Automatic Parametric Verification of a Root Contention Protocol Based on Abstract State Machines and First Order Timed Logic

Danièle Beauquier, Tristan Crolard, and Evguenia Prokofieva

Laboratory of Algorithmics, Complexity and Logic
University Paris 12 – Val de Marne[⋆](France)
{beauquier, crolard, prokofieva}@univ-paris12.fr

Abstract. The paper presents a verification of the IEEE Root Contention Protocol as an illustration of a new and innovative approach for the verification of real-time distributed systems. Systems are modeled with basic Gurevich abstract state machines (ASMs), and requirements are expressed in a first order timed logic (FOTL). FOTL is undecidable, however the protocol we study is in a decidable class of practical interest. Advantages of this framework are twofold: on the one hand, a great expressive power which permits in particular an easy treatment of parameters, on the other hand the modeling task is simplified by an adequat choice of tools.

Keywords: Parametric verification, real-time distributed systems, predicate logic, IEEE 1394a standard, root contention protocol, Gurevich abstract state machines.

1 Introduction

The IEEE 1394 serial bus protocol (also called"FireWire") is a standard that enables fast transfer of digital information between PCs and peripheral devices such as video recorders, music systems, and so on. The IEEE Root Contention Protocol (RCP), a part of the IEEE 1394 standard, has become a popular case study for investigating the feasibility of formal specification and verification techniques. Its purpose is to elect a leader among two processes, and its correctness depends on the use of randomization and timing delays. The challenge is to synthesize automatically necessary and sufficient conditions on timing parameters for the protocol's correctness.

We apply to this case study a new and innovative approach for the verification of real-time distributed systems which offers some advantages compared to existing frameworks. Our methodology consists in embedding of the whole problem in a first order timed logic (FOTL) and in reducing the problem to the validity of a closed formula in this logic. The process comprises three steps:

[⋆] 61, Av. du Général de Gaulle, 94010 Créteil Cedex France

- *Step 1.* the system under consideration is specified as a Gurevich Abstract State Machine (ASM) and this machine is translated in an automatic way in a formula Φ_{Syst} of FOTL.
- *Step 2.* The requirements specification is written down as a FOTL formula Φ_{Spec}.
- *Step 3.* An automatic verification of the validity of $\Phi_{Syst} \to \Phi_{Spec}$ is applied.

Since validity is undecidable for FOTL, step 3 is not always realizable. In [BS02] decidable classes of this logic with practical interest are described. The fundamental idea which prevails and sustains the philosophy hidden behind these classes is the following observation: in "practical systems" the algorithm which governs the behavior of the system and the requirements to verify are not very complicated, in other words, if there is a counter-model to $\Phi_{Syst} \to \Phi_{Spec}$, there must exist a "simple" one, i.e. a counter-model of bounded complexity. Roughly speaking, the complexity is given by the number of intervals of time where the system has a "constant" behavior.

In [BS02] sufficient semantic conditions for Φ_{Syst} and Φ_{Spec} are given which ensure that if $\Phi_{Syst} \to \Phi_{Spec}$ has a counter-model, then it has a counter-model of a fixed known complexity. The cornerstone is that one can decide whether a FOTL formula has a (counter-)model of a given complexity. The algorithm consists of an elimination of abstract function symbols, which leads to a first order formula of the decidable theory of real addition and multiplication, for which we have to check the satisfiability. At this point, we use a quantifier elimination algorithm implemented in the tool Reduce [Hea99].

In the case of the RCP, we prove (in a simple way) that all the runs of the ASM modeling a cycle of the protocol have a complexity bounded by some constant. So clearly, if the protocol does not satisfy the requirements, there is a run with a bounded complexity which is a counter-model. Then the sufficient conditions given in [BS02] do not need to be proved.

What are the advantages of our approach? A first one is that modeling the system and the requirements does not need much effort and the models we get are very close to the initial specification. As a result, there are less sources of errors in the modeling process. A second advantage is the expressive power of the chosen formalism. FOTL contains arithmetic operations over reals and is much more expressive than classical temporal logics and classical timed automata. Moreover, treatment of parameters is not a difficulty in this framework. Thirdly, for a large class of practical verification problems, the sufficient conditions which lead to a decidable class are satisfied and easy to prove. At last, in case of failure, an efficient description of all the counter-models of a given complexity is provided, which is very important for practical detection of errors.

What are the limits of this method? The limits are in the complexity of *Step 3* in the verification process. The quantifier elimination algorithm has a rather "high" practical complexity and the method can only be applied if the number of quantifiers to eliminate is not too large. To overcome this obstacle we decompose the ASM into smaller ones.

The rest of this paper is organised as follows. Section 2 gives a short description of the Root Contention Protocol. Section 3 presents our logical framework for verification of real-time systems: FOTL syntax and semantics are described as well as basic Gurevich abstract state machines; the notion of finite complexity of a model is introduced. Section 4 is devoted to our modeling of the RCP and to the obtained verification results. In the last section, we compare our techniques to the existing ones and we draw some conclusions.

2 The Root Contention Protocol

The IEEE 1394 standard specifies a high performance serial bus that enables fast data transfer within networks of multimedia systems and devices. It is defined as a multi-layered system using different protocols for different phases. Components connected to the bus are referred to as nodes. Each node has ports for bidirectional connection to some other nodes. The Root Contention Protocol is a sub-protocol of the Initialization phase. The Initialization phase takes place after bus resets. That may occur if a component was added or removed from the network or an error was detected. This phase attemps to find a spanning tree out of the network and then to elect a leader to act as a bus manager. Informally, the leader election works as follows. At initial state, a node waits for a "be my parent" request (parent notification, PN signal) from its neighbors in the spanning tree. When it receives such a request from a neighbor node it replies with a CN signal (Child Notification) as acknowledgement and adds this node to its list of children. When a node has received from all of its ports except one a PN signal, it sends to the remaining neighbor a PN signal. The leaf nodes start to send a PN signal to their only neighbor at the initial state. The tree is constructed bottom-up and at the end, either a leader is elected (the node which has received PN signals from all its neighbors becomes the root), or two nodes are in contention, i.e. each node wants the other one to be a root. Contention is detected independently by each node when a PN request arrives while waiting for a CN acknowledgement.

At this moment the Root Contention Protocol starts. The node sends an Idle signal to its partner and picks a random bit. If the bit 1 comes up, then the node waits for a short time (*fast*) in the interval $[fast\_min,\ fast\_max]$. If the bit 0 comes up, then the node waits for a long time (*slow*) in the interval $[slow\_min,\ slow\_max]$. Of course, $0 < fast\_min \leq fast\_max < slow\_min \leq slow\_max$ holds. When this time is over, the node looks for a message from the contender node. If there is a PN signal, it sends a CN signal and becomes a leader. If there is no message (Idle), it sends a PN signal and waits for acknowledgment. In this case a root contention may reoccur.

The communication between two nodes has a delay with an upper bound *delay*. The choice of timing parameters plays an important role. For the protocol to work correctly constraints on the timing parameters are essential. Figure 1 gives two examples of scenario depending on parameters values.

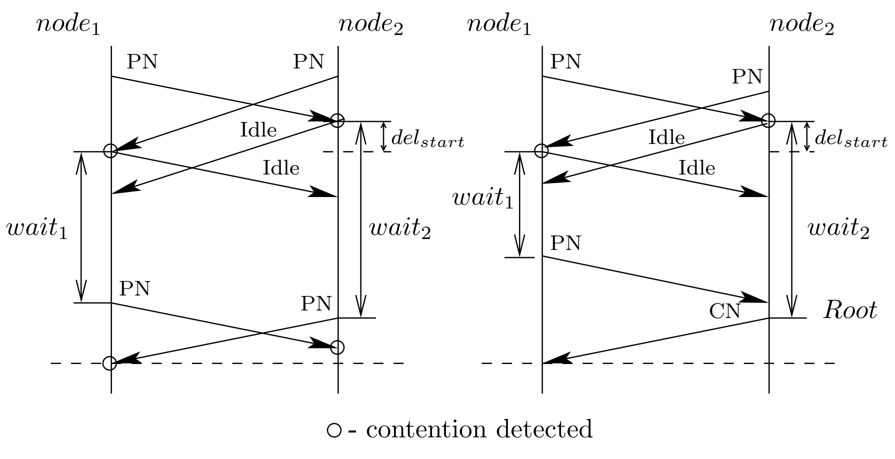

Fig. 1. Two among possible scenarios of the RCP

The aim of verification is to derive automatically necessary and sufficient conditions on timing parameters for a correct protocol behavior, namely eventually, the protocols ends with one and only one node elected as root.

3 First Order Timed Logic (FOTL)

In this section we briefly present the first order logic framework for real-time verification from [BS97], [BS02] and the notion of basic Gurevich ASM [GH96].

3.1 Syntax and Semantics of FOTL

A First Order Timed Logic used in this framework is constructed in two steps. Firstly, we choose a simple, if possible decidable theory to deal with concrete mathematical objects (the underlying theory), like reals, and secondly, we extend it in a 'minimal' way by abstract functions to deal with our specifications. Here we take as the underlying theory the theory of real addition and multiplication, and extend it by functions with at most one time argument and with other arguments being of finite abstract sorts. More details are given below.

Syntax of FOTL

The vocabulary W of a FOTL consists of a finite set of *sorts*, a finite set of *function symbols* and a finite set of *predicate symbols*. To each sort there is attributed a set of variables. Some sorts are predefined, i. e. have fixed interpretations. Here the predefined sorts are the real numbers $\mathbb{R}$, time $\mathcal{T} =_{df} \mathbb{R}_{\geq 0}$ treated as a subsort of $\mathbb{R}$ and $Bool = \{True, False\}$. The other sorts are finite.

Some functions and predicates are also predefined. As predefined constants we take all rational numbers. Addition and multiplication of reals are predefined functions of the vocabulary. The predicates $=$, $\leq$, $<$ over reals are predefined predicates of W. The vocabulary contains $' =' $ for all types of objects, and the identity function *id* of the type $\mathcal{T} \to \mathcal{T}$ to represent current time.

Any *abstract function* (i. e. without any a priori fixed interpretation) is of the type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$, and any *abstract predicate* is of the type $\mathcal{T} \times \mathcal{X} \to Bool$, where $\mathcal{X}$ is a direct product of finite sorts and $\mathcal{Z}$ is an arbitrary sort. The (sub)vocabulary of abstract functions and predicates will be denoted V.

A function symbol is *dynamic* if it has a time argument and *static* otherwise.

A vocabulary W being fixed, the notion of *term* and that of *formula* over W are defined in a usual way.

Semantics of FOTL

A priori we impose no constraints on the admissible interpretations. Thus, the notions of interpretation, model, satisfiability and validity are treated as in first order predicate logic modulo the predefined part of the vocabulary.

3.2 Gurevich Basic Abstract State Machines (ASM)

To represent the algorithms we use Gurevich Abstract State Machine (ASM) [Gur95]. This formalism is powerful, gives a clear vision of semantics of timed algorithms and permits to change easily the level of abstraction. A basic ASM is a program with one external loop consisting of simultaneously executed **If-Then**-operators. In principle, Gurevich ASMs may serve as an intermediate language between user's languages for algorithm specification and a logic framework. (This claim is supported by numerous applications of Gurevich ASM, see http://www.eecs.umich.edu/gasm/.)

A *basic ASM* is a tuple of the form $(W, Init, Prog)$, where W is a vocabulary, $Init$ is a closed formula describing the initial state and $Prog$ is a program.

Sorts, variables and functions are like in subsection 3.1 except that *time cannot be an argument of functions*. We classify the functions using the same terms as in subsection 3.1, namely *abstract* or *predefined* on the one hand and *static* or *dynamic* on the other hand. Dynamic functions are classified into *external* and *internal*.

External functions are not changed by the ASM, internal functions, on the contrary, are computed by the ASM and are obviously abstract and dynamic. Predefined static functions have a fixed interpretation valid for every $t \in \mathcal{T}$. The interpretation of a predefined dynamic function, though changing with time, does not depend on the functioning of the machine. We assume that any ASM vocabulary contains a predefined external dynamic function CT of type $\to \mathcal{T}$ which gives the current time.

The program $Prog$ has the following syntax:

```
Repeat
ForAll $\omega \in \Omega$
  InParallelDo
    If $G_1(\omega)$ Then $A_1(\omega)$ EndIf
    If $G_2(\omega)$ Then $A_2(\omega)$ EndIf
    ...........
    If $G_m(\omega)$ Then $A_m(\omega)$ EndIf
  EndInParallelDo
EndForAll
EndRepeat
```

Ω is an abstract sort which permits to parametrize the ASM. We will not use it in the paper. Each G_i is a *guard*, i. e. a formula over the vocabulary W, not having free variables except ω, and each A_i is a list of assignments (called *updates*) whose terms also do not have free variables except ω. Each assignment is of the form $f(T) := \theta$, where f is an internal function, θ is a term and T is a list of terms of the type corresponding to the type of f.

Informally all guards are checked simultaneously and instantaneously, and all the updates of rules with true guards are executed also simultaneously and instantaneously. To save the space we will not write **Repeat** and **InParallelDo**.

Semantics of an ASM

Precise semantics is given in [BS02] and follows [GH96]. We give here just an intuitive description. For a given interpretation of abstract sorts we define the semantics of the program in terms of runs (executions). Informally, given an input, that is an interpretation of external functions for each moment of time, the machine computes a run which is an interpretation of internal functions for each moment of time or at least, for an initial segment of $\mathcal{T}$. Notice that external functions which are classified as static have the same interpretation for every moment of time.

The behavior of the machine is deterministic. All the **If-Then**-operators are executed simultaneously in parallel and instantaneously as well as all the assignments in any **Then**-part if the corresponding guard is true. Of course, whenever the assignments are inconsistent, the execution is interrupted, and the run of the algorithm becomes undefined thereafter. Notice that the effect of an assignment executed at time t takes place *after* time t but not *at* time t.

We consider only *total runs*, i.e. those defined on whole $\mathcal{T}$. Below “run” means “total run”.

3.3 Translation of an ASM into a FOTL Formula

In order to reason about the behavior of an ASM we are to embed the functions of the vocabulary of the machine into FOTL. And at this moment, time becomes explicit to represent our vision of the functioning of the machine. To ‘time’ the dynamic functions of an ASM we proceed as follows.

If f is a dynamic function of type $\mathcal{X} \to Z$ in the vocabulary of an ASM, the corresponding logical function is denoted by f° and is of type $\mathcal{T} \times \mathcal{X} \to Z$. Thus the predefined dynamic function CT becomes CT° in the logic, that is the identity: $CT^\circ(t) = t$.

It turns out that one can characterize the set of total runs of a basic ASM by an FOTL formula ([BS02]). We generate automatically this formula using the translator that we developed for this purpose[1].

3.4 Finite Complexity

Complexity of finite partial interpretations

A general definition of this notion is given in [BS02]. To treat our case study, we need only abstract functions of type $\mathcal{T} \rightarrow Z$ where Z is a predefined sort. For this reason the definition we give below is limited to this case. The general definition given in [BS02] needs more technical details.

A partial interpretation f^* of an abstract function f over $\mathcal{T}' \subset \mathcal{T}$ is a finite partial interpretation (FPI) with complexity k if $\mathcal{T}'$ is a union of k disjoint intervals and f^* is constant on each interval.

A finite partial interpretation (FPI) of V with complexity k is a collection of FPIs with complexity k, one for each abstract function.

Complexity of interpretations

An interpretation f^* of f has complexity k if $\mathcal{T}$ is a union of k disjoint intervals and f^* is constant on each interval (intervals are arbitrary, in particular they can be reduced to a single point).

An interpretation of V has complexity k if the interpretation of each abstract function from V has complexity k.

3.5 Decision Procedure

In [BS02], the following result is proved and the corresponding algorithm is described:

Theorem 1 *Given an integer $k > 0$, one can decide whether a closed FOTL formula has a (counter)-model with complexity k.*

On the other hand, there exist some semantic sufficient conditions ([BS02]) on formulas Φ and Ψ which ensure that if $\Phi \rightarrow \Psi$ has a counter-model, it has a counter-model of bounded complexity. Formula Φ must be "finitely satisfiable", and formula Ψ must be "finitely refutable". As a matter of fact, it turns out that these conditions are verified by a large class of verification problems. We mention it to underline that our framework is general enough. But we do not need to use these conditions here, because we prove directly that the runs of the ASM modeling the protocol have a complexity equal to 3, so we have only to apply the decision procedure of Theorem 1. This decision procedure works as follows. Given a closed FOTL-formula G and an integer k, there exists a formula G_0 of the decidable theory of real addition and multiplication which is valid iff G has a model of complexity k. The formula G_0 is obtained by an elimination

[1] Available at http://www.univ-paris12.fr/lacl/crolard.

of abstract function symbols using the fact that models of complexity k can be described by a finite set of real parameters. Notice that G_0 depends on G and k. In [BS02] the following result is proved:

Proposition 1 *A closed FOTL formula G has a model of complexity k if and only if G_0 is valid (interpreted over $\mathbb{R}$ with its usual relations, addition and multiplication).*

Thus, starting with a FOTL formula G (of the form $\Phi_{syst} \rightarrow \Phi_{spec}$ in our case), we build the formula G_0 of the theory of real addition and multiplication and we check its validity using an algorithm of elimination of real quantifiers. We use for this elimination the tool Reduce [Hea99]. If the formula G is not closed and contains free parameters, the elimination procedure applied to G_0 gives a quantifier free formula to be satisfied by parameters in order that G is satisfied.

4 Modeling and Verification of RCP

This section falls in three parts. Firstly we present how we use ASMs to model the behavior of the protocol. Secondly we express the requirements in FOTL. Thirdly, we give a report on the results of our model checking experiment.

4.1 The RCP Model as an ASM

Our model describes the behavior of contented nodes during one RCP cycle. First we give below an ASM A which corresponds to the behavior of a node from the moment it enters a contention state. The arrival of signals is modeled by an update of variables representing ports. The probabilistic choice is replaced by a non deterministic one which is modeled by an external boolean function.

The vocabulary of the ASM A consists of

Predefined sorts:

- $Signal = \{\text{PN, CN, Idle}\}$.

There are three possible signals: Idle, PN which is a Parent Notification signal and CN which is a Child Notification signal.

Static functions:

- *delay* is the (maximal) communication delay.
- $fast\_min$, $fast\_max$ are minimal and maximal values of short waiting time.
- $slow\_min$, $slow\_max$ are minimal and maximal values of long waiting time.

All these functions have type $\rightarrow \mathcal{T}$.

External dynamic functions:

- $fast : \rightarrow \mathcal{T}$ and $slow : \rightarrow \mathcal{T}$ are respectively fast and slow waiting times.
- $Alea : \rightarrow Bool$ is a random bit.
- $Port_{rcpt} : \rightarrow Signal$ is the receipt port of the node, the node reads the signals he receives on this port.

Internal dynamic functions:

- $WaitEnd :\rightarrow \mathcal{T}$ is the time when the waiting period of the node stops
- $Port_{snd} :\rightarrow Signal$ is used by the node to send signals to its contention partner. If the node sends a signal at time t, it arrives to its contender node at time $t + delay$. This event is modeled by an update of function $Port_{snd}$ at time $t + delay$.
- $State :\rightarrow Signal$ is the value of the last signal sent by the node.
- $Root :\rightarrow Bool$ is equal to $True$ when the node knows it is root.

Static and external timed functions are subject to the following axiom:

Axiom:
$delay > 0 \ \wedge \ (\forall t \geq 0$
$(0 < fast\_min \leq fast^{\circ}(t) \leq fast\_max < slow\_min \leq slow^{\circ}(t) \leq slow\_max))$.

When a contention phase starts for a node, its state is Idle, and the content of its $Port_{snd}$ is PN (Initial Conditions). Then the behavior of the node in the machine is as follows. It picks (here at time 0) a random bit, this is done by a test on the value of the external function $Alea$. According to the value of the random bit, the value of $Wait$ is put equal to $slow$ or to $fast$ (rules (1) and (1')). The values $slow$ and $fast$ satisfy the constraints $slow\_min \leq slow \leq slow\_max$ and $fast\_min \leq fast \leq fast\_max$ respectively. At the same time, the node sends an Idle signal, which arrives at time $delay$ (rule(2)). When the node ends its waiting period, according to the value of the receipt port ($Port_{rcpt}$), it changes its state (rules (3), (3')). Rules (4) and (4') correspond to the arrival of the signal sent at the end of the waiting period (PN or CN). Later a new contention situation will be detected by the node if it is in a PN state (which means that the last signal it sent is a PN signal), and it receives a PN signal from its partner. Initial conditions of the machine A are:

$Init$:

$State = \text{Idle} \wedge Port_{snd} = \text{PN} \wedge Port_{rcpt} = \text{PN} \wedge Root = false.$

```
(1) If $Alea = \text{true} \wedge CT = 0$ Then $WaitEnd := CT + fast$ EndIf
(1') If $Alea = \text{false} \wedge CT = 0$ Then $WaitEnd := CT + slow$ EndIf
(2) If $State = \text{Idle} \wedge CT = delay$ Then $Port_{snd} := \text{Idle}$ EndIf
(3) If $State = \text{Idle} \wedge Port_{rcpt} = \text{Idle} \wedge CT = WaitEnd$
    Then $State := \text{PN}$ EndIf
(3') If $State = \text{Idle} \wedge Port_{rcpt} = \text{PN} \wedge CT = WaitEnd$
    Then $State := \text{CN}; Root = \text{true}$ EndIf
(4) If $State = \text{PN} \wedge CT = WaitEnd + delay$ Then $Port_{snd} = \text{PN}$ EndIf
(4') If $State = \text{CN} \wedge CT = WaitEnd + delay$ Then $Port_{snd} = \text{CN}$ EndIf
```

ASM A modeling one cycle of a node

The two nodes which are in contention phase are not synchronized, for this reason they do not start contention at the same time (the difference between

starting times is at most equal to *delay*). Machine A describes the behavior of one node during a cycle which starts at time zero. To describe the process the two contender nodes we need two processes which start asynchronously. Let n_i, $i \in \{1,2\}$ be the two contender nodes. For $i \in \{1,2\}$, $\bar{i}$ denotes the value in $\{1,2\}$ different from i. Since the receipt port for one process is the sending port for the other one, we denote by $Port_{i,\bar{i}}$ the sending port of node i and so $Port_{\bar{i},i}$ is its receipt one. Without loss of generality, we can skip step (1-1') and start at step (2) with initial conditions: for $i \in \{1,2\}$ $State_i = \text{Idle}$, $Port_{\bar{i},i} = \text{PN}$, $Root_i = false$ and the waiting period $wait_i$ of node n_i satisfies $fast\_min \le wait_i \le fast\_max \vee slow\_min \le wait_i \le slow\_max$. Machine B describes the behavior of the two nodes during one cycle, where at time zero node n_1 starts its contention ($delay\_start_1 = 0$) and node n_2 is in contention for a duration $delay\_start_2$ where $0 \le delay\_start_2 \le delay$.
In this machine, $wait_i$ and $delay\_start_i$ are static functions of type $\rightarrow \mathcal{T}$. For $i \in \{1,2\}$, function $delay\_start_i$ satisfies $0 \le delay\_start_i \le delay$.

<u>*Init* :</u>

$\forall\, i \in \{1,2\}$
$State_i = \text{Idle} \wedge Port_{i,\bar{i}} = \text{PN} \wedge Root_i = false \wedge delay\_start_1 = 0$

```
ForAll $i \in \{1,2\}$
  (1) If $State_i = \text{Idle} \wedge CT = delay - delay\_start_i$
        Then $Port_i := \text{Idle}$ EndIf
  (2) If $State_i = \text{Idle} \wedge Port_{\bar{i},i} = \text{Idle} \wedge CT = wait_i - delay\_start_i$
        Then $State_i := \text{PN}$ EndIf
  (2') If $State_i = \text{Idle} \wedge Port_{\bar{i},i} = \text{PN} \wedge CT = wait_i - delay\_start_i$
        Then $State_i := \text{CN}$ EndIf
  (3) If $State_i = \text{PN} \wedge CT = wait_i - delay\_start_i + delay$
        Then $Port_i = \text{PN}$ EndIf
  (3') If $State_i = \text{CN} \wedge CT = wait_i - delay\_start_i + delay$
        Then $Port_i = \text{CN}; Root_i = \text{true}$ EndIf
```

ASM B modeling one cycle of the two contender nodes

4.2 The Requirements in FOTL

The aim is to synthesize necessary and sufficient conditions on values of parameters *delay*, $fast\_min$, $fast\_max$, *slow_min*, *slow_max* for which the following statement $\mathcal{S}$ holds:

Eventually, the protocol ends up with one (and only one) node elected as root.
We will denote by $\mathcal{P}$ the list of parameters:

delay, $fast\_min$, $fast\_max$, *slow_min*, *slow_max*.

We consider three properties related to one cycle of the protocol.

Safety: No two roots are elected

Liveness: At least one root is elected

Content: A new contention appears.
The translation of these properties in FOTL is as follows:

$\boxed{\textit{Safety:}\ \neg\,(\exists t\ \mathit{IsRoot}_1(t)\ \wedge\ \exists t\ \mathit{IsRoot}_2(t))}$

where $\mathit{IsRoot}_i(t) =_{def} \mathit{Root}^\circ_i(t)\ \wedge\ \mathit{Port}^\circ_{i,\bar{i}}(t) = \mathrm{CN}\ \wedge\ \mathit{State}^\circ_{\bar{i}}(t) = \mathrm{PN}$.
Actually when a node i becomes root, the other one $\bar{i}$ knows it only when he receives the acknowledgment (the CN signal), moreover, to be sure that this aknowledgment concerns its last PN request and not the previous one, $\bar{i}$ must be in state PN.

$$\boxed{\begin{array}{l}\textit{Liveness:}\ \exists t\ (\mathit{IsRoot}_1(t)\ \vee\ \mathit{IsRoot}_2(t)).\\ \textit{Content:}\ \exists t\ \mathit{Contention}_1(t)\ \wedge \exists t\ \mathit{Contention}_2(t)\end{array}}$$

where $\mathit{Contention}_i(t) =_{def} \mathit{State}^\circ_i(t) = \mathrm{PN}\ \wedge\ \mathit{Port}^\circ_{\bar{i},i}(t) = \mathrm{PN}$.

Actually, a node detects a new contention if it is in state PN and it receives a signal PN.

The machine B is splitted into two cases according to whether random bits have the same value or not.

Case 1 : the two random bits are different. We look for a necessary and sufficient condition $\mathcal{C}_1$ on values of parameters from $\mathcal{P}$ that turn true *Safety* and *Liveness.*

Case 2 : the two random bits are equal. We look for a necessary and sufficient condition $\mathcal{C}_2$ on values of parameters from $\mathcal{P}$ that turn true *Safety* and (*Liveness* or *Content*).

Notice that in Case 2, whatever are the values of parameters in $\mathcal{P}$, there is a chance of root contention at the end of the cycle.

Clearly $\mathcal{C}_1 \wedge \mathcal{C}_2$ is a necessary and sufficient condition on parameters $\mathcal{P}$ such that whatever are values of wait_i and $\mathit{delay\_start}_i$ for $i = 1, 2$, eventually the protocol ends up with one (and only one) node elected as root. Actually if $\mathcal{C}_1 \wedge \mathcal{C}_2$ is satisfied then while the random bits are equal, due to property *Safety* and (*Liveness* or *Content*) either there is again contention or one and exactly one node is root and the process stops. Moreover if it happens that the random bits are different, then, again due to *Safety* and *Liveness* the process stops since one single node is root. As eventually one has different random bits (the probability to never get different random bits is equal to zero) the protocol satisfies the requirement $\mathcal{S}$.

Conversely, if $\mathcal{C}_1 \wedge \mathcal{C}_2$ is not satisfied, then if in a cycle the random bits are different three cases occur:
- the two nodes are root and the protocol is not correct
- no node is root and there is no new contention, the protocol stops and fails
- no node is root and there is a new contention.
Moreover for each cycle with equal random bits, a new contention can occur.
So if $\mathcal{C}_1 \wedge \mathcal{C}_2$ is not satisfied, either two roots are elected, either the protocol stops and fails because no root is elected, or the process can run infinitely entering infinitely often in contention.

We have proved that $\mathcal{C}_1 \wedge \mathcal{C}_2$ is a necessary and sufficient condition for protocol correctness. The next section describes how we get $\mathcal{C}_1 \wedge \mathcal{C}_2$.

4.3 Model-Checking Results

Let Φ_B be the FOTL formula which characterizes the set of runs of machine B. The first step is to prove that every run of machine B has complexity 3. Notice that machine B does not have external dynamic functions. So we have only to look at internal ones. Every guard has an atom of the form $CT = f$, where f is an external static function. So every guard can be true only once: at moment f. Moreover guards with the same $CT = f$ atom, like (2) and (2') for example, "are not true together". Then no more than one guard is true at any moment of time. We have three internal functions whose interpretations are computed by the ASM. Function $State_i$ can be changed by executing rules (2) or (2') at moment $delay$. So no more than one assignment will take place for $State_i$ and the maximal complexity of its interpretations is 2. Function $Root_i$ can be changed by rule (2') at $CT = wait_i - delay\_start_i$, and its complexity is also 2. At last $Port_i$ can be changed by rule (1) at time $delay$, and by rule (3) or (3') at time $wait_i + delay$, so its complexity is 3. The maximal complexity of interpretations is 3, and every model of Φ_B is a model of complexity 3.

As a consequence, for every FOTL formulas Φ, Ψ, the formula $\Phi \wedge \Phi_B \ \rightarrow \ \Psi$ has a counter-model iff it has a counter-model of complexity 3.

One can observe that the requirements depend on the equality or no equality of the random bits got by the two contender nodes. Consider the following formulas:

$A_1 : (fast\_min \leq wait_1 \leq fast\_max \wedge slow\_min \leq wait_2 \leq slow\_max)$
$A_2 : (fast\_min \leq wait_2 \leq fast\_max \wedge slow\_min \leq wait_1 \leq slow\_max)$
$A_3 : (slow\_min \leq wait_1 \leq slow\_max \wedge slow\_min \leq wait_2 \leq slow\_max)$
$A_4 : (fast\_min \leq wait_1 \leq fast\_max \wedge fast\_min \leq wait_2 \leq fast\_max)$

We split the machine B in two machines, one with the axiom $A_1 \vee A_2$ which corresponds to the case when the two bits are different, and a second one with the axiom $A_3 \vee A_4$ which corresponds to the case when the two bits are equal.

Let
$R_1 : Safety$
$R_2 : Liveness$
$R_3 : Liveness \ \vee \ Content$
where $Safety$, $Liveness$, and $Content$ are requirements defined as in subsection 4.2.

The formula $(A_1 \vee A_2) \wedge \Phi_B$ describes the runs with different random bits and the formula $(A_3 \vee A_4) \wedge \Phi_B$ with equal ones. The verification problem of RCP is reduced to compute for what values of parameters from $\mathcal{P}$ the following formulas are valid

$$\forall delay\_start \leq delay \ \forall wait_1 \ \forall wait_2(((A_1 \vee A_2) \wedge \Phi_B) \rightarrow (R_1 \wedge R_2)) \qquad (1)$$

$$\forall delay\_start \leq delay \ \forall wait_1 \ \forall wait_2(((A_3 \vee A_4) \wedge \Phi_B) \rightarrow (R_1 \wedge R_3)) \qquad (2)$$

From the remark made before, we have to compute what values of parameters ensure the existence of counter-models of complexity 3.

The next step is to generate automatically (using our FOTL2REDUCE tool) formulas $\mathcal{C}_j^i$ of the theory of real addition (here we do not need multiplication) for $i = 1, 2,\ j = 1, 2$ or $i = 3, 4,\ j = 1, 3$ such that $\mathcal{C}_j^i$ is valid iff FOTL formula $A_i \wedge \Phi_B \to R_j$ has a counter-model of complexity 3. It is done by applying to formula $\neg(A_i \wedge \Phi_B \to R_j)$ our abstract symbol elimination algorithm. It turns out that the formula $\mathcal{C}_1^i$ returned by the elimination for $i = 1, 2$ is $False$.

The last step is to apply a quantifier elimination algorithm to formulas:
$\forall delay\_start \leq delay\ \forall wait_1\ \forall wait_2 (\neg\mathcal{C}_j^i)$.

We denote the result of this quantifiers elimination by $\widetilde{\mathcal{C}_j^i}$. The conditions $\mathcal{C}_1$ and $\mathcal{C}_2$ can be now written:
$\mathcal{C}_1 = \wedge_{i=1,2,\ j=1,2}\widetilde{\mathcal{C}_j^i}$ and $\mathcal{C}_2 = \wedge_{i=3,4,\ j=1,3}\widetilde{\mathcal{C}_j^i}$

So $\mathcal{C}_1 = \widetilde{\mathcal{C}_2^1} \wedge \widetilde{\mathcal{C}_2^2}$ and $\mathcal{C}_2 = \widetilde{\mathcal{C}_2^1} \wedge \widetilde{\mathcal{C}_2^3}$. Thus formula (1) is valid iff $\mathcal{C}_1$ is true and formula (2) is valid iff $\mathcal{C}_2$ is true.

But it is not enough to get "nice" necessary and sufficient conditions. The obtained results are too big to be easily interpreted. So we experimented with different techniques of simplification realized in Reduce and obtained suitable results by applying iteratively a "tableau simplifier" method. The result we get is a small formula compared to the initial one, nevertheless it is not completely simplified, and we did not succeed to achieve the simplification with Reduce. So the last simplification is done by hand, but it is easy to perform. We get:
$\mathcal{C}_1 :\ 2 * delay < fast\_min \wedge 2 * delay < slow\_min - fast\_max$.
$\mathcal{C}_2 :\ 2 * delay < fast\_min$.

The table below summarizes the performance of our experiments.

property	parameters	time	memory	processor
$Safety$	5	0m6	750M	UltraSparc II (64bits) 440mhg
$Liveness$	5	17m03	3G	UltraSparc III (64bits)[2] (2×)900mhg
$Liveness \vee Content$	5	75m36	3G	UltraSparc III (64bits) (2×)900mhg

The entire process of verification is summarized in Figure 2.The inputs are the ASM which models the system, the complexity k, and the requirements written in FOTL. The ASM2FOTL tool transforms an ASM into an FOTL formula. The FOTL2REDUCE tool proceeds to the elimination of abstract symbol functions and gives the result with the help of Reduce.

[2] We would like to thank J.-M. Moreno from the Computer Science Department of Paris 7 University for allowing us to experiment with this server.

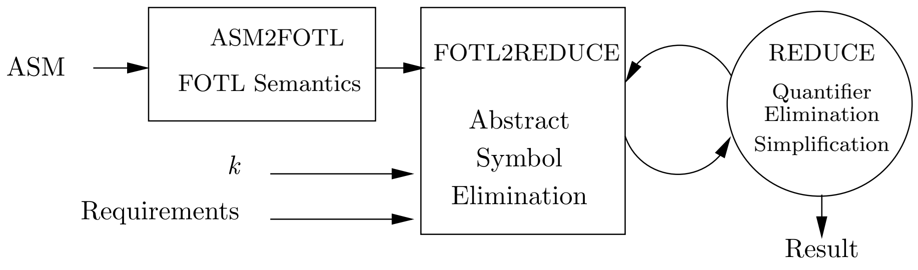

Fig. 2.

5 Comparison with Other Methods – Conclusion

The paper [Sto02] contains a comparative study of formal verification methods applied to the IEEE 1394 RCP. Case studies are divided into three classes: papers that study the functional behaviour of the protocol, papers that employ parametric model checking to synthesize the parameters constraints needed for protocol correctness, and at last papers that focus on the performance analysis of RCP. Our paper belongs to the second class, so we restrict our comparaison to this class [BLSdRT00,TSB00,CAS01,HRSV02].

The modeling

First of all, since the values of probabilities of random bits are not relevant for the analysed properties of Safety and Liveness, the probabilistic part is replaced by a non deterministic one in all the papers.

The four papers use linear parametric timed automata to model the protocol and choose different model checking tools.

The processes which represent the nodes and the wires communicate via synchronization. The abstraction which is done is rather far from the initial specification and it is not always clear whether this modeling is correct, because the behavior of the whole system is not easy to apprehend.

The use of ASMs widely reduces the efforts to accomplish in order to model the protocol. The description of the useful part of the protocol is given by a basic ASM with 5 rules and the properties to verify are very easy to write down in FOTL logic.

Another important aspect is the expressiveness of the models. The tools used in [BLSdRT00,TSB00,HRSV02] can treat only linear constraints, which is enough for RCP, but is a limitation for more general cases. ASMs with the FOTL logic used in the present paper and the TReX tool used in [CAS01] permit non linear arithmetical constraints. Nevertheless, TReX overapproximates the constraints for which a property does not hold, so several runs of the tool with different initial constraints are needed to derive the exact contraints. It is not entirely automatic.

The analysis made in [CAS01] is based on results of [SV99]. In this latter paper, in order to demonstrate that *Impl* (the parallel composition of Nodes and Wires automata) is a correct implementation of *Spec* three intermediate automata are introduced with a laborious stepwise abstraction procedure.

In comparison, our proof is more direct. We have just to verify before the mechanical part that the ASM has a bounded complexity. And this proof is easy.

Parameters treatment

In [BLSdRT00,TSB00] not all five parameters are considered at the same time. Depending on the property to verify, only one or two values are taken as parameters, the other values are fixed constants. In our method, as in [CAS01], the five values are parameters at the same time; a restriction for us is that the delay is always taken as its maximal value.

As a conclusion, we are convinced that our approach to the problem verification is a promissing one as it is demonstrated by this case study, and by a previous one, namely the Generalized RailRoad Crossing Problem [BTE03]. The modeling process of the system with ASMs is direct and easy, FOTL is a powerful logic with which requirements are very easy to write down. Our algorithm provides directly necessary and sufficient conditions for the correctness of the system. We hope to improve the time and the space necessary to the computation by using for quantifiers elimination not a general algorithm but a specific one which should be more efficient.

We do not have for the moment an achieved tool, with a user friendly interface, it is a first experiment. But, in view of these first encouraging and competitive results, we intend to develop this tool and to make more experiments.

References

[BLSdRT00] G. Bandini, R.L. Lutje Spelberg, R.C.H. de Rooij, and W.J. Toetenel. Application of parametric model checking - the root contention protocol. In *Sixth Annual Conference of the Advanced School for Computing and Imaging(ASCI 2000)*, Lommel, Belgium, June 2000.

[BS97] D. Beauquier and A. Slissenko. On semantics of algorithms with continuous time. Technical Report 97–15, Revised version., University Paris 12, Department of Informatics, 1997. Available at http://www.eecs.umich.edu/gasm/ and at http://www.univ-paris12.fr/lacl/.

[BS02] D. Beauquier and A. Slissenko. A first order logic for specification of timed algorithms: Basic properties and a decidable class. *Annals of Pure and Applied Logic*, 113(1–3):13–52, 2002.

[BTE03] D. Beauquier, Crolard T., and Prokofieva E. Automatic verification of real time systems: A case study. In *Third Workshop on Automated Verification of Critical Systems (AVoCS'2003)*, Technical Report of the University of Southampton, pages 98–108, Southampton (UK), April 2003.

[CAS01] A. Collomb-Annichini and M. Sighireanu. Parametrized reachability analysis of the IEEE 1394 root contention protocol using TReX. In *Proceedings of the Workshop on Real-Time Tools (RT-TOOLS'2001)*, 2001.

[GH96] Y. Gurevich and J. Huggins. The railroad crossing problem: an experiment with instantaneous actions and immediate reactions. In H. K. Buening, editor, *Computer Science Logics, Selected papers from CSL'95*, pages 266–290. Springer-Verlag, 1996. Lect. Notes in Comput. Sci., vol. 1092.

[Gur95] Y. Gurevich. Evolving algebra 1993: Lipari guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–93. Oxford University Press, 1995.

[Hea99] Anthony C. Hearn. Reduce user's and contributed packages manual, version 3.7. Available from Konrad-Zuse-Zentrum Berlin, Germany, February 1999. http://www.uni-koeln.de/REDUCE/.

[HRSV02] T.S. Hune, J.M.T. Romijn, M.I.A. Stoelinga, and F.W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 2002.

[Sto02] M.I.A. Stoelinga. Fun with FireWire: Experiments with verifying the IEEE 1394 root contention protocol. In J.M.T. Romijn S. Maharaj, C. Shankland, editor, *Formal Aspects of Computing*, 2002. accepted for publication.

[SV99] M.I.A. Stoelinga and F.W. Vaandrager. Root contention in IEEE 1394. pages 53–75, Bamberg, Germany, May 1999.

[TSB00] H. Toetenel, R.L. Spelberg, and G. Bandini. Parametric verification of the IEEE 1394a root contention protocol using LPMC. In *Seventh International Conference on Real-Time Systems and Applications (RTCSA'00)*, Cheju Island, South Korea, December 2000.