(l')$ and l' is t.d..

2. otherwise, let $Prog(l) = Prog(l)_{\leq_\rho er}@l''$. If $Prog(l)_{\leq_\rho er}$ is empty then the list $(t+\tau_2, er)\cdot Prog(l)$ is t.d. because $t+\tau_2 \geq Timel(Prog(l))$. This can be deduced from (i) the time-decreasing property of $(t,er)\cdot l$, and (ii) the fact that for any nonempty t.d. list $L = (\bar{t},\bar{e})\cdot L'$ we have $\bar{t}+\tau_2 \geq Timel(Prog(L))$. (This can be proved by a simple induction on L.)

If $Prog(l)_{\leq_\rho er}$ is a nonempty list of the form $L''@[(t',er')]$ then $(t',er)\cdot l''$ is a t.d. list as it can be deduced from the time-decreasing property of $Prog(l)$. □

We denote by $l_{\frown t}$ the maximal prefix l' of l such that the time of any cell from l' except possibly the last one is greater than t. We remark that for any nonempty t.d. list l, the prefix $l_{\frown t}$ and l have the same first cell. We say that a cell list $l = [(t_1,er_1),(t_2,er_2),\ldots,(t_n,er_n)]$ is *strictly rate-increasing* (*s.r.i.* in short) if the rate-values of its cells are strictly increasing: $i<j \Rightarrow er_i < er_j$, for all $i,j \in [1..n]$. We can observe that given an s.r.i. list l and two time values $t > t'$ we have $l_t \leq l_{t'}$.

Property 2 (rate_inc) *Given a t.d. cell list l the prefix $Prog(l)_{\frown Timel(l)+\tau_3}$ of $Prog(l)$ is s.r.i..*

Proof. By induction on l. If l is empty then $Prog(l)$ is empty hence s.r.i.. Otherwise by induction hypothesis we assume that $Prog(l)_{\frown Timel(l)+\tau_3}$ is s.r.i.. If $t \geq Timel(l)$ we will prove that $Prog((t,er)\cdot l)_{\frown t+\tau_3}$ is also s.r.i. by case analysis according to the definition of $Prog$.

1. if $er \geq Prog(l)_{t+\tau_3}$ then $Prog((t,er)\cdot l) = (t+\tau_3,er)\cdot l'$ where l' is such that $Prog(l) = Prog(l)_{>_\tau t+\tau_3}@l'$. It results that $Prog((t,er)\cdot l)_{\frown t+\tau_3} = [(t+\tau_3,er)]$ which is s.r.i..

2. otherwise let l'' be such that $Prog(l) = Prog(l)_{\leq_\rho er}@l''$:

2.1. if $Prog(l)_{\leq_\rho er}$ is empty then $Prog((t,er)\cdot l) = (t+\tau_2,er)\cdot Prog(l)$. We have $Prog((t,er)\cdot l)_{\frown t+\tau_3} = (t+\tau_2,er)\cdot Prog(l)_{\frown t+\tau_3}$. Moreover, $Prog(l)_{\frown t+\tau_3}$ is a prefix of $Prog(l)_{\frown Timel(l)+\tau_3}$ because $t \geq Timel(l)$. By consequence $(t+\tau_2,er)\cdot Prog(l)_{\frown t+\tau_3}$ is s.r.i..

2.2. otherwise $Prog(l)_{\leq_\rho er} = L@[(t',er')]$ for some L and $Prog((t,er)\cdot l) = (t',er)\cdot l''$. If l'' is empty, $Prog((t,er)\cdot l)_{\frown t+\tau_3}$ consists of the unique cell (t',er') and is obviously s.r.i. Since l'' is a sublist of $Prog(l)$ the list $l''_{\frown t+\tau_3}$ is a sublist of $Prog(l)_{\frown t+\tau_3}$. Therefore it is s.r.i.. Moreover er is less than the rate of the first cell of l'' (or $l''_{\frown t+\tau_3}$) when l'' is not empty. We conclude that $Prog((t,er)\cdot l)_{\frown t+\tau_3} = (t',er)\cdot l''_{\frown t+\tau_3}$ is s.r.i.. □

5 The Main Proof

We will employ the following notations. Assume that x is a cell, $Time(x)$ its time, $Er(x)$ its rate and τ_2, τ_3 the two window parameters. The model checking approaches with MEC [1] and UPPAAL [3] had to assign values to these parameters in order to proceed. Here we only assume all over the proof that $\tau_2 > \tau_3$

without any further mention of this hypothesis. We introduce $T_2(x)$ to denote $Time(x) + \tau_2$ and $T_3(x)$ to denote $Time(x) + \tau_3$. The predicate $\mathcal{S}^{\geq\tau}(l)$, resp. $\mathcal{S}^{<\rho}(l)$, is true if l is t.d., resp. s.r.i.. We will omit the quantifier prefixes of the formulas since all the variables are considered as universally quantified.

The proof of the conjecture $Acr1(l,t) = Acr(l,t)$ is immediate if we prove the *main lemma* $P(l)$, where:

$$P(l): \quad \mathcal{S}^{\geq\tau}(l) \Rightarrow Prog(l)_t = MaxEr(Wind(l,t))$$

We apply an induction on the length of l. When l is empty, the proof is immediate, by expanding the definitions of $Prog$, $MaxEr$ and $Wind$. In the step case, we suppose $P(l)$ and we try to prove $P(a' \cdot l)$. More precisely, we should prove that $Prog(a' \cdot l)_t = MaxEr(Wind(a' \cdot l, t))$ follows from $\mathcal{S}^{\geq\tau}(a' \cdot l)$. From the hypothesis $\mathcal{S}^{\geq\tau}(a' \cdot l)$ we deduce $\mathcal{S}^{\geq\tau}(l)$ since l is a sublist of $(a' \cdot l)$. Hence, we can assume that $Prog(l)_t = MaxEr(Wind(l,t))$. The arguments for proving the step case are also based on the following property depending on the induction hypothesis:

Property 3 The list $Prog(l)_{\frown T_3(a')}$ is s.r.i..

Proof. From Property 2 we have $\mathcal{S}^{\geq\tau}(l) \Rightarrow \mathcal{S}^{<\rho}(Prog(l)_{\frown(Timel(l)+\tau_3)})$. Together with the hypothesis $\mathcal{S}^{\geq\tau}(l)$, we deduce $\mathcal{S}^{<\rho}(Prog(l)_{\frown(Timel(l)+\tau_3)})$. We also have $Time(a') \geq Timel(l)$ since
$\mathcal{S}^{\geq\tau}(a' \cdot l)$ and by consequence $Prog(l)_{\frown T_3(a')}$ is a prefix of $Prog(l)_{\frown(Timel(l)+\tau_3)}$. Hence $Prog(l)_{\frown T_3(a')}$ is s.r.i.. □

By T_m we denote $Prog(l)_t$. Then $t \geq T_m$ and no cell from the list $Prog(l)$ has its time in the interval $(T_m, t]$. Let $I_{a'}$ be the time of the first cell of $Prog(a \cdot l)$. We have the following facts about $I_{a'}$.

1. $I_{a'} \in \{T_2(t'), T_3(t') \mid t'$ is the time of an $(a' \cdot l)$ cell$\}$, as can be easily proved by induction on l.

2. $I_{a'} \in [T_3(a'), T_2(a')]$. Since $a' \cdot l$ is t.d., from the first fact it results that $I_{a'} \leq T_2(a')$. Moreover from the definition of $Prog$: (i) if $Er(a') < Acr1(l, T_3(a'))$ then the prefix $Prog(l)_{\leq_\rho Er(a')}$ contains only cells with rates $\leq Er(a')$ which therefore occur at a time $> T_3(a')$. If $Prog(l)_{\leq_\rho Er(a')}$ is empty, then $I_{a'} = T_2(a')$, otherwise $I_{a'}$ is the time of the last cell of it. By consequence, $I_{a'} > T_3(a')$. (ii) if $Er(a') \geq Acr1(l, T_3(a'))$ then $I_{a'} = T_3(a')$.

The domain where $I_{a'}$ can take its value for each case is included in the hachured time interval of the corresponding figure. From the definition of $Prog$, it can be shown that the rate of the first cell of $Prog(a' \cdot l)$ is $Er(a')$. Hence $Prog(a' \cdot l) = (I_{a'}, Er(a')) \cdot L''$ for some list L''.

We perform a case analysis according to the position of t w.r.t. the values $T_2(a')$ and $T_3(a')$:

1. If $T_2(a') \leq t$ (see Fig. 2) then $Acr1(a' \cdot l, t) = Acr(a' \cdot l, t) (= Er(a'))$ because
   - $MaxEr(Wind(a' \cdot l, t)) = Er(a')$ since $Wind(a' \cdot l, t) = [a']$.
   - $Prog(a' \cdot l)_t = Er(a')$ because $(I_{a'}, Er(a'))$ is the first cell of $Prog(a' \cdot l)$ and $I_{a'} \leq T_2(a') \leq t$.

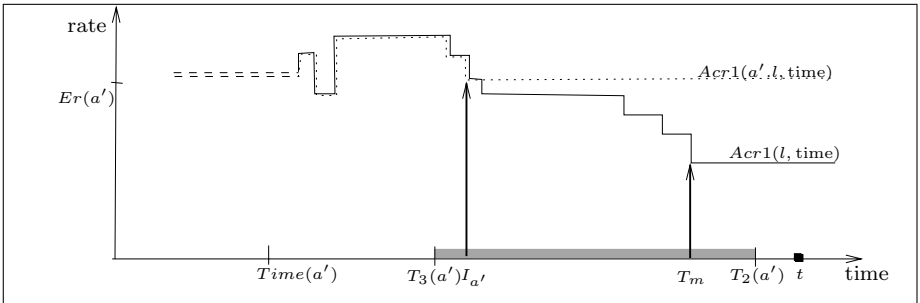

Fig. 2. $T_2(a') \leq t$

2. If $T_3(a') > t$ (see Fig. 3) then
   - a' is not member of $Wind(a' \cdot l, t)$. Therefore, $Acr(a' \cdot l, t) = Acr(l, t)$, and
   - $Acr1(a' \cdot l, t) = Acr1(l, t)$ because $I_{a'} \in [T_3(a'), T_2(a')]$.

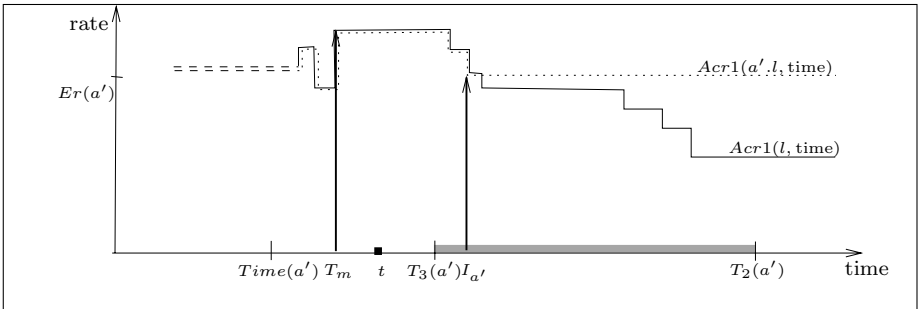

Fig. 3. $T_3(a') > t$

By the induction hypothesis $Acr1(l, t) = Acr(l, t)$. It results that $Acr1(a' \cdot l, t) = Acr(a' \cdot l, t)$.

3. If $T_2(a') > t \geq T_3(a')$ we deduce that $a' \in Wind(a' \cdot l, t)$, by definition of `Wind`.

3.1. If $Wind(l,t)$ is empty then the list l is also empty. Otherwise, the first cell of l belongs to $Wind(l,t)$. Then $Prog([a']) = [(Er(a'), T_3(a'))]$ since $t \geq T_3(a')$ and $Wind([a'],t) = [a']$. Therefore $Acr1([a'],t) = Acr([a'],t) = Er(a')$.

3.2. If $Wind(l,t)$ is nonempty, let ER_m be the value of the maximal rate of the $Wind(l,t)$ cells. According to the induction hypothesis $Acr(l,t) = Acr1(l,t)$. Thus the rate of the $Prog(l)$ cell situated at T_m equals ER_m.

In order to schedule $Er(a')$, we perform a case analysis according to the definition of $Prog$:

3.2.1. If the condition

$$Acr1(l, T_3(a')) \leq Er(a') \qquad (Cond.1)$$

is true (see Fig. 4), then $I_{a'} = T_3(a')$. Moreover $(I_{a'}, Er(a'))$ is the first cell of $Prog(a' \cdot l)$. Then $Acr1(a' \cdot l, t) = Er(a')$ because $T_3(a') \leq t$.

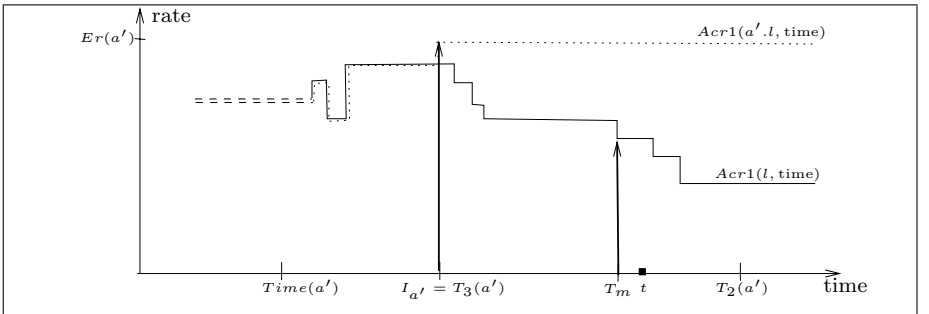

Fig. 4. $T_2(a') > t \geq T_3(a')$ and $Acr1(l, T_3(a')) \leq Er(a')$

It remains to prove that $Acr(a' \cdot l, t)$ also equals $Er(a')$ or equivalently $ER_m \leq Er(a')$.

Again from $T_3(a') \leq t$ together with the fact that the sublist $Prog(l)_{\frown T_3(a')}$ is s.r.i. according to *Property 3* we deduce $Acr1(l,t) \leq Acr1(l, T_3(a'))$. Therefore by the condition $Cond.1$ and the induction hypothesis we have $ER_m \leq Er(a')$.

3.2.2. If the condition

$$Acr1(l, T_3(a')) > Er(a') \qquad (Cond.2)$$

is true we know that $I_{a'} > T_3(a')$. We distinguish the following cases:

3.2.2.1 $T_m \leq T_3(a')$ (see Fig. 5). Since $T_3(a') \leq t$ it results that $t \geq T_3(a') \geq T_m$. By the definition of T_m, no cell of $Prog(l)$ has its time in the interval $(T_m, t]$. Hence $Acr1(l, T_3(a')) = ER_m$. On the one hand by the condition $Cond.2$ we have $ER_m > Er(a')$. Consequently $Acr(a' \cdot l, t) = ER_m$. On the other hand the instant $I_{a'}$ can be either $T_2(a')$ or the time of a $Prog(l)$ cell. $T_2(a')$ also cannot be

inside the interval $(T_m, t]$ since $T_2(a') > t$. If $I_{a'} = T_2(a')$ then $I_{a'} > t$. Assume now that $I_{a'}$ is the time of a $Prog(l)$ cell. Since $I_{a'} > T_3(a')$, $t \geq T_3(a') \geq T_m$ and there is no $Prog(l)$ cell located in the interval $(T_m, t]$ we deduce that $I_{a'} > t$. Therefore $Acr1(a' \cdot l, t)$ is also equal to ER_m.

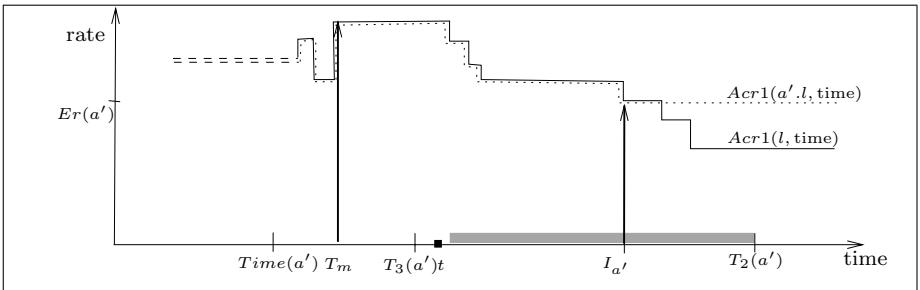

Fig. 5. $T_2(a') > t \geq T_3(a')$, $Acr1(l, T_3(a')) > Er(a')$ and $T_m \leq T_3(a')$

3.2.2.2. $T_m > T_3(a')$ (see Fig. 6). It results that $t \geq T_m > T_3(a')$. We have the following cases to consider:

– if $Er(a') < ER_m$ then $Acr(a' \cdot l) = ER_m$. From *Property 3* we deduce that $I_{a'} \geq T_m$. $I_{a'}$ can be either $T_2(a')$ or the time of a $Prog(l)$ cell which is greater than t since there is no $Prog(l)$ cell in the interval $(T_m, t]$. Hence $I_{a'} > t$ and we can conclude that $Acr1(a' \cdot l, t) = Acr1(l, t) = ER_m$.

– if $Er(a') \geq ER_m$ then $Acr(a' \cdot l) = Er(a')$. Again from *Property 3* we obtain $I_{a'} \leq T_m$. Let $Prog(l) = Prog(l)_{\leq_\rho Er(a')} @ l''$. Then $(T_m, ER_m) \in Prog(l)_{\leq_\rho Er(a')}$. It results that $Acr1(a' \cdot l, t) = Acr1(a' \cdot l, T_m) = (I_{a'}, Er(a')) \cdot l''_t = Er(a')$.

6 Comments on the Mechanical Proof

The equivalence theorem has been successfully developed within PVS [11] environment. PVS provides an expressive specification language that builds on classical typed higher-order logic with mechanisms for defining abstract datatypes. Hence the specification of the algorithms already given in functional style was relatively easy. In this work we deliberately restricted ourselves to first-order features of the language since our final goal is to prove the equivalence theorem with a first-order prover in a more automatic way, i.e. with less input from the user.

The first difficult proof step we encountered was to show that if a list of cells l is time-decreasing then its associated scheduling list $Prog(l)$ is also time-decreasing: this property is expressed by Property 1. We have tried to apply

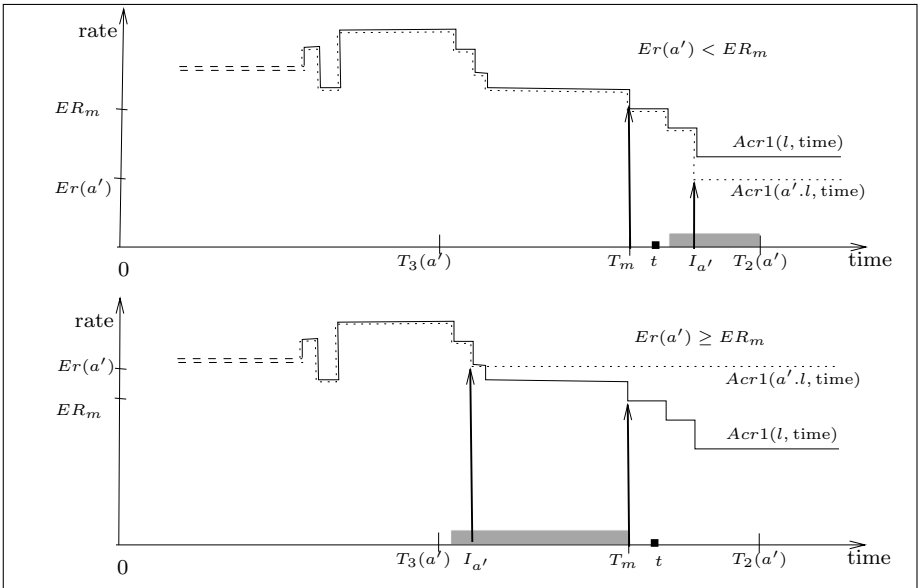

Fig. 6. Cases for $T_2(a') > t \geq T_3(a')$, $Acr1(l, T_3(a')) > Er(a')$ and $T_m > T_3(a')$

induction on the length of the list but it failed because the scheduling list in the induction conclusion is not a sublist of the one in the induction hypothesis. Therefore, the direct application of the induction hypothesis was impossible. After a closer analysis, we discovered that auxiliary lemmas concerning time properties of cell lists were needed. However the initial functions were not sufficient to express them. Hence an important decision was to enrich the initial specification with auxiliary functions such as `Timel`.

The formal verification of `time_dec` follows closely the semi-formal proof from Section 5. By the rules ASSERT and GRIND, the theorem prover PVS applies its decision procedures to perform arithmetic reasoning, employs congruence closure for equality reasoning, installs theories and rewrite rules along with all the definitions relevant to the goal in order to prove the trivial cases, to simplify complex expressions and definitions, and to perform matching. In detail, the formal proof of `time_dec` requires 32 user-steps in PVS, 6 of which are ASSERT, 5 are GRIND and 4 are LEMMA, which introduce instances of previously proved lemmas as new formulas.

The proof of `main_conj`, which codes the main lemma, is the most complex that was elaborated for this specification. It follows the skeleton of the semi-formal proof described in Section 5. We have applied an INDUCT rule to perform induction on the length of the cell list. The basic case was completed by a GRIND operation. However, the proof of the step case has needed 27 invocations of lemmas and for 6 times the application of the CASE rule, to perform case reasoning. The analysis of each particular case was a source for the development

of new lemmas that in turn may require new lemmas for their proofs. The depth of the lemmas dependency graph is 7.

The analysis of some particular cases presented in the proof of the `main_conj` lemma suggested other auxiliary functions to express additional properties. For instance, the auxiliary functions `ListUpTo(l,t)` and `SortedE(l)` denoted respectively $l_{\frown t}$ and $\mathcal{S}^{<_\rho}(l)$ that have been introduced in Section 4 to formalize Property 2. Some of the cases follow very closely the corresponding cases from the informal proof, as 1, 2, 3.1 and 3.2.1. The cases 3.2.2.1 and 3.2.2.2 are more complex and have required 17 lemma invocations. The proof of `main_conj` consists of 120 user-guided steps, seven of which are GRIND operations and 29 are ASSERT commands and indicate that the arithmetic and equality reasoning have been intensively used.

The whole proof takes 78.97s on a PC featured with an Intel Pentium II processor at 333 MHz and 128 Mbytes of RAM memory. The effort to design the mechanical proof took about two months including the time to get familiar with PVS.

7 Conclusions and Perspectives

Our objective was to derive the first mechanical proof of an ideal incremental ABR conformance algorithm. A proof by hand of the algorithm has been designed before [13]. However this kind of manual proofs is not fully convincing in general since limit cases or apparently trivial arguments are often omitted and they may be source of mistakes. In this respect our machine proof gives more evidence of the correctness of the algorithm since every single step has been verified by PVS.

On the one hand, the specification of the algorithm as a recursive function in PVS was relatively easy. The proof we obtained on the other hand was rather involved. It has required about 80 intermediate lemmas and the introduction of auxiliary definitions. We feel that there is space for optimization and many inference steps can possibly be simplified. It is also our plan to search for a proof with more automated tools. We expect to derive entirely automatic proofs for many lemmas. In another direction we also think about using higher-order specification and reasoning techniques to derive a proof that is more synthetic and therefore easier to grasp. Finally we should prove that by limiting the size of the scheduling lists the ideal ABR conformance algorithm indeed reduces to approximate algorithms such as B'.

Acknowledgements: We thank Laurent Fribourg, Bernhard Gramlich and Jean-François Monin for interesting discussions and remarks about this work.

References

1. A. Arnold. MEC: A system for constructing and analysing transition systems. In J. Sifakis, editor, *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, number 407 in LNCS, pages 117–132. Springer Verlag, 1990.
2. B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliatre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin, A. Saïbi, and B. Werner. The Coq Proof Assistant Reference Manual – Version V6.1. Technical Report 0203, INRIA, August 1997.
3. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL: a tool suite for the automatic verification of real-time systems. In R. Alur, T. A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, number 1066 in LNCS, pages 232–243, 1996.
4. A. Berger, F. Bonomi, and K. Fendick. Proposed TM baseline text on an ABR conformance definition. Technical Report 95-0212R1, ATM Forum Traffic Management Group, 1995.
5. 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)*, number 1633 in LNCS, pages 96–107, Trento, Italy, July 1999.
6. E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, NJ, 1976.
7. L. Fribourg. A closed-form evaluation for extended timed automata. Technical Report LSV-98-2, Lab. Specification and Verification, ENS de Cachan, Cachan, March, France 1998. 17 pages.
8. T.A. Henzinger, P.H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. In *CAV'97*, number 1254 in LNCS, pages 460–463. Springer-Verlag, 1997.
9. R. Jain. Congestion control and traffic management in ATM networks: Recent advances and a survey. *Computer Networks and ISDN Systems*, 28:1723–1738, 1996. ftp://ftp.netlab.ohio-state.edu/pub/jain/papers/cnis/index.html.
10. J.F. Monin and F. Klay. Correctness proof of the standardized algorithm for ABR conformance. In J. Wing, J. Woodcock, and J. Davies, editors, *Formal Methods (FM) '99*, number 1709 in LNCS, pages 662–681. Springer Verlag, 1999.
11. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer Verlag, 1992.
12. C. Rabadan. L'ABR et sa conformité. Technical report, NT DAC/ARP/034, CNET, 1997.
13. C. Rabadan and F. Klay. Un nouvel algorithme de contrôle de conformité pour la capacité de transfert "Available Bit Rate". Technical Report NT/CNET/5476, CNET, 1997.