Proving a real time algorithm for ATM in Coq

Jean-François Monin

France Télécom - CNET, DTL/MSV
2, av. P. Marzin
22307 Lannion Cedex
monin@lannion.cnet.fr

Abstract. This paper presents the techniques used for proving, in the framework of type theory, the correctness of an algorithm recently standardized at ITU-T that handles time explicitly. The structure of the proof and its formalization in Coq are described, as well as the main tools which have been developed: an abstract model of "real-time" that makes no assumption on the nature of time and a way to actually find proofs employing transitivity, using only logical definitions and an existing tactic.

1 Context and Motivation

1.1 Conformance Control in ATM

In an ATM (Asynchronous Transfer Mode) network, data cells sent by a user must not exceed a rate depending on the state of the network. Several modes for using an ATM network, called "ATM transfer capabilities" (ATC) have been defined. Each ATC may be seen as a generic contract between the user and the network, saying that the network must guarantee a number of characteristics of the connection (transfer delay and so on) provided the user sends only compliant data cells—their rate must be bounded by a value defined by the current contract. Actually, in the most interesting, recent and complicated ATC, this allowed cell rate (ACR) may vary during the same session, depending on the current state of the network. Such ATC are designed for irregular sources, that need high cell rates from time to time, but that may wait 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 contract. The key is a public algorithm for checking conformance of cells. Each ATC comes with its own conformance control algorithm.

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.

1.2 The Case of ABR

In the case of the ATC called ABR (available bit rate), a simple but very inefficient algorithm had been proposed in a first stage, and reasonably efficient

algorithms proposed later turned out to be fairly complicated. This situation has been settled when one of them has been proved correct in relation to the simple one [9]: this algorithm is now part of the I.373.1 standard. The corresponding proof was hand written, lengthy (15 pages) and somewhat tricky in places, hence we decided to formalize it in type theory in order to get a proof automatically checked by Coq.

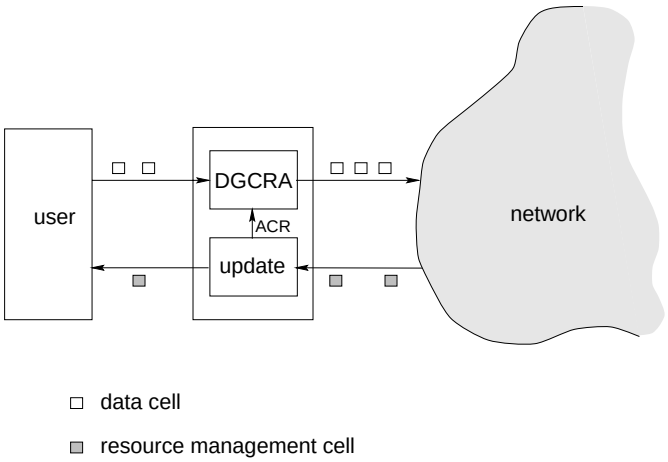

Fig. 1. conformance control

The conformance control algorithm for ABR has two parts (see fig. 1). The first one is called DGCRA (dynamic generic control of cell rate algorithm). 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. Note that, in the case of ABR, Acr depends on time: its value has to be known each time a new data cell comes from the user. This part is quite simple and has no interest here. The hard side, called "update" in fig. 1, is the computation of $\mathsf{Acr}(t)$, which depends on the sequence of values (ER_n) carried by resource management cells coming from the network. For the sake of simplicity, the cell carrying ER_n will be called itself ER_n.

Of course, $\mathsf{Acr}(t)$ depends only on cells ER_n whose arrival time t_n is such that $t_n < t$ (we order resource management cells so that $t_n < t_{n+1}$ for any n). In fact, $\mathsf{Acr}(t)$ depends only on cells ER_i such that either $t - \tau_2 < t_i \leq t - \tau_3$, or $t_i \leq t - \tau_2 < t_{i+1}$, where τ_2 and τ_3 are fixed parameters such that $\tau_3 < \tau_2$: $\mathsf{Acr}(t)$ is just the maximum of these values[1].

[1] In the protocol ABR, a resource management cell carries a value of Acr, that should be reached as soon as possible. However, because of electric propagation time, the user is aware of this expected value only after a while. Everything included, the

1.3 Effective Computation of Acr

A direct computation of $\mathsf{Acr}(t)$ would be very inefficient. However, it is not difficult to see that $\mathsf{Acr}(t)$ is constant on any interval that contains no value among $\{t_n + \tau \mid \tau = \tau_2 \vee \tau = \tau_3\}$. In other words, $\mathsf{Acr}(t)$ is determined by a sequence of values. It then becomes possible to use a scheduler handling future changes of $\mathsf{Acr}(t)$. This scheduler is updated when a new cell ER_n is received. Roughly, if s is the current time, ER_n will be taken into account at time $s + \tau_3$, while ER_{n-1} will not be taken into account after $s + \tau_2$.

The control conformance algorithm considered here exploits this idea, with the further constraint that only a small amount of memory is allocated to the scheduler. This means that information is lost. *We then just expect that the actual value of $\mathsf{Acr}(t)$ is greater or equal to its theoretical value, as defined above.*

1.4 On the Use of Coq in this Application

Coq follows the LCF approach. The user states definitions and theorems, then he (she) proves the latters by the means of *scripts* made of *tactics* and *tacticals*. Tactics are either primitive (e.g. Intro or Apply), or higher level, like Auto. Tacticals allow you to build complex tactics from simpler ones, for instance Try... Then.... However scripts *are not* proofs. Scripts *produce* actual proofs, which are data (terms) to be checked by the kernel of the proof assistant. The kernel (i.e. the type checker in the case of Coq) is only a small part of the whole code and is programmed with special care. Hence, even if scripts are difficult to read or if subtle tactics involving big programs are used (e.g. EAuto, see below), the user can be very confident in theorems proved in such a tool. This is particularly important in the application discussed here, because the manual proof seemed somewhat suspect.

Thanks to the LCF application, the user can safely program himself ad-hoc tactics. However, in this case study, existing general tactics of Coq turned out to fit our needs. In particular, we will show how EAuto (see 2.2) can be used in an efficient way. Let us explain here the difference between the tactics Auto and EAuto.

Roughly, Auto is able to prove subgoals using a sequence of introductions and of applications of already proved lemmas. For instance, Auto easily finds a proof of mortal(Socrate) from

$(\forall x : \mathsf{being})\ \mathsf{human}(x) \Rightarrow \mathsf{mortal}(x)$,
$\mathsf{human}(\mathsf{Socrate})$.

However the same goal cannot be discharged by Auto from

$(\forall x : \mathsf{being})\ (\forall y : \mathsf{beverage})\ \mathsf{human}(x) \Rightarrow \mathsf{drinks}(x, y) \Rightarrow \mathsf{mortal}(x)$,
$\mathsf{human}(\mathsf{Socrate})$,
$\mathsf{drinks}(\mathsf{Socrate}, \mathsf{hemlock})$,

---

reaction delay of the user is bounded by τ_3 and τ_2. The last resource management cell before $t - \tau_2$ is also needed: consider for instance the case where the set $\{i \mid t - \tau_2 < t_i \leq t - \tau_3\}$ is empty.

because Auto is not able to guess a witness like hemlock. Finding witnesses involves more expensive proof search. On the other hand, the recently implemented tactic EAuto makes this possible thanks to a Prolog-like strategy.

1.5 Structure of this Paper

It should be clear from the informal definition of the control conformance algorithm given in 1.2 and 1.3, that the ability to reason about real time is essential here. The point is that the algorithm itself handles time by the means of a scheduler, which is a data structure containing dates. Comparisons between dates, additions of dates and delays play a central rôle in the algorithm as well as in proofs.

This paper presents the overall structure of correctness proof of the standardized algorithm for ABR conformance, its formalization in Coq and the main tools which have been developed to this aim: an abstract model of "real-time" that makes no assumption on the nature of time; and a way to actually find proofs employing transitivity, using only logical definitions and EAuto. It is organized as follows. We discuss in section 2 the axioms about time used in the development and proof search using them. Section 3 provides an axiomatic specification of the desired algorithm. A useful purely functional program is also given there. Section 4 describes a representation in Coq of the standardized algorithm and sketches some correctness proofs. Finally, the behaviors of a complete system made of the algorithm immersed in its environment are considered in section 5. The algorithm of I.371 for ABR conformance is given in appendix A.

2 Reasonning about Real Time

In our context, time is a linearly ordered structure equipped with an addition and a subtraction. One may ask whether time should be given a discrete or a continuous—or, at least, dense[2]—representation. On the one hand, "real" time is continuous. On the other, we deal with digital systems, which are discrete in essence. But we will introduce a notion of observer, which may well be able to observe (at least a stable part of) the state at an arbitrary instant.

We decided not to choose: we introduce a type parameter DD (for dates and durations) and operations on DD with a small number of axioms. These axioms do not say anything about the discrete or dense nature of time. A special attention has been paid for designing them, in order that:

- they are consistent (we formally proved that they are satisfied on nat);
- they have other models than $\mathbb{N}$, for instance rational numbers;
- they are strong enough for our needs.

[2] We need a decidable equality, which is available on rational numbers.

2.1 A Small Theory of Dates and Durations

We consider that dates and duration share a common type, DD. Distinguishing two types (respectively Da and Du) would have been more accurate. For instance, we could introduce an addition of type $\mathsf{Du} \to \mathsf{Du} \to \mathsf{Du}$ and another of type $\mathsf{Da} \to \mathsf{Du} \to \mathsf{Da}$, but no of type $\mathsf{Da} \to \mathsf{Da} \to \mathsf{Da}$, because adding two dates makes no sense. Moreover, addition of type $\mathsf{Du} \to \mathsf{Du} \to \mathsf{Du}$ can be considered as commutative, while commutativity has no meaning on an addition of type $\mathsf{Da} \to \mathsf{Du} \to \mathsf{Da}$. However this distinction would lead us to duplicate most operations, axioms and lemmas, whereas the mathematical structures we have in mind (various kind of numbers) are almost the same. In practice, they are exactly the same, because special properties like commutativity of $+$ on Du are not needed in this case study. Hence we considered that $\mathsf{Da} = \mathsf{Du} = \mathsf{DD}$ would be more convenient here. Note that, in the axioms and lemmas below, it is quite easy to recognize whether a variable represents a date or a duration. This disciplin was strictly followed in the whole development.

In order to check that the axioms given below are consistent, we can interpret DD by a concrete inductive type, abstract operations by operations defined on this type, and we prove the formulae obtained by interpreting the axioms. The simplest model for DD is certainly the type of natural numbers. However it does not satisfy properties like $(\forall t, x)\ x - t + t = x$. The following is true on $\mathbb{N}$: $(\forall t, x)\ t \leq x \ \to\ x - t + t = x$, but it is inadequate because we have to consider cases where x is a date and t is a duration: then comparing t with x does not make sense. For instance, we do not want to exclude models with negative values, where x may be negative and t may be positive; but the premise $t \leq x$ would then not be provable. A suitable axiom is given below (A.7). We can then prove $(\forall t, x, z)\ z + t \leq x \ \to\ x - t + t = x$ and other lemmas with similar premises. Those premises do not harm, because we only have to consider dates taking place after an origin, which is roughly the starting date of the algorithm.

Formally, we work in the following context.

DD : Set.

leD : DD → DD → Prop. *(* notation: $x \leq y$ *)*

$\mathsf{ltD}(x, y)$, noted $x < y$, is defined by $x \leq y \wedge \neg(x = y)$. The axioms saying that $\leq$ is a total linear order and that equality is decidable are obvious. A zero in DD is not necessary. When we need to say that a duration τ is positive, we just write $(\forall x)\ x \leq x + \tau$. We give the axioms relating $+$ and $-$ with $\leq$ and $=$. No further axioms are needed.

$$(\forall t, x, y : \mathsf{DD}) \quad x \leq y \to x + t \leq y + t. \tag{A.1}$$

$$(\forall x, t, s : \mathsf{DD}) \quad t \leq s \to x + t \leq x + s. \tag{A.2}$$

$$(\forall t, x : \mathsf{DD}) \quad (x + t) - t = x. \tag{A.3}$$

$$(\forall t, x : \mathsf{DD}) \quad x \leq (x - t) + t. \tag{A.4}$$

$$(\forall t, x, y : \mathsf{DD}) \quad x \leq y \to x - t \leq y - t. \tag{A.5}$$

$$(\forall t, s, x : \mathsf{DD}) \quad s \leq t \to x - t \leq x - s. \tag{A.6}$$

$$(\forall x, y, z, t : \mathsf{DD}) \quad z + t \leq y \to x \leq y - t \to x + t \leq y. \tag{A.7}$$

Then we get a number of lemmas. Some of them are:

$$(\forall t, x, z : \mathsf{DD}) \quad z + t \leq x \to (x - t) + t = x. \tag{L.1}$$
$$(\forall t, x, y : \mathsf{DD}) \quad (x + t) = (y + t) \to x = y. \tag{L.2}$$
$$(\forall t, x, y : \mathsf{DD}) \quad x + t \leq y + t \to x \leq y. \tag{L.3}$$
$$(\forall x, y, z, t : \mathsf{DD}) \quad z + t \leq x \to x < y + t \to x - t < y. \tag{L.4}$$

We think that it would be difficult to find a smaller system of (still understable) axioms. Here are some comments. (A.2) could be avoided in the presence of (A.1), if $+$ was commutative. See at the beginning of this subsection why we reject commutativity. (A.5) and (A.6) would be consequences of (A.2) and (A.1) if we provided a notion of opposite. But we cannot hope to get such a notion if we want $\mathbb{N}$ to be a model of our axioms. This requirement is explicit in (A.4).

Formulae (A.7) and (L.1) are equivalent in presence of other axioms, hence choosing one or the other as an axiom is a matter of taste. More precisely, (L.1) is proved from (A.7) using antisymmetry of $\leq$ and (A.4), while (A.7) is proved from (L.1) using (A.1).

2.2 Proof Search

Typical proof obligations have the form $\Gamma \vdash a < d$ where Γ contains hypotheses $a \leq b$, $c < d$, $b + \tau < u$ and $u \leq c + \tau$, among others. Several tens of similar formulae have to be proved in the case of ABR algorithm. They can be proved directly with the lemmas mentioned above, but this is quite tedious.

We also see that automatic proof search has to find witnesses, because of the heavy use of transitivity. The tactic EAuto makes this possible in Coq. However, it must be used very carefully: four versions of transitivity are available (each of the premises may or not be a strict inequality), and it turns out that the four versions are needed. Reflexivity is also needed. Even a single transitivity rule may be used in two different ways for proving $a < d$ from $a < b$, $b < c$ and $c < d$. Altogether, naive automatic proof search faces a combinatorial explosion.

Following a suggestion of C. Paulin, it is better to work with a better formulation of the lemmas (NB: in this subsection axioms are also considered as lemmas). Intuitively, the set of lemmas used by EAuto can be seen as a Prolog program, and we want this program to be as efficient as possible. Roughly, in our case, hypotheses like $a \leq b$ should be considered as basic facts `arc(a,b)`, and instead of lemmas corresponding to clauses like:

```
le(X,Y) :- arc(X,Y).
le(X,Z) :- le(X,Y) le(Y,Z).
```

with a dangerous left recursion on the second clause, we prefer lemmas corresponding to clauses like:

```
le(X,Y) :- arc(X,Y).
le(X,Z) :- arc(X,Y)  le(Y,Z).
```

Things are a bit more complicated because the conclusion of lemmas like $x + t < y + t \ \rightarrow \ x < y$ must also be considered as a "fact", while $x + t < y + t$ should itself be proved by transitivity. This happens in the example given at the beginning of this subsection: the proof cannot be $a \leq b < u - \tau \leq c < d$, using $b + \tau < u \ \rightarrow b < u - \tau$ and similarly for u and c, but only $a \leq b < c < d$, where $b \leq c$ comes from $b + \tau \leq u \leq c + \tau$. This is so because we banish the use of subtraction as soon as possible, in order to avoid the cycles easily obtained from a combination of lemmas like $x - t \leq y \ \rightarrow \ x \leq y + t$ and $x \leq y + t \ \rightarrow \ x - t \leq y$.

Before entering into more details, let us remark that a brute change of the relations to be used may have an impact on the formulation of the specification. But it is important to keep the latter as simple and natural as possible, if we want to be convinced that we prove the right properties on the right application. Similarly, we want to keep our lemmas as their are stated in the official presentation of the theory, because they give evidence that our axioms are right. Conversely, artificial definitions and lemmas should remain hidden.

Renaming and merging $\leq$ and $<$. In order to stop the use of transitivity in proof search, we introduce a new relation arc such that

$$\mathsf{arc}(\mathsf{true}) = \mathsf{ltD} \quad \text{and} \quad \mathsf{arc}(\mathsf{false}) = \mathsf{leD}. \tag{1}$$

The basic idea is that that goals $x \leq y$ and $x < y$ are proved only by transitivity, while goals $\mathsf{arc}(b, x, y)$ are proved using assumptions, basic lemmas, or anything but transitivity rules. In the sequel, $\mathsf{arc}(\mathsf{true}, x, y)$ and $\mathsf{arc}(\mathsf{false}, x, y)$ are more conveniently noted $x \leq' y$ and $x <' y$.

Let T be (the type of) a theorem to be proved. Let T' be the formula obtained by replacing $\leq$ by $\leq'$ and $<$ by $<'$ in the premises of T. Admittedly, T' is just a harmless (from the point of view of simplicity) rephrasing of T. On the semantical side, any proof of T *is* a proof of T', because T and T' are β-convertible.

However, during the proof process, premises have now the form $\mathsf{arc}(\mathsf{true}, x, y)$ or $\mathsf{arc}(\mathsf{false}, x, y)$. The crucial point is that the proof assistant bases its proof search on the actual shape of the premises and of the goal, not on their normal form. Basic lemmas of section 2.1 are also rephrased, for instance:

$$(\forall t, x, y : \mathsf{DD}) \quad x + t \leq y + t \rightarrow x \leq' y. \tag{L.3'}$$

Changing transitivity and reflexivity rules. As $x \leq' y$ and $x < y$ are considered as "elementary facts", we first just build the transitive closure of their union. Here arc is used with an arbitrary b as first parameter: we want to conclude $a \leq d$ from $a <' b \leq' c$ without taking care of the strictness of elementary inequalities; nrtle stands for "non reflexive transitive closure of arc for less or equal".

Inductive nrtle : DD $\rightarrow$ DD $\rightarrow$ Prop :=
  arc_nrtle : $(\forall b : \mathsf{bool})(x, y : \mathsf{DD})\ \mathsf{arc}(b, x, y) \rightarrow \mathsf{nrtle}(x, y)$ |
  tra_nrtle : $(\forall b : \mathsf{bool})(x, y, z : \mathsf{DD})\ \mathsf{arc}(b, z, y) \rightarrow \mathsf{nrtle}(x, z) \rightarrow \mathsf{nrtle}(x, y)$.

Reflexivity needs to be considered only once on a path.

Inductive tle : DD → DD → Prop :=
  eq_tle : (∀x : DD) tle(x, x) |
  tle_plus : (∀x, y : DD) nrtle(x, y) → tle(x, y).

Now, we have the following theorem:

$$(\forall x, y : \mathsf{DD})\ \mathsf{tle}(x, y) \rightarrow x \leq y. \tag{T.1}$$

The proof uses the fact that $\leq$ is transitive and weaker than $<$. Thanks to this theorem, proving any subgoal $x \leq y$ by transitivity amounts to prove tle(x, y), by the means of eq_tle, tle_plus and then, recursively, arc_nrtle and tra_nrtle. This corresponds to the following algorithm:

- check if $x = y$; if true, we are done;
- if not, try to prove nrtle(x, y); to his effect,
  - first try to find an arc arc(b, x, y) (that is, an assumption saying $x < y$ or $x \leq y$, or a lemma like (L.3') whose conclusion matches the desired values for x and y); if such an arc exists, we are done;
  - if not, try to find an arc arc(b, z, y) for some z (y is fixed but not z);
    - if a suitable z is found, restart the nrtle search procedure using z instead of y;
    - if a suitable z cannot be found, the search fails.

We cope with strict inequalities by following the same lines. Instead of nrtle above, we use tl (transitive transitive closure of arc for less). Here we have to check that a strict inequality occurs at least once on a chain of inequalities.

Mutual Inductive tl : DD → DD → Prop :=
  lt_tl : (∀x, y : DD) $x <' y$ → tl(x, y) |
  tra_tl : (∀b : bool)(x, y, z : DD) arc(b, z, y) → tl_aux(b, x, z) → tl(x, y)
with
  tl_aux : bool→ DD → DD →Prop :=
    tl_aux_true : (∀x, y : DD) nrtle(x, y) → tl_aux(true, x, y) |
    tl_aux_false : (∀x, y : DD) tl(x, y) → tl_aux(false, x, y).

The clause tra_tl reads: if there is b and z such that arc(b, z, y), try to prove tl_aux(b, x, z). Then we have two cases. If $z <' y$ (b = true), it is enough to prove $x \leq y$, that is nrtle(x, y), hence the clause tl_aux_true. If $z \leq' y$ (b = false), we have to prove to prove $x < y$, that is nt(x, y), hence the clause tl_aux_false.

In order to start the corresponding proof search procedure we use the following theorem:

$$(\forall x, y : \mathsf{DD})\ \mathsf{tl}(x, y) \rightarrow x < y. \tag{T.2}$$

Efficiency. Using automatic search proof for inequalities makes scripts much shorter than without this facility. In the case of ABR conformance control, the script for the core of the correctness proof is now half the size of the manual proof given in [9]—which was fairly detailed for reasons given below (6.1).

However, performances must be good enough. They have dramatically increased with the method explained above. Just to give an example, proving $a < d$ from $a \leq b$, $c < d$, $b + t < u$ and $u \leq c + t$ takes 4s with the new procedure on a PC 486 (33 MHz) under linux, against 83s with the old one. Proving $b < d$ from the same hypotheses takes 1s instead of 12s. This is enough for our needs in this case study (less than 4 minutes for the script of the core).

3 Specifications of the Intended Function

First we state an axiomatic characterization of the intended function Acr, which is formalized in 3.1. Then we aim at giving, in 3.3, a much more convenient (for proofs of I.371 algorithm) computational definition of this function: it is the limit of a sequence of functions (Approx_n), where Approx_n depends on Approx_{n-1} and on $\mathsf{ER}(n)$. To this end we give in 3.2 an axiomatic characterization of (Approx_n). The interesting consequence of theorem (T.3) is that $\mathsf{Acr}(t)$ is equal to $\mathsf{Approx}_n(t)$, where n is the number of the last RM cell seen at date t. Loosely speaking, Approx_n is up to date.

Given the sequence of RM cells (ER_i) whose arrival date are respectively (t_i), the desired allowed cell rate at time t is defined by :

$$\mathsf{Acr}(t) = \max\{\mathsf{ER}_i \mid i \in I(t)\}, \tag{2}$$

where I is the interval defined by :

$$i \in I(t) \quad \text{iff} \quad (t - \tau_2 < t_i \leq t - \tau_3) \vee (t_i \leq t - \tau_2 < t_{i+1}) \ . \tag{3}$$

The t_i are taken in increasing order : $t_1 < t_2 < \ldots t_n < \ldots$
The following equivalent characterization of $I(t)$ is easier to handle:

$$i \in I(t) \quad \text{iff} \quad t_i + \tau_3 \leq t < t_{i+1} + \tau_2 \tag{4}$$

The initial (inefficient) ABR conformance control algorithm was a direct computation of Acr according to (2).

3.1 Formalization in Type Theory

Intervals are represented by predicates on nat. We need to characterize the maximum of $\{f(n) \mid P(n)\}$. Such a maximum might not exist, but we still have the uniqueness property. We use an inductive predicate.

Inductive is_max $[f : \mathsf{nat} \to \mathsf{nat}; P : \mathsf{nat} \to \mathsf{Prop}] : \mathsf{nat} \to \mathsf{Prop} :=$
is_max_intro :
$(\forall i : \mathsf{nat})\ P(i) \to$
$((\forall j : \mathsf{nat})\ P(j) \to f(j) \leq f(i)) \to$
$\mathsf{is\_max}(f, P, f(i))$.

The formal specification of the expected value is defined by:

$$\mathsf{In}(t, i) \ = \ (t_i + \tau_3 \leq t) \wedge (t < t_{i+1} + \tau_2).$$

$$\mathsf{is\_ACR}(a) \;=\; \mathsf{is\_max}(\mathsf{ER}, \mathsf{In}, a).$$

The definition of In corresponds to (4) above, but of course we also proved that the formal definition corresponding to (3) is equivalent.

3.2 Approximations of the Ideal Value of Acr

The incremental computation of Acr (t) is intuitively based on the knowledge we have at instant s about t_i and $\mathsf{ER}(i)$. Therefore we consider the n^{th} approximation of $\mathsf{In}(t)$, defined by:

Inductive I_a $[n : \mathsf{nat}; t : \mathsf{DD}; i : \mathsf{nat}] : \mathsf{Prop} :=$
  like_In : $i < n \to \mathsf{In}(t, i) \to \mathsf{I\_a}(n, t, i)$ |
  last_Ia : $i = n \to t_i + \tau_3 \leq t \to \mathsf{I\_a}(n, t, i)$.

and we check that this approximation agrees with In for the current time s (in fact for any t less than $s + \tau_3$). We represent the fact that n is the number of the last RM cell received until s by $\mathsf{current\_last}(s, n)$.

$$\mathsf{current\_last}(s : \mathsf{DD}, n : \mathsf{nat}) \;=\; (t_n \leq s) \wedge (s < t_{n+1}).$$

In an environment containing the hypothesis $\mathsf{current\_last}(s, ns)$, we have the theorem:

$$t \leq s + \tau_3 \to \mathsf{In}(t, i) \leftrightarrow \mathsf{I\_a}(ns, t, i).$$

We can then work with is_Approxn instead of is_ACR:

$$\mathsf{is\_Approxn}(n : \mathsf{nat}, t : \mathsf{DD}) \;=\; \mathsf{is\_max}(\mathsf{ER}, \mathsf{I\_a}(n, t)).$$

Theorem:

$$(\forall a : \mathsf{nat})\ \mathsf{is\_ACR}(t, a) \leftrightarrow \mathsf{is\_Approxn}(ns, t, a). \tag{T.3}$$

We now give a computational definition of $\mathsf{is\_Approxn}(n)$.

3.3 Purely Functional Realization

Computing the $n+1^{\text{th}}$ approximation of ACR from the n^{th} turns out quite simple in the functional setting. First we need the maximum of two natural numbers. It is defined in a symmetrical way.

Definition tot_le : $(\forall n, m : \mathsf{nat})\{n \leq m\} + \{m \leq n\} \quad := \quad \ldots$
Definition $\mathsf{maxb}(n, m) :=$ **Case** $\mathsf{tot\_le}(n, m)$ **of** $[\_]m\ [\_]n$ **end**.

Then we define, in an environment where the date t is of type DD:

Fixpoint Approx $[n{:}\mathsf{nat}] : \mathsf{nat} :=$
  if $n = 0$ **then** $\mathsf{ER}(0)$
  else **if** $t < t_n + \tau_3$ **then** $\mathsf{Approx}(n - 1)$

else
if $t < t_n + \tau_2$ **then** maxb(Approx $(n-1)$, ER(n))
else ER(n)

Using Ocaml, for instance, we could propose the following implementation for ABR conformance control:

```
(* initially *)
let ACR = ref fun t ->(ER 0)

(* when a new RM cell (ER n) is received, at (t n) *)
ACR:= let oldA= !ACR in fun t ->
        if t < t n + tau3 then oldA t
        else if t < t n + tau2 then max (oldA t) (ER n)
        else ER n
```

This definition of ACR is not a realistic implementation for obvious reasons, but Approx is a basic tool in the sequel. We check that Approx has the expected behavior (s_0 represents the starting time of the algorithm):

Theorem:

$$s_0 \leq t \rightarrow (\forall n : nat)\ \mathsf{is\_Approxn}(n, t, \mathsf{Approx}(n)). \tag{T.4}$$

And then, using theorem (T.3):

Main theorem:

$$s_0 \leq t \rightarrow (\forall n : nat)\ \mathsf{current\_last}(t, n) \rightarrow \mathsf{is\_ACR}(t, \mathsf{Approx}(n)). \tag{T.5}$$

The proof of (T.4) uses a lemma stating that the n^{th} approximation is a decreasing function for dates greater than $t_n + \tau_3$.

Theorem Approxn_decrease :

$$(\forall n : \mathsf{nat})(t, t' : \mathsf{DD})\ t_{ns} + \tau_3 \leq t\ \rightarrow\ t \leq t'\ \rightarrow$$
$$(\forall a : \mathsf{nat})\ \mathsf{is\_Approxn}(ns, t, a)\ \rightarrow\ (\forall a' : \mathsf{nat})\ \mathsf{is\_Approxn}(ns, t', a')\ \rightarrow\ a' \leq a.$$

Incremental computations of maxima using the binary version are based on the following theorem.

$$(\forall P : \mathsf{nat} \rightarrow \mathsf{Prop})(\forall n : \mathsf{nat})(\forall m : \mathsf{nat})$$
$$\mathsf{is\_max}(P(m)) \rightarrow \mathsf{is\_max}(\lambda.j : nat\ (P(j) \vee (j = n))\mathsf{maxb}(m, f(n))).$$

4 States and Transitions

The device to be represented (called update in Fig. 1) is an automaton whose states are made of a few variables. One of them, ACR, is instantaneously delivered to DGCRA upon request. It is not difficult to see that if an arbitrarily large

number of RM cells may be received during an interval of length $\tau_2 - \tau_3$, the theoretical value of ACR cannot be computed with a bounded amount of memory. Hence it is only asked that the actual value of ACR is an upper bound of the theoretical value given above. In order to formalize a system made of the device and of an external clock delivering the current time s (the current number of the last RM cell, n, is also needed), and which is able to deliver a suitable value for ACR, we consider a dependent type including having the shape akin to:

Record state : Set := mkstate {
 s : DD;
 n : nat ;
 l_n_s : current_last(s, n) ;
 ACR : nat ; ... *(* other fields for a scheduler *)*
 l_wanted : Approx$(s, n) \leq$ ACR ;
}.

In fact it is a bit more convenient to consider the state at time s, where s is a parameter, and to split l_n_s (see below the next definition of state). The device reacts to two kinds of events.

- External events: a new RM cell is received; then the scheduler is updated.
- Scheduled events (also called internal events); then the field ACR is updated (and the scheduler too).

In each case the device evolves according to the algorithm given in the standard I.371. Such transitions are formalized here by a function from state(s) to state(s'), where the new current time is either the arrival date of a RM cell, or a date programmed in the scheduler. This ensures that the invariant of the system is preserved during its evolution.

4.1 Type of States

The algorithm under study uses a two places scheduler. The first scheduled event is "at tfi (fi stands for first), the value of ACR will be Efi", the last scheduled event (tla, Ela) is similar. Ela happens to contain the value of the last ER cell received. As an optimization trick, Emx contains the maximum of Efi and Ela. The wanted invariant is replaced by l_tfs and l_Ub1 (l_wanted is a consequence of l_tfs and l_Ub1, because we have either tfi $\leq s$ or $s <$ tfi; in the first case, apply l_tfs; in the second case, apply l_Ub1 with $t := s$).

Record state [s:DD] : Set := mkstate {
 n : nat ;
 l_ns : $t_n \leq s$;
 l_sn : $s \leq t_{n+1}$;
 ACR, Efi, Ela, Emx : nat ;
 tfi, tla : DD;
 l_max : Emx = maxb(Efi, Ela);
 l_Ela : Ela = ER(n);

I_fla : tfi $\leq$ tla;
I_lan : tla $\leq tb_n + \tau_2$;
I_tfs : tfi $\leq s \rightarrow (\forall t : \mathsf{DD}) s \leq t \rightarrow \mathsf{Approx}(t, n) \leq \mathsf{ACR}$;
I_Et1 : ACR $<$ Efi $\rightarrow$ tfi $\leq t_n + \tau_3$;
I_Et2 : Efi $<$ Ela $\rightarrow$ tla $\leq t_n + \tau_3$;
I_ttE : $tfi =$ tla $\rightarrow$ Efi $=$ Ela;
I_Ub1 : $(\forall t : \mathsf{DD}) s \leq t \rightarrow t <$ tfi $\rightarrow \mathsf{Approx}(t, n) \leq \mathsf{ACR}$;
I_Ub2 : $(\forall t : \mathsf{DD})$tfi $\leq t \rightarrow t <$ tla $\rightarrow \mathsf{Approx}(t, n) \leq$ Efi;
I_Ub3 : $(\forall t : \mathsf{DD})$tla $\leq t \rightarrow \mathsf{Approx}(t, n) \leq$ Ela
}.

4.2 Updating the State

Let e be the state of the system at date s , n the number of the last RM cell, and let s' be a new date. The algorithm under study computes from e a new value of type state(s'), when s' is either t_{n+1} or tfi(s, e). Formally, we just state and prove a goal of the form state(s'). Then we give the witnesses for ACR(s'), Efi(s'), etc., according to I.371 (see appendix A), and we prove the subgoals corresponding to the preservation of invariants.

For instance, let us sketch the treatment of internal events. The number k is defined as the successor of n. We assume two preconditions, stating that tfi(s, e) is greater or equal to the current time (i.e. the scheduler is not empty) and that the next external event will occur after tfi(s, e). Identifiers XXX_ are an abbreviation for XXX(s, e).

Hypothesis Gi1 : $s\_ \leq'$ tfi_.
Hypothesis Gi2 : tfi_ $\leq' t_k$.

Definition subsi : (state tfi_).
(n ACR Efi Ela Emx tfi tla *)*
Exists n_ Efi_ Ela_ Ela_ Ela_ tla_ tla_ ;
(Discharging various proof obligations *)*
Defined.

The treatment of external events is more complicated, because a number of comparisons are done, for instance between ER(k) and ACR, Efi and Ela; but the principle remains the same.

5 Putting Things Together

A trajectory of the system is an inhabitant of $(\forall s : \mathsf{DD}) s_0 \leq s \rightarrow$ state(s). We cannot construct such a function in a direct way: we have no inductive definition for DD. But we only need to formalize the ability to observe the state of the system at an arbitrary date t.

To this end we iterate updating steps, and we consider a last transition that do not change the state (only time progresses) from the date s of the last external or scheduled event occuring before t, to t.

Transitions dates have one of the forms t_n, $t_n + \tau_3$ and $t_n + \tau_2$, where t_n is the arrival date of a RM cell. Hence it is clear that if n RM cells arrive before s (the observation dates we consider are bounded by dates of RM cells; we could as well suppose that (t_n) is divergent, that is, for any date u, there is a k such that $u < t_k$; then we take $u := s$), the number of transitions before s is bounded by $3n + 1$.

In order to build the desired trajectory we put together

- the transition function $\mathcal{T}$ described in sec 4.2, which computes, for s' a new date, the state at s' from the old state at s,
- a function $\mathcal{E}$ representing the environment of the system, more precisely the progress of time: given the sequence (t_n), the current date s, the date of the next scheduled event if any, taken from the state of the system at s, and may be an observation date, $\mathcal{E}$ returns a new date s' compatible with these constraints. This ensures that the preconditions of $\mathcal{T}$ are satisfied at s'.

The evolution of the whole is then given by a sequence of alternating $\mathcal{E}$ and $\mathcal{T}$ applied to the initial state.

Finally, we define the state of the system at a date s using at most $3n + 1$ iterations of $\mathcal{T} \circ \mathcal{E}$ (n defined as above), and we get the value a of ACR delivered by the system at t. Thanks to conservation of the invariant and to theorem (T.5), we check that a is greater or equal to the theoretical value of ACR, as desired.

6 Concluding Remarks

6.1 Relation between the Manual and the Formal Proof

Recall that the formalized proof follows the lines of the manual proof. We remarked by the end of section 2.2 that the script for the core of the correctness proof is half the size of the original proof given in [9]. Of course, the same proof in standard mathematical style would be even shorter. But [9] is definitely *not* a standard piece of mathematics: the proof given there was intended to be readable with as little effort as possible by experts in ATM, hence almost nothing was left implicit excepted arithmetic laws.

Let us compare both approaches—manual or using a automated proof assistant. In the former, the reader has to check a semi-formal specification (based on the three formulae at the beginning of section 3 and on the components ACR, Efi, Ela, tfi and tla of the state similar to the ones given in 4.1) the reasonning based on the purely functional realization presented in 3.3 and a bunch of 80 more or less boring semi-formal proofs.

In the latter he (she) has only to check the specification (3.1), the definition of the state given in 4.1, including the invariants, the definition of transitions and the statement of the theorem saying that I_tfs and I_Ub1 implies I_wanted). On the other hand, a completely formal notation makes formulae slightly harder to read. The way of representing a transition corresponding to an assignment may be considerd unnatural. This objection is attenuated if the external reader

believes that the Coq user worked on a fair translation of its problem. Moreover, he can check that two Coq users agree on the meaning of the main sentences.

Another advantage of the automatized approach is that studying a new variant of the algorithm can reuse the already developed framework, and even the specification. Finally, recall that the we can be much more confident in the proof checked by a proof tool (especially when the latter is based on the LCF approach) than in a manual proof. As a matter of fact, one of the last 80 proofs of [9] was false! It did not harm so much, because the statement was still provable, but my opinion is now that manual proofs of complicated algorithms should be systematically considered as suspect.

Alltogether, this experience seems to show that using a proof assistant is the most convincing approach.

6.2 Related Work and Future Directions

The general problem we deal with can be stated as follows. Let (e_n) be a sequence of values arriving respectively at the dates (t_n), and let $a(t)$ be a function whose value depends only on $e_0 \ldots e_i$, with $t_0 < \ldots < t_i \leq t$. In real-time applications, we want $a(t)$ to be provided very quickly. An efficient strategy is to compute a new approximation of the future values of a each time a new e_i arrives, and to schedule these values. The problem is then to verify that such an algorithm conforms to the theoretical value of $a(t)$.

The author is not aware of other experiments using formal methods in such situations, though much work has already be done for proving temporal properties of algorithms. Some of the most popular techniques are model checking [3], TLA [8] and Unity [2]. Unfortunately the notion of time used there is logical, not quantitative. In contrast, the algorithm considered here handles time in an explicit way by the means of a scheduler. Further work has been done for specifying distributed algorithms having real-time properties like "such event actually occurs before such date", since the early 1990's [1,6]. Certain classes of such systems can be dealt with model checking techniques (see [7] for instance). The notion of temporized automaton on a dense time considered in [7] inflenced our specification.

We saw in 2.2 that automatizing proofs employing transitivity requires some work. An interesting possibility would have been to consider $\mathsf{DD} = \mathsf{nat}$ in order to take profit of a general tactic like Omega[3], which is able to discharge formulae in Presburger arithmetic. Omega is certainly much more efficient and easy to use than EAuto in our context. But then we would loose the benefits of abstraction mentionned in 2. Let us also remark that it is sometimes useful to understand a proof found by the tool. The written form of the proof terms produced by our technique is exploitable, but is hardly helpful for proofs found by Omega.

The algorithm I.371 has been proposed to other researchers in order to test other formal methods (e.g. temporized automata). This work is going on[4]. On

[3] Omega has been implemented by P. Crégut [4].

[4] This work is supported by action FORMA (MENRT, CNRS, DGA). The work presented in this paper has also been partly done in this framework.

the other hand, we are currently investigating the application of the framework presented here to an algorithm due to Francis Klay, that computes a better approximation of Acr than I.371.

Acknowledgement

The problem has been submitted by Christophe Rabadan, who is the main author of the algorithm. This work has benefited of fruitful discussions with him, Annie Gravey and Francis Klay. Many improvements are due to the comments of anonymous referees.

References

1. R. Alur C. Courcoubetis and D. Dill. Model-Checking for Real-Time Systems. In *5th Symp. on Logic in Compouter Science*. IEEE, 1990.
2. K. M. Chandy and J. Misra. *Parallel Program Design*. Austin, Texas, Addison-Wesley, 1989.
3. D. Clark, E. M. Emerson eand A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: a practical approach. *Proc. 10th ACM Symp. on Principles of Programming Languages*. 1983.
4. B. Barras, S. Boutin, C. Cornes, J. Courant, J-C. Filliâtre, E. Giménez, H. Herbelin, G. Huet, P. Manoury, C. Muñoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi and B. Werner, The Coq Proof Assistant User's Guide, version 6.1 (INRIA-Rocquencourt et CNRS-ENS Lyon, November 1996)
5. ITU-T Recommendation I.371.1 Traffic control and congestion control in B-ISDN, February 1997
6. E. Harel O. Lichtenstein and A. Pnueli. Explicit clock temporal logic. In *5th Symp. on Logic in Compouter Science*. IEEE, 1990.
7. Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems, *Information and Computation*, **111** (1994) 193–244
8. L. Lamport. The temporal logic of actions. ACM Transactions on Programming Languages and Systems, **16-3** (1994), 872–923.
9. Jean-François Monin and Francis Klay Formal specification and correction of I.371.1 algorithm for ABR conformance, internal report NT DTL/MSV/003, CNET, 1997

A Algorithm I.371 for ABR conformance

When real time reaches t_k :

if $t_k <$ tfi **then**
  if Emx $\leq$ ER_k **then** *(* Emx = max(Efi,Ela) *)*
    if tfi $< t_k + \tau_3$ **then**
      if $t_k + \tau_3 <$ tla $\vee$ tfi $=$ tla **then**
        Emx $:=$ ER_k || Ela $:=$ ER_k || tla $:= t_k + \tau_3$ *(* simultaneous *)*
      else *(* assignment *)*
        Emx $:=$ ER_k || Ela $:=$ ER_k
    else
      if ACR $\leq$ ER_k **then**
        Emx $:=$ ER_k || Efi $:=$ ER_k || Ela $:=$ ER_k || tfi $:= t_k + \tau_3$ || tla $:= t_k + \tau_3$
      else
        Emx $:=$ ER_k || Efi $:=$ ER_k || Ela $:=$ ER_k || tla $:=$ tfi
  else
    if $\mathrm{ER}_k <$ Ela **then**
      Efi $:=$ Emx || Ela $:=$ ER_k || tla $:= t_k + \tau_2$
    else
      Efi $:=$ Emx || Ela $:=$ ER_k
else
  if ACR $\leq$ ER_k **then**
    Efi $:=$ ER_k || Ela $:=$ ER_k || Emx $:=$ ER_k || tfi $:= t_k + \tau_3$ || tla $:= t_k + \tau_3$
  else
    Efi $:=$ ER_k || Ela $:=$ ER_k || Emx $:=$ ER_k || tfi $:= t_k + \tau_2$ || tla $:= t_k + \tau_2$

When real time reaches tfi:

ACR $:=$ Efi || tfi $:=$ tla || Efi $:=$ Ela || Emx $:=$ Ela

If tfi $= t_k$, we run the algorithm for tfi, then the algorithm for t_k.